

# Windows Installer

## Purpose

Microsoft Windows Installer is an installation and configuration service provided with Windows. The installer service enables customers to provide better corporate deployment and provides a standard format for component management. The installer also enables the advertisement of applications and features according to the operating system. For more information, see [Platform Support of Advertisement](#).

This documentation describes Windows Installer 5.0 and earlier versions. Not all the capabilities available in later Windows Installer versions are available in earlier versions. This documentation does not describe versions earlier than Windows Installer 2.0. Installation packages and patches that are created for Windows Installer 2.0 can still be installed by using Windows Installer 3.0 and later.

Windows Installer 3.0 and later, can install multiple patches with a single transaction that integrates installation progress, rollback, and reboots. The installer can apply patches in a specified order regardless of the order that the patches are provided to the system. Patching using Windows Installer 3.0 only updates files affected by the patch and can be significantly faster than earlier installer versions. Patches installed with Windows Installer 3.0 or later can be uninstalled in any order to leave the state of the product the same as if the patch was never installed. Accounts with administrator privileges can use the API of Windows Installer 3.0 and later to query and inventory product, feature, component, and patch information. The installer can be used to read, edit, and replace source lists for network, URL, and media sources. Administrators can enumerate across user and install contexts, and manage source lists from an external process.

Windows Installer 4.5 and later can install multiple installation packages using [transaction processing](#). If all the packages in the transaction cannot be installed successfully, or if the user cancels the installation, the Windows Installer can roll back changes and restore the computer to its original state. The installer ensures that all the packages belonging to a multiple-package transaction are installed or none of the packages are

installed.

Beginning with Windows Installer 5.0, a package can be authored to secure new accounts, Windows Services, files, folders, and registry keys. The package can specify a security descriptor that denies permissions, specifies inheritance of permissions from a parent resource, or specifies the permissions of a new account. For information, see [Securing Resources](#). The Windows Installer 5.0 service can enumerate all components installed on the computer and obtain the key path for the component. For more information, see [Enumerating Components](#). By [Using Services Configuration](#), Windows Installer 5.0 packages can customize the services on a computer. Setup developers can use Windows Installer 5.0 and [Single Package Authoring](#) to develop single installation packages capable of installing an application in either the per-machine or per-user [installation context](#).

## **Where Applicable**

Windows Installer enables the efficient installation and configuration of your products and applications. The installer provides new capabilities to advertise features without installing them, to install products on demand, and to add user customizations.

## **Developer Audience**

This documentation is intended for software developers who want to make applications that use Windows Installer. It provides general background information about installation packages and the installer service. It contains complete descriptions of the application programming interface and elements of the installer database. This documentation also contains supplemental information for developers who want to use a table editor or a package creation tool to make or maintain an installation.

## **Run-Time Requirements**

Windows Installer 5.0 will be released with, and require Windows Server 2008 R2 or Windows 7. Versions earlier than Windows Installer 5.0 were released with Windows Server 2008, Windows Vista, Windows Server 2003, Windows XP, and Windows 2000.

Windows Installer 4.5 requires Windows Server 2008, Windows Vista, Windows XP with Service Pack 2 (SP2) and later, and Windows Server 2003 with Service Pack 1 (SP1) and later.

Windows Installer 4.0 requires Windows Vista or Windows Server 2008. There is no redistributable for installing Windows Installer 4.0 on other operating systems. An updated version of Windows Installer 4.0, which does not add any new features, is available in Windows Vista with Service Pack 1 (SP1) and Windows Server 2008.

Windows Installer 3.1 requires Windows Server 2003, Windows XP, or Windows 2000 with Service Pack 3 (SP3).

Windows Installer 3.0 requires Windows Server 2003, Windows XP, or Windows 2000 with SP3. Windows Installer 3.0 is included in Windows XP with Service Pack 2 (SP2). It is available as a redistributable for Windows 2000 Server with Service Pack 3 (SP3) and Windows 2000 Server with Service Pack 4 (SP4), Windows XP RTM and Windows XP with Service Pack 1 (SP1), and Windows Server 2003 RTM.

[Windows Installer Redistributables](#) are available for Windows Installer 4.5 and earlier versions. There is no redistributable available for Windows Installer 4.0.

Windows Installer 2.0 is contained in Windows Server 2003 and Windows XP.

Windows Installer 2.0 is available as a package for installing or upgrading to Windows Installer 2.0 on Windows 2000. This package should not be used to install or upgrade Windows Installer 2.0 on Windows Server 2003 and Windows XP.

You can find all the available Windows Installer redistributables at the [Microsoft Download Center](#).

## In This Section

Topic	Description
<a href="#">Roadmap</a>	A guide to Windows Installer documentation.
<a href="#">Overview</a>	General information about the installer.

<a href="#">What's New in Windows Installer</a>	Lists additions and changes to Windows Installer.
<a href="#">Reference</a>	Documentation of Windows Installer functions.
<a href="#">Windows Installer Scripting Examples</a>	Windows Installer examples using script.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Legal Information

## Windows Installer

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

© 2008 Microsoft Corporation. All rights reserved.

Microsoft, BackOffice, JScript, MS-DOS, MSDN, Visual Basic, Visual C++, Win32, Windows, Windows Server, and Windows Vista are trademarks of the Microsoft group of companies.

All other trademarks are property of their respective owners.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Roadmap to Windows Installer Documentation

This documentation is the primary source of reference material for Windows Installer. It provides information about installation packages and the installer service. It also provides complete descriptions of the application programming interface (API) and the elements of the installer database. This documentation also contains a discussion of basic examples of installation and update packages in [Windows Installer Examples](#).

The [Role-based Guide to Windows Installer Documentation](#) is an alternative provided as a guide to readers that prefer to see links to topics organized by professional role and common task scenarios.

For information about Windows Installer newsgroups see also the topic: [Other Sources of Windows Installer Information](#).

For a list of tips on using the Windows Installer, see [Windows Installer Best Practices](#).

The following list describes each section of the installer documentation.

- [About Windows Installer](#) provides an overview of installer capabilities and benefits, such as advertisement, installation-on-demand, resiliency, customization, and component management. This section introduces the concepts of installer components and features, which are essential to understanding how the installer organizes an installation. It also discusses several high-level subjects about installation, such as System Policy, File Versioning Rules, and Rollback Installation.
- [Using Windows Installer](#) discusses a variety of topics, such as a standard method for organizing an application into components that the installer can install or remove from a user's computer; how to download an installation package from the World Wide Web; and using compressed source images.

- [Digital Signatures and Windows Installer](#) describes how digital signatures can be used with packages, transforms, patches, merge modules, and external cabinet files.
- [Assemblies](#) explains how to use Windows Installer to install and manage common language run time and Win32 assemblies.
- [User Interface](#) gives information about the installer's user interface capabilities. Although the installer does not provide a user interface, a package author can keep all the data and logic required to run a fully interactive internal or external user interface in the installation database. The Reference section describes elements of the user interface that are specifiable in the database tables, including dialog boxes, controls, and control events.
- [Standard Actions](#) discusses the standard actions used by the installer in the sequence tables to perform an installation. This information is intended primarily for package developers.
- [Custom Actions](#) describes how to create additional functionality in the installer. Custom actions enable an author of an installation package to extend the capabilities of standard actions by including executables, dynamic-link libraries, and script. This information is intended for package developers who need to perform installation functions not found elsewhere in the installer.
- [Properties](#) gives information about the properties the installer uses during an installation. The About and Using sections give an overview of these global variables and each property is described in the Reference section.
- [Summary Information Stream](#) documents the summary information properties used by the installer. This information is of interest to all developers.
- [Patching and Upgrades](#) discusses using the installer to perform file updates, QFEs, minor updates, product upgrades, and patching.
- [Transforms](#) explains how to alter or customize an installation

database using a database transform and how to generate, secure, and apply transforms.

- [Package Validation](#) discusses using Internal Consistency Evaluators (ICEs) to test the internal consistency of installation packages that are under development.
- [Merge Modules](#) presents a standard for the design of merge modules. This standard should be followed by developers who are creating their own merge modules as well as by developers who plan to use the installer to deliver shared code to their applications.
- [Windows Installer on 64-bit Operating Systems](#) discusses how to use Windows Installer to install and manage installer components designed to run on 64-bit operating systems.
- Windows Installer Examples includes a step-by-step example of creating an installation package with an internal user interface in [An Installation Example](#). For an example of authoring a major upgrade for an existing package, see [An Upgrade Example](#). To learn how a customization transform disables features and adds new resources, see [A Customization Transform Example](#). For an example of creating a patch package that applies a small update to an existing installation package, see [A Small Update Patching Example](#). To learn how to localize an existing installer package, see [A Localization Example](#).
- [Automation Interface](#) provides information to developers who want to use the automation interface of Windows Installer.
- [Installer Functions](#) describes function calls to the installer API. These are the functions that other applications call to access the installer services to install, maintain, or remove applications. The Using sections include discussions about how to request features, initiate installations, and reinstall missing components programmatically. The Reference section is the primary reference material for the installer service functions.

- [Installer Database](#) discusses the installation database. The installer keeps all of the logic and data necessary for an installation in a relational database located in an .msi file. The About section provides an overview with schema diagrams for the major functional groups of tables of the database. The Using section discusses working with the most important of these tables. These sections contain information that is essential to developers who are authoring installation packages or writing package creation tools. The Reference section contains complete reference material for each database table. This section also contains the primary reference for each of the database functions. The database functions are used internally by the installer to access the database and are primarily of interest to developers of installer package creation tools.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Role-based Guide to Windows Installer Documentation

Windows Installer is the recommended solution for the installation and setup of applications on Windows. Therefore, some of the information contained in this SDK will be of interest to a wide range of software development and IT professionals. This section is provided as a guide to readers who prefer to see links to topics organized by professional role and common task scenarios. Because roles can differ greatly between organizations, the following grouping should only be considered as a guide to a location to start searching for the information you need.

- [Application Developers](#)
- [Setup Authors](#)
- [IT Professionals](#)
- [Infrastructure Developers](#)

This documentation is intended for software developers who want to make applications that use Windows Installer. As the primary source of reference material for the installer, the SDK provides information about installation packages and the installer service. It contains complete descriptions of the application programming interface (API) and the elements of the installer database.

For more information, see [Other Sources of Windows Installer Information](#).

## Application Developers

Application developers create applications that call the Windows Installer application programming interface and install Windows installer packages at run time. The Windows Installer can do work in an application such as self-repair and installation-on-demand. Typically, application Developers do the following:

- Enable installation-on-demand of applications at run time from within

another application.

For more information, see the following:

- [Using Installer Functions](#)
  - [Installer Function Reference](#)
  - [Installation-On-Demand](#)
  - [Component Management](#)
  - [Editing Installer Shortcuts](#)
  - **[OLEAdvtSupport Property](#)**
  - [Platform Support of Advertisement](#)
- Enable self-repair of applications by reinstalling components as needed at run time.

For more information, see the following:

- [Using Installer Functions](#)
  - [Installer Function Reference](#)
  - [Resiliency](#)
  - [Source Resiliency](#)
  - [Searching for a Broken Feature or Component](#)
  - [Replacing Existing Files](#)
- Display a user interface to collect user information and configuration preferences the first time an application is installed or run. The user interface must be added by the Setup Author of the Windows Installer package.

For more information, see the following:

- [Using Installer Functions](#)
  - [Initializing an Application](#)
  - [FirstRun Dialog](#)
  - [About the User Interface](#)
- Create applications that use an indirection model to refer to

components with parallel functionality. The qualified component categories must be added by the Setup Author of the Windows Installer package.

For more information, see the following:

- [Qualified Components](#)
- [Using Qualified Components](#)
- Use private and side-by-side assemblies to isolate applications and reduce DLL conflicts.

For more information, see the following:

- [Assemblies](#)
- [Assembly Registry Keys Written by Windows Installer](#)
- [Installing Win32 Assemblies for Side-by-Side Sharing on Windows XP](#)
- [Installing Win32 Assemblies for the Private Use of an Application on Windows XP](#)
- [MsiAssembly Table](#)
- [MsiAssemblyName Table](#)
- [\*\*MsiProvideAssembly\*\*](#)
- [\*\*MsiWin32AssemblySupport Property\*\*](#)
- [\*\*MsiNetAssemblySupport Property\*\*](#)
- [\*\*Isolated Components\*\*](#)
- Prepare the application to install its own comprehensive major upgrades.

For more information, see the following:

- [Patching and Upgrades](#)
- [Major Upgrades](#)
- [\*\*UpgradeCode Property\*\*](#)
- [\*\*Using an UpgradeCode\*\*](#)

- [Preventing an Old Package from Installing Over a Newer Version](#)
- Prepare the application to install its own minor upgrades, small updates, or fixes.

For more information, see the following:

- [Patching and Upgrades](#)
- [Small Updates](#)
- [Minor Upgrades](#)
- Organize application resources into components that can work with the Windows Installer.

For more information, see the following:

- [Windows Installer Components](#)
- [Working with Features and Components](#)
- [Using Transitive Components](#)
- [What happens if the component rules are broken?](#)
- [Organizing Applications into Components](#)
- [Isolated Components](#)
- [Qualified Components](#)

## Setup Authors

Setup Authors create Windows Installer packages (.msi files) that contain the setup logic and information needed to install an application. They typically use authoring tools such as [Orca.exe](#) to populate the Windows Installer database with the setup logic and information. Typically, Setup Authors do the following:

- Determine the functionality available with different Windows Installer versions.

For more information, see the following:

- [Determining the Windows Installer Version](#)
- [Released Versions of Windows Installer](#)
- [What's New in Windows Installer](#)
- Organize application resources into Windows Installer components.

For more information, see the following:

- [Windows Installer Components](#)
- [Organizing Applications into Components](#)
- [Changing the Component Code](#)
- [What happens if the component rules are broken?](#)
- [Windows Installer Examples](#)
- Use third-party Windows Installer package authoring tools or SDK tools such as [Orca.exe](#) to populate an installation database and create a Windows Installer package.

For more information, see the following:

- [Windows Installer Development Tools](#)
- [Installation Package, About the Installer Database](#)
- [Windows Installer File Extensions](#)
- [Database Tables](#)
- [Package Codes](#)
- [Authoring a Large Package](#)
- [Windows Installer on 64-bit Operating Systems](#)
- [Naming Custom Tables, Properties, and Actions](#)
- [OLE Limitations on Streams](#)
- [Column Definition Format](#)
- [Reducing the Size of an .msi File](#)
- Author the Windows Installer database to install files.

For more information, see the following:

- Core Tables Group
- File Tables Group
- File Table
- File Searching
- File Costing
- File Installation
- Companion Files
- File Versioning Rules
- Default File Versioning
- Replacing Existing Files
- Using Cabinets and Compressed Sources
- Removing Stranded Files
- Installing Permanent Components, Files, Fonts, Registry Keys
- FileSFPCatalog Table
- Searching for a File and Creating a Property Holding the File's Path
- Searching for a Directory and a File in the Directory
- Windows Installer Examples
- Author a Windows Installer database that installs a directory structure and folders.

For more information, see the following:

- Core Tables Group
- File Tables Group
- Component Table
- Directory Table
- Using the Directory Table
- Using a Directory Property in a Path
- System Folder Properties

- [CreateFolder Table](#)
  - [LockPermissions Table](#)
  - [MsiLockPermissionsEx Table](#)
  - [Changing the Target Location for a Directory](#)
  - [Windows Installer Examples](#)
- Author a Windows Installer database that installs registry keys.

For more information, see the following:

- [Core Tables Group](#)
  - [Registry Tables Group](#)
  - [Registry Table](#)
  - [Modifying the Registry](#)
  - [Adding or Removing Registry Keys on the Installation or Removal of Components](#)
  - [Adding and Removing an Application and Leaving No Trace in the Registry](#)
  - [Installing Permanent Components, Files, Fonts, Registry Keys](#)
  - [Searching for Existing Applications, Files, Registry Entries or .ini File Entries](#)
  - [Searching for a Registry Entry and Creating a Property Holding the Value of the Registry](#)
  - [Assembly Registry Keys Written by the Windows Installer](#)
  - [Uninstall Registry Key](#)
  - [SelfReg Table](#)
  - [Specifying the Order of Self Registration](#)
  - [Windows Installer Examples](#)
- Author a Windows Installer database that installs services.

For more information, see the following:

- [ServiceInstall Table](#)

- [ServiceControl Table](#)
- [Component Table](#)
- Author a Windows Installer database that installs isolated components or COM components.

For more information, see the following:

- [Registry Tables Group](#)
- [Class Table](#)
- [Complus Table](#)
- [Isolated Components](#)
- [Using Isolated Components](#)
- [Installation of Isolated Components](#)
- [Reinstallation of Isolated Components](#)
- [Removal of Isolated Components](#)
- [Installing a COM Component to a Private Location](#)
- [Make a COM Component in an Existing Package Private](#)
- [Installing a COM+ Application with the Windows Installer](#)
- [Installing a non-COM Component to a Private Location](#)
- [Make a non-COM Component in an Existing Package Private](#)
- Author a Windows Installer database that installs assemblies.

For more information, see the following:

- [MsiAssembly Table](#)
- [MsiAssemblyName Table](#)
- [Assemblies](#)
- [Assembly Registry Keys Written by the Windows Installer](#)
- [Installation of Win32 Assemblies](#)
- Author a Windows Installer database that installs ODBC drivers and translators.

For more information, see the following:

- [ODBCAttribute Table](#)
  - [ODBCDriver Table](#)
  - [ODBCTranslator Table](#)
  - [ODBCDataSource Table](#)
  - [ODBCSourceAttribute Table](#)
- Author a Windows Installer database that installs MIME.

For more information, see the following:

- [MIME Table](#)
  - [Extension Table](#)
  - [Modifying the Registry](#)
- Author a Windows Installer database that installs environment variables.

For more information, see the following:

- [Environment Table](#)
- Author a Windows Installer database that installs shortcuts.

For more information, see the following:

- [Shortcut Table](#)
  - [MsiShortcutProperty Table](#)
  - [Editing Installer Shortcuts](#)
  - [Windows Installer Examples](#)
- Author a Windows Installer database that installs multiple instances of applications.

For more information, see the following:

- [Installing Multiple Instances of Products and Patches](#)
  - [Authoring Multiple Instances with Instance Transforms](#)
  - [Installing Multiple Instances with Instance Transforms](#)
- Specify default feature selection states and options.

For more information, see the following:

- [Core Tables Group](#)
  - [Component Table](#)
  - [Feature Table](#)
  - [FeatureComponents Table](#)
  - [Controlling Feature Selection States](#)
  - [Feature Installation Options Properties](#)
- Specify conditions that must be met to install an application or selected components.

For more information, see the following:

- [Condition Table](#)
  - [LaunchCondition Table](#)
  - [Component Table](#)
  - [Using Properties in Conditional Statements](#)
  - [Conditional Statement Syntax](#)
  - [Conditioning Actions to Run During Removal](#)
  - [Examples of Conditional Statement Syntax](#)
- Author the sequence of actions used to install the application.

For more information, see the following:

- [Using a Sequence Table](#)
- [Installation Procedure Tables Group](#)
- [Sequence Table Detailed Example](#)
- [Actions with Sequencing Restrictions](#)
- [Actions without Sequencing Restrictions](#)
- [Using Properties in Conditional Statements](#)
- [Conditional Statement Syntax](#)
- [Examples of Conditional Statement Syntax](#)
- [Conditioning Actions to Run During Removal](#)

- [Standard Actions](#)
- [Windows Installer Examples](#)
- Prepare the installation package of the application for future upgrades of the application by the Windows Installer service.

For more information, see the following:

- [Patching and Upgrades](#)
- [Preparing an Application for Future Major Upgrades](#)
- [Using an UpgradeCode](#)
- [Upgrade Table](#)
- [\*\*UpgradeCode Property\*\*](#)
- [Preventing an Old Package from Installing Over a Newer Version](#)
- [Changing the Product Code](#)
- [Updating Assemblies](#)
- [Windows Installer Examples](#)
- Troubleshoot Windows Installer packages under development.

For more information, see the following:

- [Package Validation](#)
- [Internal Consistency Evaluators - ICEs](#)
- [Windows Installer Logging](#)
- [Checking the Installation of Features, Components, Files](#)
- [Authoring a Large Package](#)
- [Willogutl.exe](#)
- [Windows Installer Development Tools](#)
- [Validating Merge Modules](#)
- [Validating an Installation Database](#)
- [Validating an Installation Upgrade](#)
- [Searching for a Broken Feature or Component](#)

- [Windows Installer Error Messages](#)
- [Logging of Reboot Requests](#)
- Ensure a secure setup and installation of the application.

For more information, see the following:

- [Guidelines for Authoring Secure Installations](#)
- [Guidelines for Securing Custom Actions](#)
- [Custom Action Security](#)
- [Guidelines for Securing Packages on Locked-Down Computers](#)
- [Authoring a Fully Verified Signed Installation Using Automation](#)
- [A URL-Based Windows Installer Installation Example](#)
- [Authoring the User Interface for Password Input](#)
- [Digital Signatures and Windows Installer](#)
- [Using Windows Installer with UAC](#)
- [User Account Control \(UAC\) Patching](#)
- [Msicert.exe](#)
- **[AdminUser property](#)**
- **[Privileged property](#)**
- **[SecureCustomProperties property](#)**
- Create a user interface to present options to configure the installation and obtain information from the user about the pending installation process.

For more information, see the following:

- [About the User Interface](#)
- [Adding Controls and Text](#)
- [Authoring a ProgressBar Control](#)
- [Authoring Disk Prompt Messages](#)

- [Authoring a Conditional "Please Wait . . ." Message Box](#)
  - [Previewing the User Interface](#)
  - [Adding Text Stored in a Property](#)
  - [MsiSetInternalUI](#)
- Create an external user interface to present a custom user interface to configure the installation and obtain information from the user about the pending installation process.

For more information, see the following:

- [MsiSetExternalUI](#)
  - [Monitoring an Installation Using MsiSetExternalUIRecord](#)
  - [Parsing Windows Installer Messages](#)
  - [Returning Values from an External User Interface Handler](#)
  - [INSTALLUI\\_HANDLER](#)
  - [Handling Progress Messages Using MsiSetExternalUI](#)
  - [Monitoring an Installation Using MsiSetExternalUI](#)
- Set information for the application in **Add/Remove Programs** (ARP.)

For more information, see the following:

- [Configuring Add/Remove Programs with Windows Installer](#)
  - [Adding and Removing an Application and Leaving No Trace in the Registry](#)
  - [Uninstall Registry Key](#)
- Write custom actions to handle setup logic that is not natively supported by Windows Installer.

For more information, see the following:

- [Custom Actions](#)
- [Summary List of All Custom Action Types](#)
- [Guidelines for Securing Custom Actions](#)
- [Custom Action Reference](#)

- [Using a Custom Action to Create User Accounts on a Local Computer](#)
  - [Using a Custom Action to Launch an Installed File at the End of the Installation](#)
  - [Accessing a Database or Session from Inside a Custom Action](#)
  - [Accessing the Current Installer Session from Inside a Custom Action](#)
  - [Changing the System State Using a Custom Action](#)
- Bootstrap the Windows Installer onto a user's computer.

For more information, see the following:

- [Bootstrapping](#)
  - [Instmsi.exe](#)
  - [Internet Download Bootstrapping](#)
  - [Msistuff.exe](#)
  - [A URL-Based Windows Installer Installation Example](#)
  - [Configuring the Setup.exe Resources](#)
  - [Downloading an Installation from the Internet](#)
- Adhere to Active Accessibility guidelines when writing Windows Installer packages.

For more information, see the following:

- [Accessibility](#)
- Prepare for the internationalization of an application setup.

For more information, see the following:

- [Preparing a Windows Installer Package for Localization,](#)
- [Localizing a Windows Installer Package](#)
- [Code Page Handling \(Windows Installer\)](#)
- [Adding Localized Resources](#)

- [A Localization Example](#)
  - [Localizing the Error and ActionText Tables](#)
  - [Localizing Database Columns](#)
  - [Creating a Database with a Neutral Code Page](#)
  - [Code Page Handling of Imported and Exported Tables](#)
  - [Localizing the Language Displayed by Dialogs](#)
  - [Importing Localized Error and ActionText Tables](#)
  - [Updating ProductLanguage and ProductCode Properties](#)
  - [Updating a Summary Information Stream](#)
  - [Qualified Components](#)
  - [UIText Table](#)
  - [Manage Language and Codepage](#)
  - [Checking the Installation Database Code Page](#)
- Create Windows Installer packages for 32-bit and 64-bit platforms.

For more information, see the following:

- [Windows Installer on 64-bit Operating Systems](#)
  - [64-Bit Custom Actions](#)
  - [Using 64-bit Custom Actions](#)
  - [Using 64-bit Merge Modules](#)
- Redistribute shared Windows Installer components and setup logic as merge modules.

For more information, see the following:

- [Merge Modules](#)
  - [Authoring a Language Transform for a Multiple Language Merge Module](#)
  - [Applying a Configurable Merge Module with Customizations](#)
- Schedule or suppress reboots during a Windows Installer installation.

For more information, see the following:

- [System Reboots](#)
- [Logging of Reboot Requests](#)
- Create updates or fixes for an existing application by creating a patch.

For more information, see the following:

- [Creating a Small Update Patch](#)
- [A Small Update Patching Example](#)
- Author a dual-purpose package capable of installing an application either for only the current user or for all users of the computer.

For more information, see the following:

- [Installation Context](#)
- [Single Package Authoring](#)
- [Single Package Authoring Example](#)
- Customize services on the computer using the Windows Installer.

For more information, see the following:

- [Using Services Configuration](#)
- Secure resources on the computer using the Windows Installer.

For more information, see the following:

- Guidelines for Authoring Secure Installations
- [Securing Resources](#)
- Enumerate all components installed on the computer and obtain the key path for the component.

For more information, see the following:

- [Enumerating Components](#)
- Install multiple packages using *transaction processing*.

For more information, see the following:

- [Multiple-Package Installations](#)
- Embed a custom user interface in the Windows Installer package.

For more information, see the following:

- [Using the User Interface](#)
- [Using an Embedded UI](#)

## IT Professionals

IT Professionals and Administrators customize and deploy existing Windows Installer packages. These users repackage setups for existing applications into Windows Installer installation packages, and install and maintain administrative images of Windows Installer installations on networks.

- Customize applications and setup by generating and applying Windows Installer transforms

For more information, see the following:

- [Customization](#)
- [Database Transforms](#)
- [A Customization Transform Example](#)
- [Merges and Transforms](#)
- [Using Transforms to Add Resources](#)
- [Generate a Transform](#)
- [Command Line Options](#)
- [Msitrans.exe](#)
- [Apply a Transform](#)
- [View a Transform](#)
- [View Differences Between Two Databases](#)
- [Patching Customized Applications](#)

- Deploy a Windows Installer installation package, update, or patch.

For more information, see the following:

- [Installing an Application](#)
  - [Patching and Upgrades](#)
  - [Transforms](#)
  - [Installing a Package with Elevated Privileges for a Non-Admin](#)
  - [Applying Major Upgrades by Patching the Local Installation of the Product](#)
  - [Applying Major Upgrades by Installing the Product](#)
  - [Applying Small Updates by Patching the Local Installation of the Product](#)
  - [Applying Small Updates by Reinstalling the Product](#)
  - [Applying Small Updates by Patching an Administrative Image](#)
  - [Patching Initial Installations](#)
  - [Command Line Options](#)
- Troubleshoot Windows Installer packages.

For more information, see the following:

- [Windows Installer Logging](#)
  - [Checking the Installation of Features, Components, Files](#)
  - [Wilogutl.exe](#)
  - [Searching for a Broken Feature or Component](#)
  - [Windows Installer Error Messages](#)
  - [Msicert.exe](#)
- Use scripting to query Windows Installer packages for information about a product and modify the installation.

For more information, see the following:

- [Automation Interface](#)
- [Windows Installer Scripting Examples](#)

- [Using Windows Installer with WMI](#)
- Create and maintain administrative installations.

For more information, see the following:

- [Administrative Installation](#)
- [Command Line Options](#)
- **[AdminProperties Property](#)**
- [Applying Small Updates by Patching an Administrative Image](#)
- [Applying a Patch Package to an Administrative Installation](#)
- [Action Execution Order](#)
- **[IsAdminPackage Property](#)**
- [Order of Property Precedence](#)
- **[AdminProperties Property](#)**
- Make an application available to all users of a computer or to a specified user only.

For more information, see the following:

- [Installation Context](#)
- **[ALLUSERS Property](#)**
- Interpret packages, install products, and configure feature options using a command line.

For more information, see the following:

- [Command Line Options](#)
- [Setting Public Property Values on the Command Line](#)
- [Getting and Setting Properties](#)
- [Reinstalling a Feature or Application](#)
- [Applying Small Updates by Patching the Local Installation of the Product](#)
- [Applying Small Updates by Reinstalling the Product](#)

- [Changing the Target Location for a Directory](#)
- [Applying Small Updates by Patching an Administrative Image](#)
- [Applying Major Upgrades by Installing the Product](#)
- [Configuration Properties](#)
- [Feature Installation Options Properties](#)
- Work with policy to manage access rights and permissions.

For more information, see the following:

- [Machine Policies](#),
- [User Policies](#),
- [Installing a Package with Elevated Privileges for a Non-Admin](#)
- [Advertising a Per-User Application To Be Installed with Elevated Privileges](#)
- [Using a Custom Action to Create User Accounts on a Local Computer](#)
- **[AdminUser Property](#)**
- **[Privileged Property](#)**
- **[EnableUserControl Property](#)**
- **[UserSID Property](#)**
- **[SecureCustomProperties Property](#)**
- Install multiple packages using transaction processing.

For more information, see the following:

- [Multiple-Package Installations](#)
- Embed a custom user interface within a Windows Installer package..

For more information, see the following:

- [Using the User Interface](#)
- [Using an Embedded UI](#)

## **Infrastructure Developers**

Infrastructure Developers can create unified platforms for the deployment and management of software that uses the Windows Installer service. They can use the Windows Installer programming interface to query, manage, and distribute applications, patches, and sources on a system.

- Locate, inventory and query for the state, information, and clients of components.

For more information, see the following:

- [Component-Specific Functions](#)
  - [System Status Functions](#)
  - [Installer Object](#)
  - [Product Object](#)
  - [Patch Object](#)
- Inventory and query for information and the state of products and features.

For more information, see the following:

- [Inventory products and patches](#)
  - [System Status Functions](#)
  - [Product Query Functions](#)
  - [Installer Object](#)
  - [Product Object](#)
  - [Patch Object](#)
- Improve source resiliency by using the Windows Installer to inventory, query, and modify the source list of applications, upgrades, and patches.

For more information, see the following:

- [SOURCELIST Property](#)
- [Source Resiliency](#)
- [Installation and Configuration Functions](#)

- [Installer Object](#)
  - [Product Object](#)
  - [Patch Object](#)
- Improve source resiliency by using the Windows Installer to inventory, query, and modify media sources.

For more information, see the following:

- **[SOURCELIST Property](#)**
  - [Source Resiliency](#)
  - [Installation and Configuration Functions](#)
  - [Product Object](#)
  - [Patch Object](#)
- Inventory and query for information and the state of patches.

For more information, see the following:

- [Inventory products and patches](#)
  - [Installer Function Reference](#)
  - [Patch Object](#)
- Work with policy to manage access rights and permissions.

For more information, see the following:

- [Machine Policies](#)
- [User Policies](#)
- [Installing a Package with Elevated Privileges for a Non-Admin](#)
- [Advertising a Per-User Application To Be Installed with Elevated Privileges](#)
- [Using a Custom Action to Create User Accounts on a Local Computer](#)
- **[AdminUser Property](#)**
- **[Privileged Property](#)**

- **EnableUserControl Property**
- **UserSID Property**
- **SecureCustomProperties Property**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Other Sources of Windows Installer Information

The Windows Installer SDK contains the most complete and current descriptions of the application programming interface (API) and the elements of the Installer database. It is the primary source of developer reference material for the Windows Installer.

The following supplemental information sources may also be helpful to software developers and IT professionals who use the Windows Installer.

## Windows Developer Center

You can check the Windows Developer Center for information about using Windows Installer when building applications for Windows.

- [Windows Developer Center](#)

## Windows Installer Newsgroups

You can read and post to a newsgroup that discusses the Windows Installer. Past newsgroup messages can be searched.

Microsoft Technical Communities provides access to public newsgroups about Windows Installer:

- [Microsoft Public Newsgroups](#)

You can search the available public newsgroups for Windows developer discussions about the Windows SDK and MSI.

## Windows Installer Team Blog

You can read commentary by Microsoft Bloggers on the [MSDN blog site](#).

Read blogs by developers about the Windows Installer.

- [Windows Installer Team Blog](#)

## Windows Installer Technical Chats

You can participate in live MSDN technical chat sessions about the Windows Installer:

- [Technical Chats](#)  
Experts from the Windows Installer team at Microsoft answer questions you ask about Windows Installer.
- [Archive of MSDN Windows Installer Chats](#)  
Transcripts of previous chats about the Windows Installer are archived for you to read at any time.

## TechNet Articles and Newsgroups

You can search Technet for articles and newsgroups about the Windows Installer:

- [Search TechNet Site](#)

## Knowledge Base Articles

You can search the Knowledge Base (KB) for articles about the Windows Installer:

- [Search the Knowledge Base](#)  
Search for both Windows Installer and MSI.

## White Papers

You can read white papers that describe the Windows Installer:

- [Windows Installer: Benefits and Implementation for System Administrators](#)
- [New Features and Design Changes in Windows Installer 2.0](#)
- [Using Software Restriction Policies to Protect Against Unauthorized](#)

## Software

- [Patch Sequencing in Windows Installer version 3.0](#)
- [Software Installation and Maintenance](#)
- [Step-by-Step Guide to Creating Windows Installer Packages and Repackaging Software for the Windows Installer](#)
- [Standardizing the Patch Experience](#)
- [Delta Compression Application Programming Interface](#)

## Online Windows Installer SDK

You can read about Windows Installer online:

- [Windows Installer SDK](#)

**Note** You need to download and install the [Microsoft Windows Software Development Kit \(SDK\)](#) to obtain the development tools described in the [Windows Installer Development Tools](#) section of the documentation.

## Orca Documentation

[Orca](#) is a GUI .msi file editor that is provided with the Windows Installer SDK. It provides full access to the Windows Installer database tables.

A set of Help files are provided with Orca. To obtain the orca.exe and Orca documentation you must download and install the Windows Installer SDK, which is provided as an Orca.msi file. After you install the [Microsoft Windows Software Development Kit \(SDK\)](#), double click the Orca.msi file to install Orca.

**Note** It is not possible to download Orca separately.

Although Orca provides access to all features of the Windows Installer, it is not intended to replace a fully featured package-authoring environment. In many cases, it is easier to create a Windows Installer installation for an application by using one of the commercial package-creation tools available from independent software vendors.

## Windows Installer Software Vendors

You can contact an independent software vendor about tools to create a Windows Installer package for an application. These tools can provide a package authoring environment that may be easier to use than the tools provided in the Windows Installer SDK:

- [Wise Solutions](#)
- [InstallShield](#)
- [InstallAware](#)
- [InstallSite](#) also provides a list of Windows Installer tool vendors.

## Helpful Windows Installer Sites on the Web

You can visit non-Microsoft Web sites that provide useful information about the Windows Installer. The following list identifies some of those sites:

- [InstallSite](#)  
Windows Installer information for developers.
- [Desktop Engineer's Junk Drawer](#) and [AppDeploy.com](#)  
Windows Installer information for IT professionals.
- [Sourceforge.net](#)  
Windows Installer XML (WiX) project on Sourceforge.net.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Windows Installer Best Practices

This section enumerates a list of tips, linked to the main Windows Installer SDK documentation, to help Application Developers, Setup Authors, IT Professionals, and Infrastructure Developers discover best practices for using the Windows Installer:

- [Update the Windows Installer version.](#)
- [Meet the Windows Logo certification requirements.](#)
- [Prepare the package for localization.](#)
- [Update your Windows Installer development tools and documentation.](#)
- [If you decide to repackage a legacy setup application, follow good repackaging practices.](#)
- [Do not try to replace protected resources.](#)
- [Do not depend upon non-critical resources.](#)
- [Use the API to retrieve Windows Installer configuration information.](#)
- [Organize the installation of your application around components.](#)
- [Reduce the size of large Windows Installer packages.](#)
- [If you use custom actions, follow good custom action practices.](#)
- [If you use assemblies, follow good assembly practices](#)
- [Do not ship concurrent installations.](#)
- [Keep package names and package codes consistent.](#)
- [Do not use the SelfReg and TypeLib tables.](#)
- [Provide the option to install without a user interface.](#)
- [Avoid using the AlwaysInstallElevated policy.](#)
- [Enable the DisableMedia policy to limit unauthorized installation.](#)
- [Keep the original package source files secure and available to users.](#)
- [Enable verbose logging on user's computer when troubleshooting deployment.](#)

- Uninstallation leaves the user's computer in a clean state.
- Test packages for both per-user and per-machine installation deployment.
- Plan and test a servicing strategy before shipping the application.
- Reduce the dependency of updates upon the original sources.
- Do not distribute unserviceable merge modules.
- Avoid patching administrative installations.
- Register updates to run with elevated privilege.
- Use the MsiPatchSequence Table to sequence patches.
- Test the installation package thoroughly.
- Fix all validation errors before deploying a new or revised installation package.
- Author a secure installation.
- Use PMSIHANDLE instead of HANDLE

## **Update the Windows Installer version.**

- Use Windows Installer 5.0 on Windows Server 2008 R2 and Windows 7. This is the Windows Installer version provided with the operating system.
- Use Windows Installer 4.5 on Windows Server 2008, Windows Server 2003 with Service Pack 1 (SP1), Windows Vista with Service Pack 1 (SP1), or Windows XP with Service Pack 2 (SP2). For information about obtaining the latest Windows Installer version, see [Windows Installer Redistributables](#).
- Use Windows Installer 3.1 on Windows 2000 with Service Pack 3 (SP3). Windows Installer version 3.1 has features that facilitate better application servicing and [patching](#).
- Many important features were introduced with version 3.0 and are listed in the section [Not Supported in Windows Installer Version 2.0](#). Installation packages and updates that were created for Windows

Installer 2.0 can be installed using Windows Installer 3.0 and later. Patch packages that contain the new tables used by Windows Installer 3.0 can still be applied using earlier versions of Windows Installer but without the Windows Installer 3.0 patching functionality. It is also possible to author patches that explicitly require the Windows Installer 3.0 that cannot be applied by earlier versions of Windows Installer. If a user is unable to update the installer version, ensure that your application or update will be compatible with a future update of the Windows Installer.

- For a list of the Windows Installer features not supported by earlier versions of the Windows installer see [What's New in Windows Installer](#).

## **Meet the Windows Logo certification requirements.**

- Even if you do not intend to submit your application to the logo program, following the logo certification guidelines can help make your Windows Installer package better. For an overview of the logo requirements, and links to specific logo certification programs, see [Windows Installer and Logo Requirements](#).

## **Prepare the package for localization.**

- It is a good practice to prepare for future localization when authoring the original installation package. You can follow the suggested package localization procedure in [Localizing a Windows Installer Package](#).

## **Update your Windows Installer development tools and documentation.**

- The [Windows Installer Development Tools](#) are not redistributable, and you should only use the versions of these tools available from

Microsoft. These are available in the [Windows SDK Components for Windows Installer Developers](#) in the Microsoft Windows Software Development Kit (SDK).

- Several independent software vendors offer tools to create or modify Windows Installer packages. These tools can provide a package authoring environment that may be easier to use than the tools provided in the Windows Installer SDK. You can learn more about these tools from the information resources discussed in [Other Sources of Windows Installer Information](#).
- The capability to build a package from text files may be more intuitive for some developers. The Windows Installer XML (WiX) toolset available on [Sourceforge.net](#) builds Windows installation packages from XML source code.
- The documentation in the [Windows Installer SDK](#) released in the MSDN Online Library is updated the most frequently.
- Use the recent version of [Msizap.exe](#) (version 3.1.4000.2726 or greater) that is available in the Windows SDK Components for Windows Installer Developers for Windows Vista or greater. Lesser versions of Msizap.exe can remove information about all updates that have been applied to other applications on the user's computer. If this information is removed, these other applications may need to be removed and reinstalled to receive additional updates.
- The database table editor [Orca.exe](#) is a database table editor for creating and editing Windows Installer packages and merge modules. It has a basic GUI interface but supports advanced editing of Windows Installer databases. Even if you use another application as your primary development tool, you may find using Orca.exe is convenient when troubleshooting and testing a package.
- See [Other Sources of Windows Installer Information](#) for current Windows Installer information available in blogs, technical chats, newsgroups, technical articles, and websites.

## **If you decide to repackage a legacy setup application, follow good repackaging practices.**

Many application vendors provide native Windows Installer packages for the installation of their products. Software that converts an existing legacy setup application into a Windows Installer package is referred to as a repackaging tool. The whitepaper [Step-by-Step Guide to Creating Windows Installer Packages and Repackaging Software for the Windows Installer](#) describes a commercially available repackaging tool.

Repackaging an existing setup application is not the best development practice. Applications that have been designed from the start to take advantage of Windows Installer features can be easier for users to install and service. If you decide to use a repackaging software the following practices can help you produce a better Windows Installer package.

- Repackaging tools convert legacy installations into a Windows Installer package by taking a picture of a staging system before and after installation. Any registry changes, file changes, or system setting that occurs during the capture process is included in the installation. Configure the hardware and software of the computer used to repackage the installation as close as possible to the intended user's system. Create a separate package for each different hardware configuration. Repackage using a clean staging computer. Remove any unnecessary applications. Stop all unnecessary processes. Close all non-essential system services.
- Always make a copy of the original installation before starting to work on it. Always work on the copy. Never stop a repackager before it finishes building the package. If the repackager damages the package you will still have the original.
- Do not repackage Microsoft software updates into a Windows Installer package. Microsoft releases software updates, such as service packs, as self-extracting files that automatically runs installation. These updates use different installers than the Windows Installer to replace protected Windows resources and cannot be

converted into a Windows Installer package. For information about how to deploy Windows service packs, see the service pack's deployment guide on [Microsoft TechNet](#).

- Do not use a repackaging tool to convert a Windows Installer package into a new package. The Windows Installer adds configuration information to the system as well as application resources. When a repackaging tool compares the system before and after the installation, the repackager misinterprets the configuration information as part of the application. This usually damages the repackaged application. Instead use [customization transforms](#) to modify an existing Windows Installer package, or make an new package. You can create customization transforms using the [Msitrans.exe](#) tool.
- Do not use a repackaging tool to consolidate several Windows Installer packages into a single package. Instead you can use the [Msistuff.exe](#) tool to configure the Setup.exe bootstrap executable to install the packages one after another.
- Make your Windows Installer package so that it can be easily customized by the customer. Global variables used by the Windows Installer during an installation can be set using public [properties](#) or customization transforms. Provide documentation for the use of those properties and practical default values for all customizable values. For information about getting and setting properties see [Using Properties](#). For an example of a customization transform see [A Customization Transform Example](#).

## **Do not try to replace protected resources.**

Windows Installer packages should not attempt to replace protected resources during installation or update. The Windows Installer does not remove or replace these resources because Windows prevents the replacement of essential system files, folders, and registry keys. Protecting these resources prevents application and operating system

failures.

- When running on Windows Server 2008 or Windows Vista, the Windows Installer skips the installation of any file or registry key that is protected by Windows Resource Protection (WRP), the installer enters a warning in the log file, and continues with the remainder of the installation without an error. For information, see [Using Windows Installer and Windows Resource Protection](#).
- WRP is the new name for Windows File Protection (WFP). WRP protects registry keys and folders as well as essential system files. In Windows Server 2003, Windows XP, and Windows 2000, when the Windows Installer encountered a WFP-protected file, the installer would request that WFP install the file. For information, see [Using Windows Installer and Windows Resource Protection](#).

## **Do not depend upon non-critical resources.**

Your installation or update should not depend upon the installation of non-critical resources for the following reasons.

- Custom actions can fail if they depend upon a component belonging to a feature that the user advertises rather than installs.
- Custom actions sequenced before the [InstallFinalize](#) action can fail if they depend on a component containing an assembly that is being installed. The Windows Installer does not commit assemblies to the Global Assembly Cache (GAC) until the InstallFinalize action completes.

## **Use the API to retrieve Windows Installer configuration information.**

The installation of your application or update should not depend upon direct access to Windows Installer configuration information saved on your computer. Instead use the Windows Installer application

programming interface to retrieve configuration information. The location and format of configuration information is managed by the Windows Installer service and can change.

- The Installation and Configuration Functions of the Windows Installer API are described in the [Installer Function Reference](#).
- The [Configuration Properties](#) are described in the Property Reference.
- The automation methods and properties are described in the [Automation Interface Reference](#). The sample script `WiLstPrd.vbs` connects to an Installer object and enumerates registered products and product information. For information see [List Products, Properties, Features, and Components](#).

## **Organize the installation of your application around components.**

The Windows Installer service installs or removes collections of resources referred to as [components](#). Because components are commonly shared, the author of an installation package must follow rules when specifying the components of a feature or application.

- Adhere to the component rules when [Organizing Applications into Components](#) to ensure that new components, or new versions of components, can be installed and removed without damaging other applications. You can follow the procedure described in [Defining Installer Components](#).
- The installer tracks every component by the respective component ID GUID specified in the [Component table](#). It is essential for the operation of the Windows Installer reference-counting mechanism that the component ID GUID is correct. Follow the guidelines in [Changing the Component Code](#).
- If your package must break the component rules, be aware of the possible consequences and ensure that your installation never

installs these components where they may damage components on the user's system. For information see [What happens if the component rules are broken?](#).

- Be aware how the Windows Installer applies the [File Versioning Rules](#) when [Replacing Existing Files](#). The Windows Installer first determines whether the component's key file is already installed before attempting to install any of the files of the component. If the installer finds a file with the same name as the component's key file installed in the target location, it compares the version, date, and language of the two key files and uses file versioning rules to determine whether to install the component provided by the package. If the installer determines it needs to replace the component base upon the key file, then it uses the file versioning rules on each installed file to determine whether to replace the file.

## **Reduce the size of large Windows Installer packages.**

Very large Windows Packages take system resources and can be difficult for users to install. It is good practice to reduce the size of very large Windows Installer Packages by the following methods.

- Compress the files in the installation and store them in a cabinet (.cab) file . The Installer allows the .cab file to be stored as a separate external file, or as a data stream in the MSI package itself. For information see [Using Cabinets and Compressed Sources](#).
- Remove wasted storage space in the .msi file using one of the options discussed in [Reducing the Size of an .msi File](#).
- If your Windows Installer package contains more than 32767 files, you must change the schema of the database. For information see [Authoring a Large Package](#).

**If you use custom actions, follow good custom action practices.**

The Windows Installer has many built-in [standard actions](#) for the installation and maintenance of applications. Developers should attempt to rely upon the standard actions as much as practical, rather than create their own [custom actions](#). However, there are situations where the developer of an installation package finds it necessary to write a custom action.

- Follow the guidelines for [Using Custom Actions](#).
- Follow the [Guidelines for Securing Custom Actions](#). Custom actions that use sensitive information should not write this information into the log. For information see [Custom Action Security](#).
- Custom Actions must not attempt to set a System Restore entry point from within a custom action. For information see [Setting a restore point from a Custom Action](#).
- Return error messages from custom actions and write them to the log to facilitate troubleshooting custom actions. For information see [Error Message Custom Actions](#) and [Returning Error Messages from Custom Actions](#).
- Do not change the system state from an immediate custom action. Custom actions that change the system directly or call another system service must be deferred to the time when the installation script is executed. Every [deferred execution custom action](#) that changes the system state must be preceded by a [rollback custom action](#) to undo the system state change on an installation rollback. For information see [Changing the System State Using a Custom Action](#).
- Custom actions that perform complex installation operations should be an [executable file](#) or [dynamic-link library](#). Limit the use of custom actions based on [scripts](#) to simple installation operations.
- Make the details of what your custom action does to the system easily discoverable to system administrators. Put the details of registry entries and files used by your custom action into a custom

table and have the custom action read from this table. This is demonstrated by the example in [Using a Custom Action to Create User Accounts on a Local Computer](#). For information on adding custom tables to a database, see [Working with Queries](#) and [Examples of Database Queries Using SQL and Script](#).

- A custom actions should not display a dialog box. Custom actions requiring a user interface can use the **MsiProcessMessage** function instead. See [Sending Messages to Windows Installer Using MsiProcessMessage](#).
- Custom actions must not use any of the functions listed on the page: [Functions Not for Use in Custom Actions](#).
- If the installation is intended to run on a terminal server, test that all your custom actions are capable of running on a terminal server. For more information see [TerminalServer](#) property.
- To have a custom action run when a particular patch is uninstalled the custom action must either be present in the original application or be in a patch for the product that is always applied. For more information see [Patch Uninstall Custom Actions](#).
- Custom actions should not use the UI level as a condition for sending error messages to the installer because this can interfere with logging and external messages. For information see [Determining UI Level from a Custom Action](#).
- Use conditional statements and [Conditional Statement Syntax](#) to ensure that your package correctly runs custom actions, does not run a custom actions, or runs alternate custom action when the package is uninstalled. Test that the package works as expected when [uninstalling custom actions](#). For information see [Conditioning Actions to Run During Removal](#)
- If the installation must be capable of being run by users that do not have administrator privileges, test to ensure that all the custom actions can be run with non-administrator privileges. Custom actions

require elevated privileges to modify parts of the system that are not user specific. The installer may run custom actions with elevated privileges if a managed application is being installed or if the system policy has been specified for elevated privileges. Any custom actions that require elevated privileges must include the `msidbCustomActionTypeInScript` and `msidbCustomActionTypeNoImpersonate` [Custom Action In-Script Execution Options](#) in the custom action type. This is demonstrated by the example in [Using a Custom Action to Create User Accounts on a Local Computer](#).

## **If you use assemblies, follow good assembly practices**

If your package uses software [assemblies](#), follow the guidelines for [Adding Assemblies to a Package](#), [Updating Assemblies](#), and [Installing and Removing Assemblies](#).

## **Do not ship concurrent installations.**

[Concurrent Installations](#), also called Nested Installations, install another Windows Installer package during a currently running installation. The use of concurrent installations is not a good practice because they are difficult for customers to service. Patching and upgrading may not work with concurrent installations. The recommended alternative to using concurrent installations is to instead use a setup application and external UI handler to install several Windows Installer packages sequentially.

For more information about using an external UI handler see [Monitoring an Installation Using MsiSetExternalUI](#). For more information about using a record-based external handler, see [Monitoring an Installation Using MsiSetExternalUIRecord](#).

Concurrent installations are sometimes used in controlled corporate environments to install applications that are not intended for the public. Follow these guidelines if you decide to use concurrent installations.

- Do not use concurrent installations to install or update a shipping

product.

- Concurrent installations should not share components.
- An administrative installation should not contain a concurrent installation.
- Integrated ProgressBars should not be used with concurrent installations.
- Resources that are to be advertised should not be installed by a concurrent installation.
- A package that performs a concurrent installation of an application should also uninstall the concurrent application when the parent product is uninstalled. A nested installation exists under the context of the parent product in the Add/Remove Programs in Control Panel.

## **Keep package names and package codes consistent.**

The .msi file can be given any name that helps users identify the package, but the name should not be changed without also changing the product code.

- Give your .msi file a user-friendly name that enables the user to identify the contents of the Windows Installer package.
- The [product code](#) is the principal identification of an application and must change whenever there is a comprehensive update to the application. For information, see [ProductCode](#) and [Changing the Product Code](#). Changing the name of the application's .msi file is considered a comprehensive change and always requires a corresponding change of the product code to maintain consistency.
- The [package code](#) is the primary identifier used by the installer to search for and validate the correct package for a given installation. No two nonidentical .msi files should ever have the same package code. If a package is changed without changing the package code, the installer may not use the newer package if both are still

accessible to the installer. The package code is stored in the [Revision Number Summary](#) Property of the [Summary Information Stream](#).

- Note that letters in product code and package code [GUIDs](#) must all be uppercase.

## **Do not use the SelfReg and TypeLib tables.**

- Installation package authors are strongly advised against using self registration and the [SelfReg](#) table. Instead they should register modules by authoring one or more of the tables in the [Registry Tables Group](#). Many of the benefits of the Windows Installer are lost with self registration because self-registration routines tend to hide critical configuration information. For a list of the reasons for avoiding self registration see [SelfReg table](#).
- Installation package authors are strongly advised against using the [TypeLib](#) table. Instead of using the TypeLib table, register type libraries by using [Registry](#) table. If an installation using the TypeLib table fails and must be rolled back, the rollback may not restore the computer to the same state that existed prior to the rollback.

## **Provide the option to install without a user interface.**

Administrators often prefer to deploy applications within a corporation without requiring user interaction. It is good practice to enable your application to provide the option of being installed with the [user interface level](#) of None.

- Use [public properties](#) for configuration information. Administrators can provide this information on the command-line.
- Do not require that the installation depend upon information gathered from user interaction with dialog boxes. This information is not available during a silent installation.

- Do not automatically restart the user's computer during a silent installation.
- Administrators can set the user interface level when installing by using the [command line option](#) "/q". The user interface level can also be set programmatically with a call to [MsiSetInternalUI](#).

## **Avoid using the AlwaysInstallElevated policy.**

If the [AlwaysInstallElevated](#) policy is not set, applications not distributed by the administrator are installed using the user's privileges and only managed applications get elevated privileges. Setting this policy directs Windows Installer to use system permissions when it installs the application on the system. This method can open a computer to a security risk, because when this policy is set, a non-administrator user can run installations with [elevated](#) privileges and access secure locations on the computer. It is a good practice to use another method than the AlwaysInstallElevated policy when [Installing a Package with Elevated Privileges for a Non-Admin](#) or [Patching Per-User Managed Applications](#).

## **Enable the DisableMedia policy to limit unauthorized installation.**

The [DisableMedia](#) policy can prevent unauthorized installation of applications. When this policy is enabled, users and administrators running a maintenance installation of one product are prevented from using the Browse Dialog to browse media sources, such as CD-ROM, for the sources of other installable products. Browsing for other products is prevented regardless of whether the installation is done with elevated privileges. It is still possible for the user to reinstall the product from media if the user has a correctly labeled media source.

## **Keep the original package source files secure and available to users.**

In some cases the original source of the Windows Installer package may be needed to install-on-demand, repair, or update an application. If the installer is unable to locate an available source, the user is asked to

provide media or to go to a network location containing the needed sources. It is a good practice to ensure that the installer has the sources it needs without having to prompt the user.

- Use [Digital Signatures and External Cabinet Files](#) to ensure that the original sources being used by the installer are secure. An uncompressed source image stored in a public location is not secure.
- Include a complete list of network or URL source paths to the application's installation package in the **SOURCELIST** property.
- Use a Distributed File System (DFS) share for the source path.
- Use the Windows Installer API to retrieve and modify source list information for Windows Installer applications and patches. For information see [Managing Installation Sources](#).
- Use the methods and properties of the **Installer Object**, **Product Object**, and **Patch Object** to retrieve and modify source list information for Windows Installer applications and patches.
- Adhere to points listed in [Preventing a Patch from Requiring Access to the Original Installation Source](#) points to minimize the possibility that your patch will require access to the original sources.
- Store the package source files in a location that is not the system's temporary folder. Windows Installer source files stored in the temporary folder can become unavailable to users.

## **Enable verbose logging on user's computer when troubleshooting deployment.**

[Windows Installer Logging](#) includes a verbose logging option that can be enabled on a user's computer. The information in a verbose log can be helpful when trying to troubleshoot Windows Installer package deployment.

- You can enable verbose logging on the user's computer by using

[Command Line Options](#), the **MsiLogging** property, [Logging policy](#), **MsiEnableLog**, and **EnableLog** method.

- A very useful resource for interpreting Windows Installer log files is [Wilogutl.exe](#). This tool assists the analysis of log files and displays suggested solutions to errors that are found in a log file.
- For more information about interpreting Windows Installer log files, see the white paper available on the TechNet site: [Windows Installer: Benefits and Implementation for System Administrators](#).
- The verbose logging option should be used only for troubleshooting purposes and should not be left on because it can have adverse effects on system performance and disk space. Each time you use the Add/Remove Programs tool in Control Panel, a new file is created.

## **Uninstallation leaves the user's computer in a clean state.**

Application removal is as important as installation. When a Windows Installer package is uninstalled it should leave no useless parts of itself behind on the user's computer.

- If a file that should have been removed from the user's computer remains installed after running an uninstall, the installer may not be removing the component containing the file for one or more of the reasons described in [Removing Stranded Files](#).
- If an application must be registered, author the package to remove registry information when the application is uninstalled. For information see [Adding or Removing Registry Keys on the Installation or Removal of Components](#). If an application is not registered, the application is not listed in the Add or Remove Programs feature in Control Panel and cannot be managed by using the Windows Installer.
- To hide an application from the Add or Remove Programs feature in

Control Panel and still be able to use the Windows Installer to manage the application, follow the guidelines described in [Adding and Removing an Application and Leaving No Trace in the Registry](#).

- Custom actions should be conditioned to run or not as needed upon uninstallation. Different custom actions may need to run on install and uninstall.
- User-specific customization information can be stored in a text file on the computer. This has the advantage that the file can be removed when the application is uninstalled, even if the user of this customization is not currently logged on.

## **Test packages for both per-user and per-machine installation deployment.**

It is good practice to enable customers to decide whether to deploy a package for installation in either the per-machine or per-user [installation context](#).

- Consider whether the application should be available to only particular users or all users of the computer during the development process.
- Test that the package works correctly for both the per-user installation and per-machine installation contexts.
- Make the package easily customizable and let customers decide whether to deploy it per-user or per-machine.

## **Plan and test a servicing strategy before shipping the application.**

You should decide how you intend to service the application before deploying the application for the first time.

- Consider the types of updates you expect to use to service your application in the future. The Windows Installer provides three types

of updates: [Small Update](#), [Minor Upgrade](#), and [Major Upgrades](#). The differences between these are described in [Patching and Upgrades](#) topic.

- Before shipping your application, test that it works as expected after servicing with each update type.

## **Reduce the dependency of updates upon the original sources.**

If the original source files are required to update your application, this can make servicing the application more difficult. The following methods can help reduce the dependence of updates upon the original sources.

- Use a [delta patch](#) to update the baseline versions of your application, such as the RTM version and the service pack versions. Follow the guidelines for using delta patches described in the topic: [Reducing Patch Size](#).
- Follow the recommendations listed in the topic: [Preventing a Patch from Requiring Access to the Original Installation Source](#).

## **Do not distribute unserviceable merge modules.**

Applications should not depend on [merge modules](#) for the installation of component if the owner of the merge module and the owner of the application are different. This can make it difficult to service the application because both owners need to coordinate to update the application or module. Without knowing all the applications that have used the merge module, the owner of the application is unable update the merge module without risking that the update might be incompatible with another application. The owner of the merge module has no direct method for updating Windows Installer packages that have already installed the merge module.

- Consider providing the needed components to users as another Windows Installer installation.

## Avoid patching administrative installations.

Provide on a network an [administrative installation](#) of your application's original Windows Installer package to enable members of a workgroup to install the application. Users of this administrative image should then apply updates to the local instance of the application located on their computer. This keeps users in synchronization with the administrative image. Applying updates to the administrative installation is not recommended for the following reasons.

- The size and latency of the download required for users to obtain an update is increased compared to downloading a patch. The entire updated Windows Installer package and source files must download, recached, and reinstalled.
- Users are unable to [installation-on-demand](#) and repair applications from an updated administrative installation until they recache and reinstall the application.
- Applying a patch to an administrative installation removes the digital signature from the package. An administrator must resign the package. For more information about using digital signatures, see [Digital Signatures and Windows Installer](#).
- Many binary patches target the RTM image of the application and require a previous file version. The local instance of an application installed from an updated administrative installation may not work with other updates. Many binary patch applications can fail.
- Applying a patch to an administrative installation updates the source files and the .msi file but does not stamp the network image with information about the update. Users cannot determine which updates they have received from the administrative installation. This makes it impossible to sequence updates applied on the user side with those already applied on the administrative image side.
- Patches applied to an administrative installation are not [uninstallable patches](#). This can prevent the package code cached on the user's

computer from becoming different than the package code on the administrative installation. If the package code cached on the user's computer becomes different from that on the administrative installation, reinstall the application from the administrative installation and then patch the client computer.

- If you decide to apply small updates by patching an administrative image, follow the guidelines described in the topic: [Applying Small Updates by Patching an Administrative Image](#).

## Register updates to run with elevated privilege.

Beginning with Windows Installer 3.0, it is possible to apply patches to an application that has been installed in a per-user-managed context, after the patch has been registered as having elevated privileges. You cannot apply patches to applications that are installed in a per-user managed context using versions of Windows Installer earlier than version 3.0.

- Use the [SourceListAddSource](#) method or [MsiSourceListAddSourceEx](#) function to register a patch package as having elevated privileges. Follow the guidelines and examples provided in [Patching Per-User Managed Applications](#).
- When running Windows Installer version 4.0 on Windows Vista you can also use [User Account Control \(UAC\) Patching](#) to enable the authors of Windows Installer installations to identify digitally-signed patches that can be applied in the future by non-administrator users. This is only available with the installation of packages in the per-machine installation context (ALLUSERS=1).
- Ensure that least-privilege patching has not been disabled by setting the [MSIDISABLELUAPATCHING](#) property or the [DisableLUAPatching](#) policy.

## Use the MsiPatchSequence Table to sequence patches.

Include a [MsiPatchSequence Table](#) in your package and add patch

sequencing information. Beginning with Windows Installer version 3.0, the installer can use the MsiPatchSequence Table when [installing multiple patches](#) to determine the best patch application sequence. Use the guidelines described in the [Patch Sequencing in Windows Installer version 3.0](#) white paper for defining patch families.

- If practical, specify all patches as belonging to a single patch family. In many cases a single patch family provides enough flexibility to sequence patches. The complexity of authoring increases when multiple patch families are used. Assign a meaningful name to the patch family and assign sequence values in that patch family that increase over time. Follow the [Multiple Patching Example](#) to apply patches in the order in which they have been issued.
- Use the [PatchSequence Table](#) in [Patchwiz.dll](#) to generate the information in MsiPatchSequence Table. The version of PATCHWIZ.DLL that was released with Windows Installer 3.0 can automatically generate patch sequencing information. For more information on how to add a new patch, see [Generating Patch Sequence Information](#). For more information about patch sequencing scenarios, see the whitepaper: [Patch Sequencing in Windows Installer version 3.0](#) .

## **Test the installation package thoroughly.**

Test for correct installation, repair, and removal of your Windows Installer package. You can divide your testing process into the following parts.

- Installation Testing - Test the installation with all possible combinations of application features. Test all types of installation, including Administrative Installation, [Rollback Installation](#), and Installation-On-Demand. Try all possible methods of installation, including clicking on the .msi file, [command line options](#), and installing from the control panel. Test that the package can be installed by users in all possible privilege contexts. Try installing the

package after it has been deployed by all possible methods. Enable [Windows Installer Logging](#) for each test and resolve all errors found in the installer log and event log.

- User Interface Testing - Test the package when installed with all possible [user interface levels](#). Test the package installed with no user interface and with all information provided through the user interface. Ensure the [Accessibility](#) of the user interface and that the user interface functions as expected for different screen resolutions and font sizes.
- Servicing and Repair Testing - Test that the package can handle [Patching and Upgrades](#) delivered by a [Small Update](#), [Minor Upgrade](#), and [Major Upgrades](#). Before deploying the package, write a trial update of each type and try applying it to the original package.
- Uninstall Testing - Verify that when the package is removed it leaves no useless parts of itself behind on the user's computer and that only information belonging to the package has been removed. Reboot the test computer after uninstalling the package and verify the integrity of common system tools and other standard applications. Test that the package can be removed by users in all possible privilege contexts. Test all methods to remove the package, click the .msi file, try the [command line options](#), and try removing the package from the control panel. Enable [Windows Installer Logging](#) for each test and resolve all errors found in the installer log and event log.
- Product Functionality Testing - Ensure that the application functions as expected after the installation, repair, or removal of the package.

## **Fix all validation errors before deploying a new or revised installation package.**

Run [package validation](#) on a new or revised Windows Installer package before attempting to install it for the first time. Validation checks the Windows Installer database for authoring errors. Attempting to install a

package that does not pass validation can damage the user's system.

- You can validate your package using [Orca.exe](#) or [Msival2.exe](#). Both tools are provided with the Windows SDK. Third-party vendors may also incorporate the ICE validation system into their authoring environment.
- You can use the standard set of [Internal Consistency Evaluators - ICEs](#) included in the .cub files provided with the SDK, or customize validation by [Building an ICE](#) and adding it to the .cub file.
- You can use the Evalcom2.dll to implement [Validation Automation](#) for installation packages and merge modules.

## Author a secure installation.

Following these guidelines when developing your package to help maintain a secure environment during installation.

- [Guidelines for Authoring Secure Installations](#)
- [Guidelines for Securing Packages on Locked-Down Computers](#)
- [Guidelines for Securing Custom Actions](#)

## Use PMSIHANDLE instead of HANDLE

The **PMSIHANDLE** type variables is defined in msi.h. It is recommended that your application use the **PMSIHANDLE** type because the installer closes **PMSIHANDLE** objects as they go out of scope, whereas your application must close **MSIHANDLE** objects by calling [MsiCloseHandle](#).

For example, if you use code like this:

```
MSIHANDLE hRec = MsiCreateRecord(3);
```

Change it to:

```
PMSIHANDLE hRec = MsiCreateRecord(3);
```

© 2009 Microsoft Corporation. All rights reserved.

# What's New in Windows Installer

The following pages list changes to the Windows Installer API by Windows Installer version.

This topic identifies the additions and changes that are available in Windows Installer 5.0.

## [What's New in Windows Installer 5.0](#)

These topics identify Windows Installer features that are not supported by earlier versions of the Windows Installer.

[Not Supported in Windows Installer 4.5](#) and earlier versions.

[Not Supported in Windows Installer 4.0](#) and earlier versions.

[Not Supported in Windows Installer 3.1](#) and earlier versions.

[Not Supported in Windows Installer 3.0](#) and earlier versions.

[Not Supported in Windows Installer 2.0](#) and earlier versions.

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# What's New in Windows Installer 5.0

The information in this topic identifies the additions and changes that are available in Windows Installer 5.0.

Windows Installer 5.0 is available for Microsoft Windows 7 and Windows Server 2008 R2. For a complete list of all Windows Installer versions and redistributables, see [Released Versions of Windows Installer](#).

This page is provided as a guide to the documentation. You should refer to the Requirements section on the main reference pages to determine the actual operating system requirements. Parts of the Windows Installer that are not linked to from this page may be available in another version of the Windows Installer. For information about other Windows Installer versions, see [What's New in Windows Installer](#).

## Standard Actions

- [MsiConfigureServices](#)

## Installer Functions

- [MsiEnumComponentsEx](#)
- [MsiEnumClientsEx](#)
- [MsiGetComponentPathEx](#)

## Column Data Types

- [FormattedSDDLText](#)

## Properties

- [MSIFASTINSTALL](#)
- [MSIINSTALLPERUSER](#)

## Database Tables

- [MsiServiceConfig Table](#)
- [MsiServiceConfigFailureActions Table](#)

- MsiShortcutProperty Table
- MsiLockPermissionsEx Table

## ControlEvents

- MsiPrint ControlEvent
- MsiLaunchApp ControlEvent

## Controls

- Hyperlink Control

## Internal Consistency Evaluators - ICEs

- ICE101
- ICE102
- ICE103
- ICE104
- ICE105

## Automation Interface

- Properties of the **Installer Object**
  - **Installer.ClientsEx**
  - **Installer.ComponentsEx**
  - **Installer.ComponentPathEx**
- Properties of the **Components Object**
  - **Components.ComponentCode**
  - **Components.UserSID**
  - **Components.Context**
- Properties of the **Client Object**
  - **Client.ProductCode**

- [Client.ComponentCode](#)
- [Client.UserID](#)
- [Client.Context](#)
- Properties of the [ComponentInfo](#) Object
  - [ComponentInfo.ComponentCode](#)
  - [ComponentInfo.Path](#)
  - [ComponentInfo.State](#)

## Notes

Setup developers can use Windows Installer 5.0 to author a single installation package capable of either per-machine installation or per-user installation of the application. For more information, see [Single Package Authoring](#). The internal consistency evaluator [ICE105](#) checks that the package has been authored to be installed in a per-user context. An application capable of being installed, updated, run, and removed by a standard user without elevation is called a Per-User Application (PUA.) A PUA can provide a better user experience, minimize effects on the system and other users of the computer, and reserves UAC prompting to situations that actually require the elevation of the user's privileges. The Single Package Authoring features of Windows Installer 5.0 can facilitate the development of Per-User Applications.

Services configuration options enable a Windows Installer package to customize the services on a computer. For more information, see [Using Services Configuration](#).

Beginning with Windows Installer 5.0, a Windows Installer package is capable to secure new accounts, Windows Services, files, folders, and registry keys. The MsiLockPermissionsEx table can specify a security descriptor that denies permissions, specifies inheritance of permissions from a parent resource, or specifies the permissions of a new account. For information, see [Securing Resources](#).

Windows Installer 5.0 can enumerate all components installed on the computer and obtain the key path for the component. For more information, see [Enumerating Components](#).

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Not Supported in Windows Installer 4.5

The Windows Installer functions, tables, and properties listed on this page are not supported by Windows Installer 4.5 and earlier versions. The absence of a feature from this list does not guarantee that the feature is supported. See the main documentation to determine which Windows Installer version is required for a particular feature. For information about other Windows Installer versions see [What's New in Windows Installer](#).

Windows Installer 4.5 is available as a redistributable for Windows Server 2008, Windows Vista with Service Pack 1 (SP1), Windows XP with Service Pack 2 (SP2) and later, and Windows Server 2003 with Service Pack 1 (SP1) and later. For a complete list of all Windows Installer versions and redistributables, see [Released Versions of Windows Installer](#).

The following features are not supported in Windows Installer 4.5 and earlier versions.

## Standard Actions

- [MsiConfigureServices](#)

## Installer Functions

- [MsiEnumComponentsEx](#)
- [MsiEnumClientsEx](#)
- [MsiGetComponentPathEx](#)

## Column Data Types

- [FormattedSDDLText](#)

## Properties

- [MSIFASTINSTALL](#)

- **MSIINSTALLPERUSER**

## Database Tables

- MsiServiceConfig Table
- MsiServiceConfigFailureActions Table
- MsiShortcutProperty Table
- MsiLockPermissionsEx Table

## ControlEvents

- MsiPrint ControlEvent
- MsiLaunchApp ControlEvent

## Controls

- Hyperlink Control

## Internal Consistency Evaluators - ICEs

- ICE102
- ICE103
- ICE104
- ICE105

## Automation Interface

- Properties of the **Installer Object**
  - **Installer.ClientsEx**
  - **Installer.ComponentsEx**
  - **Installer.ComponentPathEx**
- Properties of the **Component Object**
  - **Component.ComponentCode**
  - **Component.UserSID**

- **Component.Context**
- Properties of the **Client Object**
  - **Client.ProductCode**
  - **Client.ComponentCode**
  - **Client.UserID**
  - **Client.Context**
- Properties of the **ComponentInfo** Object
  - **ComponentInfo.ComponentCode**
  - **ComponentInfo.Path**
  - **ComponentInfo.State**

## Notes

Windows Installer 4.5 does not support some features that enable a single installation package to be installed in either the per-machine or per-user installation context. For more information, see [Single Package Authoring](#).

Windows Installer 4.5 does not support some services configuration options that can enable a package to customize the services on a computer. For more information, see [Using Services Configuration](#).

Windows Installer 4.5 does not support some features that enable the Windows Installer to secure new accounts, Windows Services, files, folders, and registry keys. For information, see [Securing Resources](#).

Windows Installer 4.5 does not support some features that enable the installation to enumerate all components installed on the computer and obtain the key path for the component. For more information, see [Enumerating Components](#).

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Not Supported in Windows Installer 4.0

The Windows Installer functions, tables, and properties listed on this page are not supported by Windows Installer 4.0 and earlier versions. The absence of a feature from this list does not guarantee that the feature is supported. See the main documentation to determine which Windows Installer version is required for a particular feature. For information about other Windows Installer versions see [What's New in Windows Installer](#).

Windows Installer 4.0 is available for Microsoft Windows Server 2008 and Windows Vista. For a complete list of all Windows Installer versions and redistributables, see [Released Versions of Windows Installer](#).

The following features are not supported in Windows Installer 4.0 and earlier versions.

## Installer Functions

- [MsiBeginTransaction](#)
- [MsiEndTransaction](#)
- [MsiJoinTransaction](#)

## Properties

- [MSIDISABLEEEEUI](#)
- [MSIUNINSTALLSUPERSEDEDCOMPONENTS](#)

## Database Tables

- [MsiEmbeddedChainer table](#)
- [MsiEmbeddedUI table](#)
- [MsiPackageCertificate Table](#)
- [Component table](#)

`msidbComponentAttributesUninstallOnSupersedence`

msidbComponentAttributesShared

- [CustomAction](#)

ExtendedType Column

[Custom Action Patch Uninstall Option](#)

[MsiTransformView<PatchGUID>](#)

msidbCustomActionTypePatchUninstall

[System Policy](#)

- [DisableSharedComponent](#)
- [MsiDisableEmbeddedUI](#)

Callback Function Prototypes

- [EmbeddedUIHandler](#)
- [InitializeEmbeddedUI](#)
- [ShutdownEmbeddedUI](#)

[Internal Consistency Evaluators - ICEs](#)

- [ICE92](#) verifies no component has both the `msidbComponentAttributesPermanent` and `msidbComponentAttributesUninstallOnSupersedence` attributes.

## Notes

Windows Installer 4.0 cannot perform [Multiple Package Installations](#) using *transaction processing*.

Using Windows Installer 4.0 or earlier versions of the installer, small updates and minor upgrades can fail when using the [EnforceUpgradeComponentRules](#) policy or **MSIENFORCEUPGRADECOMPONENTRULES** property because the update removes a component.

A custom user interface cannot be embedded within the Windows Installer package by using the method described in [Using an Embedded](#)

UI.

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Not Supported in Windows Installer 3.1

The Windows Installer functions, tables, and properties listed on this page are not supported by Windows Installer 3.1 and earlier versions. The absence of a feature from this list does not guarantee that the feature is supported. See the main documentation to determine which Windows Installer version is required for a particular feature. For information about other Windows Installer versions see [What's New in Windows Installer](#).

Windows Installer 3.1 is available for Windows Server 2003, Windows XP, or Windows 2000. For a list of all Windows Installer versions and redistributables, see [Released Versions of Windows Installer](#).

The following features are not supported in Windows Installer 3.1 and earlier versions.

## Installer Functions

- [MsiGetPatchFileList](#)

## Properties

- [MSIARPSETTINGSIDENTIFIER](#)
- [MSIDEPLOYMENTCOMPLIANT](#)
- [MSIDISABLERMRESTART](#)
- [MsiLogFileLocation](#)
- [MsiLogging](#)
- [MSIRESTARTMANAGERCONTROL](#)
- [MsiRestartManagerSessionKey](#)
- [MSIRMSHUTDOWN](#)
- [MsiRunningElevated](#)
- [MsiSystemRebootPending](#)

- **MsiTabletPC**
- **MSIUSEREALADMINDETECTION**

### Summary Information Properties

- The **Word Count Summary Property** has new flag bits to specify whether the installation of the package requires elevated privileges.

### System Policy

- **DisableAutomaticApplicationShutdown**
- **DisableLoggingFromPackage**

### Database Tables

- **Shortcut Table**  
New columns: DisplayResourceDLL, DisplayResourceId, DescriptionResourceDLL, and DescriptionResourceId

### Dialog Boxes

- **MsiRMFilesInUse Dialog**

### Control Attributes

- **ElevationShield**

### ControlEvents

- **RmShutdownAndRestart**

### External UI Message Types

- **INSTALLLOGMODE\_RMFILESINUSE**

### Windows Installer on 64-bit Operating Systems

- **msidbComponentAttributesDisableRegistryReflection** attribute in **Component Table**

## Automation Interface

- Methods of **Installer Object**
  - **Installer.AdvertiseProduct**
  - **Installer.AdvertiseScript**
  - **Installer.CreateAdvertiseScript**
  - **Installer.ProvideAssembly**
- Properties of **Installer Object**
  - **Installer.PatchFiles**
  - **Installer.ProductElevated**
  - **Installer.ProductInfoFromScript**

## Notes

The Windows Installer service must run on Windows Vista to enable use of Restart Manager, [User Account Control](#), and [User Account Control \(UAC\) Patching](#). For information, see [Using Windows Installer with Restart Manager](#) and [Using Windows Installer with UAC and User Account Control \(UAC\) Patching](#).

Windows Installer 3.1 supports Windows File Protection (WFP) and does not support Windows Resource Protection (WRP). WRP in Windows Server 2008 and Windows Vista replaces WFP in Windows Server 2003, Windows XP, and Windows 2000. For information about Windows Installer and WFP, see [Using Windows Installer and Windows Resource Protection](#).

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Not Supported in Windows Installer 3.0

The Windows Installer functions, tables, and properties listed on this page are not supported by Windows Installer 3.0 and earlier versions. The absence of a feature from this list does not guarantee that the feature is supported. See the main documentation to determine which Windows Installer version is required for a particular feature. For information about other Windows Installer versions see [What's New in Windows Installer](#).

Windows Installer 3.0 is available for Windows Server 2003, Windows XP, or Windows 2000. For a list of all Windows Installer versions and redistributables, see [Released Versions of Windows Installer](#).

The following features are not supported in Windows Installer 3.0 and earlier versions.

## Installer Functions

- [MsiSetExternalUIRecord](#)
- [MsiNotifySidChange](#)

## Callback Function Prototype

- [INSTALLUI\\_HANDLER\\_RECORD](#)

## Database Tables

- The Value column of the [MsiPatchMetadata Table](#) accepts OptimizedInstallMode and MinorUpdateTargetRTM.

## Properties

- [Msix64 Property](#)

## Windows Installer on 64-bit Operating Systems

- The **Template Summary Property** accepts the value x64.

## Internal Consistency Evaluators - ICEs

- **ICE99**

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Not Supported in Windows Installer 2.0

The Windows Installer functions, tables, and properties listed on this page are not supported by Windows Installer 2.0 and earlier versions. The absence of a feature from this list does not guarantee that the feature is supported. See the main documentation to determine which Windows Installer version is required for a particular feature. For information about other Windows Installer versions see [What's New in Windows Installer](#).

Windows Installer 2.0 can run on Microsoft Windows 2000, Microsoft Windows XP, or Windows Server 2003. For a list of all Windows Installer versions, see [Released Versions of Windows Installer](#).

The following features are not supported in Windows Installer 2.0 and earlier versions.

## Installer Functions

- [MsiRemovePatches](#)
- [MsiDeterminePatchSequence](#)
- [MsiApplyMultiplePatches](#)
- [MsiEnumPatchesEx](#)
- [MsiGetPatchInfoEx](#)
- [MsiEnumProductsEx](#)
- [MsiGetProductInfoEx](#)
- [MsiQueryFeatureStateEx](#)
- [MsiQueryComponentState](#)
- [MsiExtractPatchXMLData](#)
- [MsiDetermineApplicablePatches](#)
- [MsiSourceListEnumSources](#)
- [MsiSourceListAddSourceEx](#)
- [MsiSourceListClearSource](#)

- **MsiSourceListClearAllEx**
- **MsiSourceListForceResolutionEx**
- **MsiSourceListGetInfo**
- **MsiSourceListSetInfo**
- **MsiSourceListEnumMediaDisks**
- **MsiSourceListAddMediaDisk**
- **MsiSourceListClearMediaDisk**

#### Windows Installer Structures

- **MSIPATCHSEQUENCEINFO**

#### Database Tables

- MsiPatchCertificate
- MsiPatchSequence
- MsiPatchMetadata

#### Properties

- **MSIDISABLELUAPATCHING**
- **MSIENFORCEUPGRADECOMPONENTRULES**
- **MSIPATCHREMOVE**
- **MsiPatchRemovalList**
- **MsiUISourceResOnly**
- **MsiUIHideCancel**
- **MsiUIProgressOnly**

#### System Policy

- DisableLUAPatching
- DisablePatchUninstall
- DisableFlyWeightPatching
- EnforceUpgradeComponentRules

- MaxPatchCacheSize

## Error Codes

- ERROR\_PATCH\_REMOVAL\_UNSUPPORTED
- ERROR\_UNKNOWN\_PATCH
- ERROR\_PATCH\_NO\_SEQUENCE
- ERROR\_PATCH\_REMOVAL\_DISALLOWED
- ERROR\_INVALID\_PATCH\_XML
- ERROR\_PATCH\_MANAGED\_ADVERTISED\_PRODUCT

## Logging Modes

- **INSTALLLOGMODE\_LOGONLYONERROR**

## Automation Interface

- Properties of **Product Object**
  - **Product.UserSid**
  - **Product.Sources**
  - **Product.MediaDisks**
  - **Product.FeatureState**
  - **Product.Context**
  - **Product.InstallProperty**
  - **Product.ProductCode**
  - **Product.ComponentState**
  - **Product.State**
  - **Product.SourceListInfo**
- Methods of **Product Object**
  - **Product.SourceListForceResolution**
  - **Product.SourceListClearMediaDisk**
  - **Product.SourceListClearAll**

- **Product.SourceListClearSource**
- **Product.SourceListAddSource**
- **Product.SourceListAddMediaDisk**
- Properties of **Patch Object**
  - **Patch.UserSid**
  - **Patch.State**
  - **Patch.Sources**
  - **Patch.SourceListInfo**
  - **Patch.ProductCode**
  - **Patch.PatchProperty**
  - **Patch.PatchCode**
  - **Patch.MediaDisks**
  - **Patch.Context**
- Methods of the **Patch Object**
  - **Patch.SourceListForceResolution**
  - **Patch.SourceListClearAll**
  - **Patch.SourceListClearSource**
  - **Patch.SourceListClearMediaDisk**
  - **Patch.SourceListAddSource**
  - **Patch.SourceListAddMediaDisk**
- Properties of the **Installer Object**
  - **Installer.ProductsEx**
  - **Installer.ProductInfo**
  - **Installer.PatchesEx**
- Methods of the **Installer Object**
  - **Installer.ApplyMultiplePatches**
  - **Installer.ApplyPatch**

- **Installer.RemovePatches**
- **Installer.ExtractPatchXMLData**

The following features are also not supported in Windows Installer version 2.0.2600. Windows Installer version 2.0.2600 was released with Windows XP and Windows 2000 Server. The features are available beginning with Windows Installer version 2.0.3790 released with Windows Server 2003. For a list of all Windows Installer versions, see Released Versions of Windows Installer.

### Installer Functions

- **MsiAdvertiseProductEx**
  - MSIADVERTISEOPTIONS\_INSTANCE
- **MsiApplyPatch**
- **MsiGetProductInfo**

### Automation Interface

- Methods of the **Installer Object**
  - **Installer.ApplyPatch**
  - **Installer.ProductInfo**

### Properties

- **MSINEWINSTANCE**
- **MSIINSTANCEGUID**
- **MsiINTSuiteWebServer**

### Custom Action In-Script Execution Options

- **msidbCustomActionTypeTSAware**

### Error Codes

- **ERROR\_INSTALL\_REMOTE\_PROHIBITED**

## Machine Policies

- [DisableMSI](#)
- [TransformsSecure policy](#)

## Command Line Options

- [/c](#)
- [/n](#)
- [/Lx](#)

## Control Attributes

- **ImageHandle** was removed

## Notes

The [Standard Installer Command-Line Options](#) are not supported by Windows Installer 2.0. Instead use the Windows Installer Command-Line Options.

When Windows Installer 2.0 installs a patch package, it ignores information in the MsiPatchSequence or MsiPatchMetadata tables. Later versions of the Windows Installer can use the information in these tables to improve patch sequencing, removal, and optimization. For information about improved patching functionality in Windows Installer, see [Patching](#).

Windows Installer 2.0 does not support [Uninstallable Patches](#) and the only method to remove particular patches from an application is to uninstall the entire patched application and then reinstall without reapplying any patches being removed.

Windows Installer 2.0 does not support patch sequencing and installs patches in the order that they are provided to the system when [installing multiple patches](#).

Windows Installer 2.0 does not support the use of [User Account Control \(UAC\) Patching](#) to enable digitally-signed patches that can be applied by non-administrator users.

Windows Installer 2.0 does not support [Patch Optimization](#). Patching can

take significantly more time than with later Windows Installer versions that only updates files affected by the patch.

Windows Installer 2.0 does not support [Using Windows Installer to Inventory Products and Patches](#).

Windows Installer 2.0 does not support the retrieval and modification of source list information for Windows Installer applications and patches installed on the system for all users. Later versions of Windows Installer enable administrators to manage source lists and source list properties for network, URL and media sources. Later versions enable administrators to manage source lists from an external process. For more information see [Managing Installation Sources](#).

Windows Installer 2.0 does not support installing multiple instances of products or patches without a separate installation package for each instance. Later Windows Installer versions can install multiple instances of a product by using product code transforms and one .msi package or one patch. For information see [Installing Multiple Instances of Products and Patches](#).

Starting with Windows XP with Service Pack 2 (SP2), Windows Installer can use the HTTP, HTTPS, and FILE protocols. The installer does not support the FTP and GOPHER protocols.

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# About Windows Installer

To install your applications efficiently and reduce the total cost of ownership (TCO) for your customers, you can use the Windows Installer. This section covers the major functional areas of the installer:

- [Overview of Windows Installer](#)
- [Administrative Installation](#)
- [Rollback Installation](#)
- [Maintenance Installation](#)
- [Windows Installer File Extensions](#)
- [Command Line Options](#)
- [System Reboots](#)
- [System Policy](#)
- [Source Resiliency](#)
- [Using Windows Installer and Windows Resource Protection](#)
- [System Restore Points and the Windows Installer](#)
- [File Versioning Rules](#)
- [Product Codes](#)
- [Package Codes](#)
- [Merges and Transforms](#)
- [Qualified Components](#)
- [Windows Installer Logging](#)
- [Companion Files](#)
- [Isolated Components](#)
- [Installation Context](#)

## See Also

[Released Versions, Tools, and Redistributables](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Overview of Windows Installer

The Windows Installer reduces the total cost of ownership (TCO) for your customers by enabling them to efficiently install and configure your products and applications. The installer can also provide your product with new capabilities to advertise features without installing them, to install products on demand, and to add user customizations.

The following sections provide a high-level overview of the installer:

- [Installation Package](#)
- [Components and Features](#)
- [Installation Mechanism](#)
- [Component Management](#)
- [Advertisement](#)
- [Installation-On-Demand](#)
- [Resiliency](#)
- [Customization](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installation Package

An installation package contains all of the information that the Windows Installer requires to install or uninstall an application or product and to run the setup user interface. Each installation package includes an .msi file, containing an installation database, a summary information stream, and data streams for various parts of the installation. The .msi file can also contain one or more transforms, internal source files, and external source files or cabinet files required by the installation.

Application developers must author an installation to use the installer. Because the installer organizes installations around the concept of [components and features](#), and stores all information about the installation in a relational database, the process of authoring an installation package broadly entails the following steps:

- Identify the features to be presented to users.
- Organize the application into components.
- Populate the installation database with information.
- Validate the installation package.

The next section discusses installer components and features. For more information, see [Components and Features](#). The choice of features is commonly determined by the application's functionality from the user's perspective.

It is recommended that developers use a standard procedure for choosing components. For more information, see [Organizing Applications into Components](#).

Package authors can use third-party package creation tools, or a table editor and the Windows Installer SDK, to populate the installation database. All installation packages need to be validated for internal consistency. For more information, see [Package Validation](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Components and Features

The Windows Installer organizes an installation around the concepts of components and features.

A feature is a part of the application's total functionality that a user may decide to install independently. For more information, see [Windows Installer Features](#).

A component is a piece of the application or product to be installed. The installer always installs or removes a component from a user's computer as a coherent piece. Components are usually hidden from the user. When a user selects a feature for installation, the installer determines which components must be installed to provide that feature. For more information, see [Windows Installer Components](#).

Authors of installation packages need to decide how to divide their application into features and components. The selection of features is primarily determined by the application's functionality from the user's perspective. Because components commonly must be shared across applications, or even across products, it is recommended that authors follow a standard method for selecting components. For more information, see [Organizing Applications into Components](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Windows Installer Components

A component is a piece of the application or product to be installed. Examples of components include single files, a group of related files, COM objects, registration, registry keys, shortcuts, resources, libraries grouped into a directory, or shared pieces of code such as MFC or DAO.

The installer service installs or removes a component as a single coherent piece. It tracks every component by the respective component ID GUID specified in the ComponentId column of the [Component table](#).

**Note** Two components that share the same component ID are treated as multiple instances of the same component regardless of their actual content. Only a single instance of any component is installed on a user's computer. Several features or applications may therefore share some components.

Because components are commonly shared, the author of an installation package must follow strict rules when specifying the components of a feature or application. This is essential for the correct operation of the Windows Installer reference-counting mechanism. For more information, see [Organizing Applications into Components](#).

In brief, these rules are:

- Each component must be stored in a single folder.
- No file, registry entry, shortcut, or other resources should ever be shipped as a member of more than one component. This applies across products, product versions, and companies.

For more information about using components, see

- [Installing a Missing Component](#)
- [Installing Permanent Components, Files, Fonts, Registry Keys](#)
- [Using Qualified Components](#)
- [Using Transitive Components](#)
- [Working with Features and Components](#)
- [Authoring a Large Package](#)

- [Checking the Installation of Features, Components, Files](#)
- [Searching for a Broken Feature or Component](#)
- [Publishing Products, Features, and Components](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Windows Installer Features

A feature is a part of the application's total functionality that a user recognizes and may decide to install independently. For example, a feature could be a spell-checker, a thesaurus, or a set of clip art. Hierarchical relationships of parent and child features commonly exist such that if a child feature is installed, the parent feature is automatically installed as well.

See the following for information about using features:

- [Requesting a Feature](#)
- [Reinstalling a Feature or Application](#)
- [Referencing Features in Merge Modules](#)
- [Working with Features and Components](#)
- [Checking the Installation of Features, Components, Files](#)
- [Searching for a Broken Feature or Component](#)
- [Publishing Products, Features, and Components](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installation Mechanism

There are two phases to a successful installation process: acquisition and execution. If the installation is unsuccessful, a rollback phase may occur.

## Acquisition

At the beginning of the acquisition phase, an application or a user instructs the installer to install a feature or an application. The installer then progresses through the actions specified in the sequence tables of the installation database. These actions query the installation database and generate a script that gives a step-by-step procedure for performing the installation.

## Execution

During the execution phase, the installer passes the information to a process with elevated privileges and runs the script.

## Rollback

If an installation is unsuccessful, the installer restores the original state of the computer. When the installer processes the installation script it simultaneously generates a rollback script. In addition to the rollback script, the installer saves a copy of every file it deletes during the installation. These files are kept in a hidden, system directory. Once the installation is complete, the rollback script and the saved files are deleted. For more information, see [Rollback Installation](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

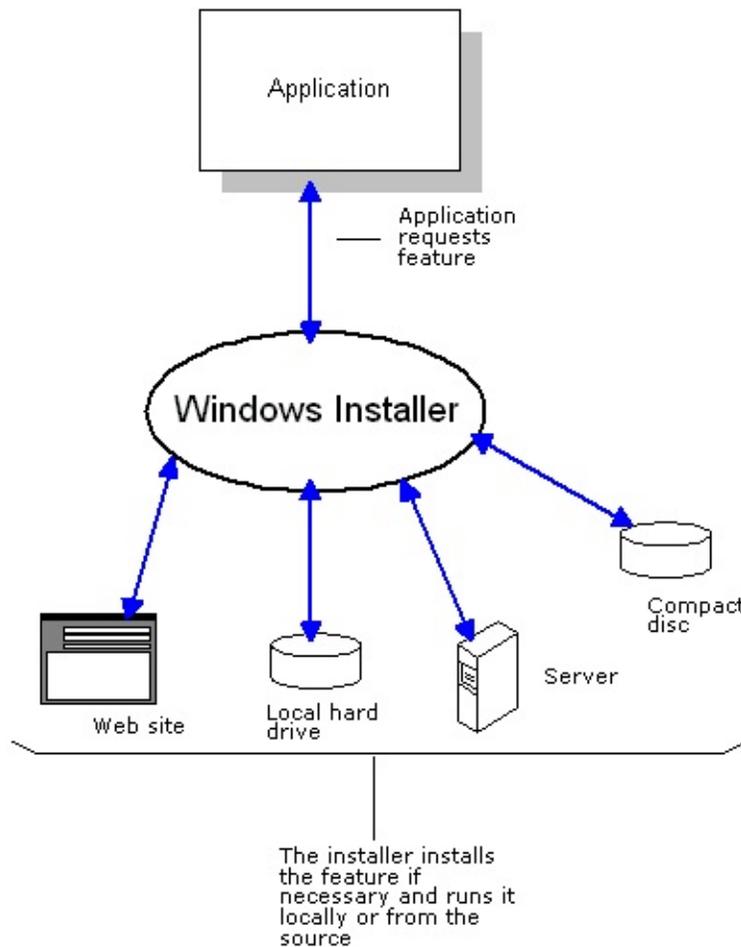
# Component Management

The Windows Installer reduces the total cost of ownership (TCO) of your applications by increasing the ability of your customers to manage and maintain application components during setup and run time. The installation database tracks which applications require a particular component, which files comprise each component, where each file is installed on the system, and where component sources are located. This allows developers to author packages that provide the following benefits:

- Increased resiliency of applications. Use the installer to detect and reinstall damaged components without having to rerun setup. The installer checks the path of a component at run time. This frees applications from dependency on static file paths which commonly differ between computers and can point to missing components. For more information, see [Resiliency](#).
- Installation-On-Demand. This feature set is not installed during setup but is specified in the database to be installed just-in-time for use if required by the application in the future. Users do not need to rerun setup. For more information, see [Installation-On-Demand](#).
- Advertisement of shortcuts to features, applications, or entire products in the user interface. Users can install these on-demand by using the shortcuts. Users can also remove features, applications, or entire products on-demand. For more information, see [Advertisement](#).
- Installation customization. An administrator can apply transforms to the database that tailor the installation for a particular group of users. For more information, see [Customization](#).
- Easier deployment of application updates. Use the installer to update your products. For more information, see [Patching and Upgrades](#).
- Feature shortcut display. The installer displays shortcuts to features that run locally with shortcuts to features that run remotely. Because

the installation database specifies the run context of each feature, visibly equivalent entry points can be presented to users as needed.

- Keep usage metrics on features. Developers can provide an installation package that keeps usage count of a feature by all client applications and removes components that are not being used.
- Incorporate installations. Developers can incorporate the component management capabilities of the installer into their applications by authoring an installation package and by using the [Installer Functions](#) in their application code. The following figure illustrates an application requesting the installation of a feature.



[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Advertisement

The Windows Installer can advertise the availability of an application to users or other applications without actually installing the application. If an application is advertised, only the interfaces required for loading and launching the application are presented to the user or other applications. If a user or application activates an advertised interface the installer then proceeds to install the necessary components as described in [Installation-On-Demand](#).

The two types of advertising are assigning and publishing. An application appears installed to a user when that application is assigned to the user. The **Start** menu contains the appropriate shortcuts, icons are displayed, files are associated with the application, and registry entries reflect the application's installation. When the user tries to open an assigned application it is installed upon demand.

The installer supports the advertisement of applications and features according to the operating system. The installer registers COM class information for assigned applications beginning with the Windows 2000 and Windows XP systems. This enables the installer to install the application upon the creation of an instance of an advertised class. For more information, see [Platform Support of Advertisement](#).

You can publish an application from the server beginning with the Windows 2000 Server. The published application is then installed through its file association or Multipurpose Internet Mail Extension (MIME) type. Publishing does not populate the user interface with any of the application's icons. The client operating system can install a published application beginning with the Windows 2000 and Windows XP systems.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Platform Support of Advertisement

The Windows Installer supports advertisement of applications and features.

The following advertisement capabilities are available on Windows Server 2008 R2, Windows 7, Windows Server 2008, Windows Vista, Windows Server 2003, Windows XP, and Windows 2000.

- Shortcuts and their icons.
- Extensions and their icons specified in the [ProgId Table](#).
- Shell and command Verbs registered underneath the ProgId key.
- CLSID contexts and InProcHandler.
- Install-On-Demand through OLE is only available programmatically through **CoCreateInstance** (C/C++), and **CreateObject Function (Visual Basic)** or **GetObject Function (Visual Basic)**.

**Note** ApplId and Typelib information is only written when an advertised component is installed.

To support file name extensions, the application must be published to Active Directory by an administrator using Group Policy.

## Remarks

The installation of the product may not require a restart, but any advertised shortcuts do not work until the machine is restarted. The advertised shortcuts of subsequent installations work without a restart.

To ensure that advertised shortcuts work correctly, the package author should schedule a forced restart at the end of the installation.

To avoid unnecessary restarts of the system, schedule a restart to run only if no version of the Windows Installer is installed.

Conditional statements can check the **ShellAdvtSupport** property and **Version9X** property. For more information about scheduling a conditional forced restarts, see [System Reboots](#) and [Using Properties in Conditional](#)

[Statements.](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installation-On-Demand

With traditional installation technology, it is necessary to exit an application and rerun setup to perform an installation task. This commonly occurred when a user wanted a feature or product not chosen during the first run of setup. This often made the process of product configuration inefficient because it required the user to anticipate the functionality required before they ever used the product.

Installation-on-demand makes it possible to offer functionality to users and applications in the absence of the files themselves. This notion is known as advertisement. The Windows Installer has the capability of advertising functionality and to make installation-on-demand of application features or entire products possible. When a user or application activates an advertised feature or product, the installer proceeds with installation of the needed components. This shortens the configuration process because additional functionality can be accessed without having to exit and rerun another setup procedure.

When a product uses the installer, a user can choose at setup time which features or applications to install and which to advertise. Then if while running an application the user requests an advertised feature that has not yet been installed, the application calls the installer to enact a just-in-time feature level installation of the necessary files. If the user activates an advertised product that has not yet been installed, the operating system calls the installer to enact a just-in-time product level installation.

[Advertisement](#) and installation-on-demand can also facilitate system management by enabling administrators to designate applications as required or optional for different groups of users. There are two types of advertising known as "assigning" and "publishing." If an administrator assigns an application to a group, then these users can install the application on-demand. If, however, the administrator publishes the application to the group, no entry points appear to these users and installation-on-demand is only activated if another application activates the published application.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Resiliency

Resiliency is the ability of an application to recover gracefully from situations in which a vital component is missing, or has been replaced by an incompatible version. By authoring an installation package and using the [Installer Functions](#), developers can use the Windows Installer to produce resilient applications capable of recovering from such situations.

- Use the installer's source list to increase the resiliency of applications that rely on network resources. For more information, see [Source Resiliency](#).
- Use the installer's API to check whether a vital feature, component, file, or file version is installed.
- Use the installer's API to check the path to a component at run time. This reduces your application's dependency on static file paths, which commonly differ between computers.
- Use the installer to reinstall damaged shortcuts, registry entries, and other components without having to rerun setup.
- Increase the resiliency of your product's installation by leaving the rollback capability of the installer enabled. For more information, see [Rollback Installation](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Customization

Because the Windows Installer keeps all information about the installation in a relational database, the installation of an application or product can be customized for particular user groups by applying transform operations to the package. Transforms can be used to encapsulate the various customizations of a base package required by different workgroups. For more information, see [Merges and Transforms](#).

Multiple transforms of a base package can be applied on-the-fly during installation. This provides a mechanism for efficiently assigning customized installations to different groups of users. For example, this would be useful in organizations where the finance and staff support departments require different installations of a particular product. The product can be made available to everyone in the organization by an [administrative installation](#) of the base package to a single administrative installation point. Each work group can then automatically receive their appropriate customization by means of an on-the-fly transform when they install the product.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Administrative Installation

The Windows Installer can perform an administrative installation of an application or product to a network for use by a workgroup. An administrative installation installs a source image of the application onto the network that is similar to a source image on a CD-ROM. Users in a workgroup who have access to this administrative image can then install the product from this source. A user must first install the product from the network to run the application. The user can choose to run-from-source when he installs and the installer uses most of the product's file directly from the network.

Administrators can run an administrative installation from the command line by using the `/a` [command line option](#).

The [ADMIN action](#) is the top-level action used to initiate an administrative installation. When this action is executed the installer calls the actions in the [AdminExecuteSequence](#) and [AdminUISequence](#) tables to perform the administrative installation.

The [AdminProperties](#) property is a semicolon delimited list of properties that are set at the time of an administration installation. The installer uses these properties during a post administration installation of the application from the administrative image.

Users who do not have continuous access to the network may install an application from an administrative image and then at times have to rely on media, such as CD-ROM disks, as their backup source. In these cases the length of the file names in the administrative image and on the media must match. Both must use long file names or both must use short file names. For example, a CD-ROM that only supports short file names could provide both the original media for installing the administrative image and a backup source.

If the [SHORTFILENAMES](#) property is set during an administrative installation, this property may need to be set again by a user subsequently applying a patch to the administrative image. When using Windows Installer to apply the patch, the installer automatically sets the **SHORTFILENAMES** property if the administrative image uses short file names.

If an administrator uses a package having a [Word Count Summary](#) property of 2 or 3 to perform an administrative installation, users of the administrative image cannot automatically reinstall from the original media source. If the administrative image becomes unavailable, users who have installed from the administrative image are prevented from reverting to the original media.

The application of *transforms* during the creation of an administrative image has no valid effect. To make a customized version of a product available to a work group, apply the transform during the installation of the product from the administrative image.

During an administrative installation, the installer creates a source image for all features in the product except those feature with 0 in the Level column of the [Feature table](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Rollback Installation

When the Windows Installer processes the installation script for the installation of a product or application, it simultaneously generates a rollback script and saves a copy of every file deleted during the installation. These files are kept in a hidden system directory and are automatically deleted once the installation is successfully completed. If however the installation is unsuccessful, the installer automatically performs a rollback installation that returns the system to its original state.

Automatic rollback of an unsuccessful installation is the default behavior of the installer. To disable rollback during an installation use one of the following:

- [DISABLEROLLBACK Property](#)
- [PROMPTROLLBACKCOST Property](#)
- [DisableRollback Action](#)
- [DisableRollback](#)
- [EnableRollback ControlEvent](#)

Whenever rollback is disabled, the installer sets the [RollbackDisabled](#) property.

If an installation uses [custom actions](#), additional authoring of the installation package is required to use rollback functionality. For more information, see [Rollback Custom Actions](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Maintenance Installation

After an application has been successfully installed, the user may want to return to the installation to add or remove features. This is known as a maintenance installation.

[Transforms](#) that have been applied to an installation must be available during subsequent maintenance installations.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Windows Installer File Name Extensions

The following file name extensions are used with the Windows Installer.

<b>Extension</b>	<b>Description</b>
.msi	Windows Installer Database.
.msm	Windows Installer Merge Module.
.msp	Windows Installer Patch.
.mst	Windows Installer Transform.
.idt	Exported Windows Installer Database Table.
.cub	Validation module.
.pcp	Windows Installer Patch Creation File

## See Also

[Installation Package](#)

[Merge Modules](#)

[Patch Packages](#)

[Transforms](#)

[Importing and Exporting](#)

[Sample .CUB File](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Command-Line Options

The executable program that interprets packages and installs products is Msiexec.exe. Note that Msiexec also sets an error level on return that corresponds to system error codes. Command-line options are case-insensitive.

The command-line options in the following table are available with Windows Installer 3.0 and earlier versions. The [Standard Installer Command-Line Options](#) are also available beginning with Windows Installer 3.0.

<b>Option</b>	<b>Parameters</b>	<b>Meaning</b>
<b>/I</b>	<i>Package ProductCode</i>	Installs or configures a product.
<b>/f</b>	[p o e d c a u m s v] <i>Package ProductCode</i>	Repairs a product. This option ignores any property values entered on the command line. The default argument list for this option is 'omus.' This option shares the same argument list as the <b>REINSTALLMODE</b> property.  p - Reinstalls only if file is missing.  o - Reinstalls if file is missing or an older version is installed.  e - Reinstalls if file is missing or an equal or older version is installed.  d - Reinstalls if file is missing or a different version is installed.  c - Reinstalls if file is missing or the stored checksum does not match the calculated value. Only repairs files that have

		<p><b>msidbFileAttributesChecksum</b> in the Attributes column of the <a href="#">File</a> table.</p> <p>a - Forces all files to be reinstalled.</p> <p>u - Rewrites all required user-specific registry entries.</p> <p>m - Rewrites all required computer-specific registry entries.</p> <p>s - Overwrites all existing shortcuts.</p> <p>v - Runs from source and re-caches the local package. Do not use the <b>v</b> reinstall option for the first installation of an application or feature.</p>
/a	<i>Package</i>	<a href="#">Administrative installation</a> option. Installs a product on the network.
/x	<i>Package ProductCode</i>	Uninstalls a product.
/j	<p>[u m]<i>Package</i> or [u m]<i>Package/tTransform List</i> or [u m]<i>Package/gLanguageID</i></p>	<p>Advertises a product. This option ignores any property values entered on the command line.</p> <p>u - Advertises to the current user.</p> <p>m - Advertises to all users of machine.</p> <p>g - Language identifier.</p> <p>t - Applies transform to advertised package.</p>

/L

[i|w|e|a|r|u|c|m|o|p|v|x|+|!|\*] *Logfile*

Writes logging information into a logfile at the specified existing path. The path to the logfile location must already exist. The installer does not create the directory structure for the logfile. Flags indicate which information to log. If no flags are specified, the default is 'iwearmo.'

i - Status messages.

w - Nonfatal warnings.

e - All error messages.

a - Start up of actions.

r - Action-specific records.

u - User requests.

c - Initial UI parameters.

m - Out-of-memory or fatal exit information.

o - Out-of-disk-space messages.

p - Terminal properties.

v - Verbose output.

x - Extra debugging information.

### **Windows Installer**

**2.0:** Not supported. The x option is available with Windows Installer version 3.0.3790.2180 and later.

+ - Append to existing file.

! - Flush each line to the log.

		<p>"*" - Wildcard, log all information except for the v and x options. To include the v and x options, specify "/l*vx".</p> <p><b>Note</b> For more information about all the methods that are available for setting the logging mode, see <a href="#">Normal Logging</a> in the <a href="#">Windows Installer Logging</a> section</p>
/m	<p><i>filename</i></p> <p><b>Note</b> The length of <i>filename</i> must be no more than eight characters.</p>	<p>Generates an SMS status .mif file. Must be used with either the install (-i), remove (-x), administrative installation (-a), or reinstall (-f) options. The ISMIF32.DLL is installed as part of SMS and must be on the path.</p> <p>The fields of the status mif file are filled with the following information:</p> <p>Manufacturer - <b>Author</b></p> <p>Product - <b>Revision Number</b></p> <p>Version - <b>Subject</b></p> <p>Locale - <b>Template</b></p> <p>Serial Number - not set</p> <p>Installation - set by ISMIF32.DLL to "DateTime"</p> <p>InstallStatus - "Success" or "Failed"</p> <p>Description - Error messages in the following order: 1) Error messages generated by installer.</p>

		<p>2) Resource from Msi.dll if installation could not commence or user exit. 3) System error message file. 4) Formatted message: "Installer error %i", where %i is error returned from Msi.dll.</p>
/p	<i>PatchPackage[;patchPackage2...]</i>	<p>Applies a patch. To apply a patch to an installed administrative image you must combine the following options:</p> <pre>/p &lt;PatchPackage&gt; [;patchPackage2...] /a &lt;Package&gt;</pre>
/q	n b r f	<p>Sets <a href="#">user interface level</a>.</p> <p>q , qn - No UI</p> <p>qb - <a href="#">Basic UI</a>. Use qb! to hide the <b>Cancel</b> button.</p> <p>qr - <a href="#">Reduced UI</a> with no modal dialog box displayed at the end of the installation.</p> <p>qf - <a href="#">Full UI</a> and any authored <a href="#">FatalError</a>, <a href="#">UserExit</a>, or <a href="#">Exit</a> modal dialog boxes at the end.</p> <p>qn+ - No UI except for a modal dialog box displayed at the end.</p> <p>qb+ - Basic UI with a modal dialog box displayed at the end. The modal box is not displayed if the user cancels the installation. Use qb+! or qb!+ to hide the <b>Cancel</b> button.</p> <p>qb- - Basic UI with no modal</p>

		<p>dialog boxes. Please note that /qb+- is not a supported UI level. Use qb-! or qb!- to hide the <b>Cancel</b> button.</p> <p>Note that the ! option is available with Windows Installer 2.0 and works only with basic UI. It is not valid with full UI.</p>
<b>/?</b> or <b>/h</b>		Displays copyright information for Windows Installer.
<b>/y</b>	<i>module</i>	<p>Calls the system function <b>DllRegisterServer</b> to self-register modules passed in on the command line. Specify the full path to the DLL. For example, for MY_FILE.DLL in the current folder you can use:</p> <p><b>msiexec /y .\MY_FILE.DLL</b></p> <p>This option is only used for registry information that cannot be added using the registry tables of the .msi file.</p>
<b>/z</b>	<i>module</i>	<p>Calls the system function <b>DllUnRegisterServer</b> to unregister modules passed in on the command line. Specify the full path to the DLL. For example, for MY_FILE.DLL in the current folder you can use:</p> <p><b>msiexec /z .\MY_FILE.DLL</b></p> <p>This option is only used for registry information that cannot be removed using the registry</p>

		tables of the .msi file.
<b>/c</b>		Advertises a new instance of the product. Must be used in conjunction with /t. Available starting with the Windows Installer version that is shipped with Windows Server 2003 and Windows XP with Service Pack 1 (SP1).
<b>/n</b>	<i>ProductCode</i>	Specifies a particular instance of the product. Used to identify an instance installed using the multiple instance support through a product code changing transforms. Available starting with the Windows Installer version shipped with Windows Server 2003 and Windows XP with SP1.

The options /i, /x, /f[p|o|e|d|c|a|u|m|s|v], /j[u|m], /a, /p, /y and /z should not be used together. The one exception to this rule is that patching an [administrative installation](#) requires using both /p and /a. The options /t, /c and /g should only be used with /j. The options /l and /q can be used with /i, /x, /f[p|o|e|d|c|a|u|m|s|v], /j[u|m], /a, and /p. The option /n can be used with /i, /f, /x and /p.

To install a product from A:\Example.msi, install the product as follows:

**msiexec /i A:\Example.msi**

Only [public properties](#) can be modified using the command line. All property names on the command line are interpreted as uppercase but the value retains case sensitivity. If you enter **MyProperty** at a command line, the installer overrides the value of MYPROPERTY and not the value of **MyProperty** in the Property table. For more information, see [About Properties](#).

To install a product with PROPERTY set to VALUE, use the following syntax on the command line. You can put the property anywhere except between an option and its argument.

Correct syntax:

```
msiexec /i A:\Example.msi PROPERTY=VALUE
```

Incorrect syntax:

```
msiexec /i PROPERTY=VALUE A:\Example.msi
```

Property values that are literal strings must be enclosed in quotation marks. Include any white spaces in the string between the marks.

```
msiexec /i A:\Example.msi PROPERTY="Embedded White Space"
```

To clear a public property by using the command line, set its value to an empty string.

```
msiexec /i A:\Example.msi PROPERTY=""
```

For sections of text set apart by literal quotation marks, enclose the section with a second pair of quotation marks.

```
msiexec /i A:\Example.msi PROPERTY="Embedded ""Quotes""  
White Space"
```

The following example shows a complicated command line.

```
msiexec /i testdb.msi INSTALLLEVEL=3 /* msi.log  
COMPANYNAME="Acme ""Widgets"" and ""Gizmos.""
```

The following example shows advertisement options. Note that switches are not case-sensitive.

```
msiexec /JM msisample.msi /T transform.mst /LIME logfile.txt
```

The following example shows you how to install a new instance of a product to be advertised. This product is authored to support multiple instance transforms.

```
msiexec /JM msisample.msi /T :instance1.mst;customization.mst /c  
/LIME logfile.txt
```

The following example shows how to patch an instance of a product that is installed using multiple instance transforms.

```
msiexec /p msipatch.msp;msipatch2.msp /n {00000001-0002-0000-
```

**0000-624474736554} /qb**

When you apply patches to a specific product, the /i and /p options cannot be specified together in a command line. In this case, you can apply patches to a product as follows.

**msiexec /i A:\Example.msi PATCH=msipatch.msp;msipatch2.msp /qb**

The **PATCH** property cannot be set in a command line, when /p option is used. If the **PATCH** property is set when the /p option is used, the value of **PATCH** property is ignored and overwritten.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Standard Installer Command-Line Options

The executable program that interprets packages and installs products is Msiexec.exe.

**Note** Msiexec also sets an error level on return that corresponds to System Error Codes.

The following table identifies the standard command-line options for this program. Command-line options are case insensitive.

**Windows Installer 2.0:** The command-line options that are identified in this topic are available beginning with Windows Installer 3.0. The Windows Installer [Command-Line Options](#) are available with Windows Installer 3.0 and earlier versions.

Option	Parameters	Meaning
<b>/help</b>		Help and quick reference option. Displays the correct usage of the command including a list of all switches and behavior. The description of usage can be displayed in the user interface. Incorrect use of any command-line option invokes this help option.  Example: <b>msiexec /help</b>  <b>Note</b> The equivalent Windows Installer Command-Line Option is <b>/?</b>
<b>/quiet</b>		Quiet display option. The installation runs without displaying a user interface. No prompts, messages, or dialog boxes are displayed to the user. The user cannot cancel the installation. Use the <b>/norestart</b> or <b>/forcerestart</b> start command-line options to control the restart behavior.

		<p>reboots. If no reboot options are specified, the installer restarts the computer whenever necessary without displaying any prompt or warning to the user.</p> <p>Examples:</p> <p><b>msiexec /package Application /quiet</b></p> <p><b>Msiexec /uninstall Application /quiet</b></p> <p><b>Msiexec /update msipatch.msi /quiet</b></p> <p><b>Msiexec /uninstall msipatch.msi /package Application.msi /quiet</b></p> <p><b>Note</b> The equivalent Windows Installer Command-Line Option is <b>/qn</b>.</p>
<p><b>/passive</b></p>		<p>Passive display option. The installer displays a progress bar to the user and indicates that an installation is in progress but no prompts or error messages are displayed to the user. The user cannot cancel the installation. Use the <b>/norestart</b> or <b>/forceresult</b> standard command-line options to control reboots. If no reboot options are specified, the installer restarts the computer whenever necessary without displaying any prompt or warning to the user.</p> <p>Example: <b>msiexec /package Application.msi /passive</b></p> <p><b>Note</b> The equivalent Windows Installer Command-Line Option is <b>/passive</b>.</p>

		<p><b>/qb!</b>- with <b>REBOOTPROMPT</b> on the command line.</p>
<b>/norestart</b>		<p>Never restart option. The installer never restarts the computer after installation.</p> <p>Example: <code>msiexec /package Application.msi /norestart</code></p> <p><b>Note</b> The equivalent Windows Installer command line has <b>REBOOT=ReallySuppress</b> set on the command line.</p>
<b>/forcerestart</b>		<p>Always restart option. The installer always restarts the computer after every installation.</p> <p>Example: <code>msiexec /package Application.msi /forcerestart</code></p> <p><b>Note</b> The equivalent Windows Installer command line has <b>REBOOT=Force</b> set on the command line.</p>
<b>/promptrestart</b>		<p>Prompt before restarting option Displays a message that a restart is required to complete the installation, and asks the user whether to restart the system now. This option cannot be used together with the <b>/quiet</b> option.</p> <p><b>Note</b> The equivalent Windows Installer command line has <b>REBOOTPROMPT = ""</b> set on the command line.</p>
<b>/uninstall</b>	<i>&lt;Package.msi ProductCode&gt;</i>	<p>Uninstall product option. Uninstalls the product.</p>

		<p><b>Note</b> The equivalent Windows Installer Command-Line Option</p>
<b>/uninstall</b>	<pre>/package &lt;Package.msi   ProductCode&gt; /uninstall &lt;Update1.msp   PatchGUID1&gt; [;Update2.msp   PatchGUID2]</pre>	<p>Uninstall update option. Uninstall update patch.</p> <p><b>Note</b> The equivalent Windows Installer Command-Line Option with <b>MSIPATCHREMOVE</b>=Update1   PatchGUID1[;Update2.msp   PatchGUID2] set on the command line.</p>
<b>/log</b>	<pre>&lt;logfile&gt;</pre>	<p>Log option. Writes logging information into a log file at the specified existing path. The path log file location must already exist. The installer does not create the directory structure for the logfile.</p> <p>The following information is written into the log:</p> <ul style="list-style-type: none"> <li>• Status messages</li> <li>• Nonfatal warnings</li> <li>• All error messages</li> <li>• Start up of actions</li> <li>• Action-specific records</li> <li>• User requests</li> <li>• Initial UI parameters</li> <li>• Out-of-memory or fatal exit information</li> <li>• Out-of-disk-space messages</li> <li>• Terminal properties</li> </ul> <p><b>Note</b> The equivalent Windows</p>

		<p>Installer Command-Line Option <b>/L*</b>.</p> <p><b>Note</b> For more information about the methods that are available for setting the logging mode, see <a href="#">N Logging</a> in the <a href="#">Windows Installer Logging</a> section.</p>
<b>/package</b>	< <i>Package.msi</i>   <i>ProductCode</i> >	<p>Install product option. Installs or configures a product.</p> <p><b>Note</b> The equivalent Windows Installer Command-Line Option</p>
<b>/update</b>	< <i>Update1.msp</i> > [; <i>Update2.msp</i> ]	<p>Install patches option. Installs or multiple patches.</p> <p><b>Note</b> The equivalent Windows Installer command line has <a href="#">PA</a> [msipatch.msp]&lt;;PatchGuid2&gt; the command line.</p>

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# System Reboots

The Windows Installer can determine when a reboot of the system is necessary and automatically prompt the user to reboot at the end of the installation. For example, the installer automatically prompts for a reboot if it needs to replace any files that are in use during the installation.

Applications that use [Windows Installer](#) version 4.0 or later for installation and servicing automatically use the Restart Manager to reduce system restarts. Windows Installer version 4.0 or later has properties and policies that enable the package author and administrators to control the interaction of Windows Installer with the Restart Manager. For more information, see [Using Windows Installer with Restart Manager](#).

Installation package authors can schedule and suppress reboots by using standard actions in the sequence tables and by setting properties. The following actions and properties are used to handle system reboots.

Action, dialog box, or property	Brief description
<a href="#">ForceReboot Action</a>	Prompts the user for a reboot during the installation.
<a href="#">ScheduleReboot Action</a>	Prompts the user for a reboot at the end of the installation.
<b>REBOOT Property</b>	Forces or suppresses certain automatic prompts for a system reboot.
<b>REBOOTPROMPT Property</b>	Suppresses the display of prompts for reboots to the user. Any needed reboots happen automatically.
<b>AFTERREBOOT Property</b>	Commonly used in a condition imposed on the ForceReboot Action.
<a href="#">InstallValidate Action</a>	Displays the FilesInUse Dialog, if necessary, giving users the opportunity to shut down

	processes and avoid some system reboots.
<a href="#">FilesInUse Dialog</a>	Gives users the opportunity to shut down processes to avoid some system reboots.
<a href="#">MsiRMFilesInUse Dialog</a>	Gives users the option to use the Restart Manager to close and restart applications. Available beginning with Windows Installer version 4.0.
<a href="#">ReplacedInUseFiles Property</a>	Set if the installer installs over a file in use. This property is used by custom actions to detect that a reboot is required.
<a href="#">MSIRESTARTMANAGERCONTROL</a>	Property to disable Windows Installer interaction with the Restart Manager. Available beginning with Windows Installer version 4.0.
<a href="#">MSIDISABLELRMRESTART</a>	Specifies how the Restart Manager closes and restarts applications. Available beginning with Windows Installer version 4.0.
<a href="#">MSIRMSHUTDOWN</a>	Specifies how the Restart Manager closes and restarts applications. Available beginning with Windows Installer version 4.0.
<a href="#">MsiSystemRebootPending</a>	Installer sets this property if a restart of the operating system is pending. Available beginning with Windows Installer version 4.0.
<a href="#">DisableAutomaticApplicationShutdown</a>	Policy to disable Windows

Installer interaction with Restart Manager. Available beginning with Windows Installer version 4.0.
---

ERROR\_INSTALL\_SUSPEND means the installation did not complete or rollback. The installation must resume before it is completed. The system may need to be rebooted before the installation can resume.

The Windows Installer returns the error code ERROR\_INSTALL\_SUSPEND when the [ForceReboot action](#) is run. It returns ERROR\_SUCCESS\_REBOOT\_REQUIRED if a reboot is required before running the application, and it returns ERROR\_SUCCESS\_REBOOT\_INITIATED if the installer has actually started a reboot. Note that because reboots are asynchronous, the reboot may actually occur before the error code is returned. For more information, see [Error Codes](#).

Custom actions can force a prompt for reboot at the end of an installation by calling [MsiSetMode](#). Custom actions can also check for a pending reboot prompt by calling [MsiGetMode](#).

## FilesInUse Dialog

The installer can determine when a reboot of the system is necessary and prompt the user with a request to reboot. Commonly, a system reboot is required because the installer is attempting to install a file that is currently being used. If the [InstallValidate action](#) detects the installation of a file in use it displays the [FilesInUse Dialog](#).

If you expect the installer to display a FilesInUseDialog, but it does not, this may be due to one of the following reasons:

- The files in use are not executables.
- The installer is not actually trying to install those files.
- The process holding those files is the process invoking the installation.
- The process holding those files is one that does not have a window

with a title associated with it.

For more information, see [Logging of Reboot Requests](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Logging of Reboot Requests

If the [Install/Validate action](#) detects the installation of a file in use it displays the [FilesInUse Dialog](#) and logs the following information.

```
Info 1603. The file E:\testdb\Test\CustAct1.dll is being held  
by the following process: Name: test, Id: 137, Window Title:
```

If the installer detects that it is about to overwrite a file that is in use, it logs the following information.

```
Info 1603. The file E:\testdb\Test\CustAct2.dll is being held
```

```
Info 1903.Scheduling reboot operation: Deleting file [filename]  
reboot to complete operation.
```

The [filename] token may actually contain a path to a file with an .rbf extension. In this case the .rbf file is actually the original file logged by the 1603 message which has been renamed to the .rbf file. The file that's in use is first renamed with an .rbf extension and then deleted.

To obtain more information about why the installer is attempting to overwrite this particular file, you can use the verbose logging option. Use the `INSTALLLOGMODE_VERBOSE` value in a call to [MsiEnableLog](#) or use the verbose output option of the [Command Line Options](#). This logs the following information.

```
MSI (s) (D0:F0): File: E:\testdb\Test\CustAct2.dll; Overwrite  
REINSTALLMODE specifies all files to be overwritten
```

The log will include a message such as "Existing file is a lower version" or "Existing file is corrupt (invalid checksum)"

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# System Policy

The installation behavior of the Windows Installer can be configured by an administrator by using the Group Policy Editor (GPE) on Windows 2000.

- [User Policies](#)
- [Machine Policies](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# User Policies

The following user policies can be configured under

**HKEY\_CURRENT\_USER\Software\Policies\Microsoft\Windows\Instal**

<b>Value name</b>	<b>Value data types</b>	<b>Description</b>
<a href="#">AlwaysInstallElevated</a>	REG_DWORD	<p>If this value is set to "1" and the corresponding computer value is also set, the installer always installs with elevated privileges.</p> <p>Otherwise, the installer uses elevated privileges to install managed applications and uses the current user's privilege level for nonmanaged applications.</p>
<a href="#">DisableMedia</a>	REG_DWORD	<p>If the DisableMedia policy is set to "1", users and administrators running a maintenance installation of one product are prevented from using the Browse Dialog to browse media sources, such as CD-ROM, for the sources of other installable products. Browsing for other products is prevented regardless of whether the installation is with elevated privileges. It is still possible for the user to reinstall the product from media if the user has a correctly labeled media source.</p>
<a href="#">DisableRollback</a>	REG_DWORD	<p>If this value is set to "1", the installer will not store rollback files during installation, disabling installation rollback. By default, rollback is enabled. Administrators</p>

		are advised to not use this policy unless it is absolutely essential.
<a href="#">SearchOrder</a>	REG_SZ	<p>Order in which the installer searches the three different types of sources:</p> <p>"n" – network</p> <p>"m" – media (CD-ROM or DVD)</p> <p>"u" – URL (Uniform Resource Locator)</p> <p>For example, a value of "nmu" instructs the installer to search network sources first, media sources second, and URL sources last. Leaving out a letter removes the corresponding volume type from the search. Default order in absence of this value is network first, then media followed by URL.</p>
<a href="#">TransformsAtSource policy</a>	REG_DWORD	<p>If this value exists and is set to "1"; the installer searches for transform files in the root of any network sources in the <a href="#">sourcelist</a> for the product. By default, transforms are stored in the Application Data folder of a user's profile.</p>

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# AlwaysInstallElevated

You can use the AlwaysInstallElevated policy to install a Windows Installer package with elevated (system) privileges.

To install a package with elevated (system) privileges, set the AlwaysInstallElevated value to "1" under both of the following registry keys:

**HKEY\_CURRENT\_USER\Software\Policies\Microsoft\Windows\Instal**

**HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta**

If the AlwaysInstallElevated value is not set to "1" under both of the preceding registry keys, the installer uses elevated privileges to install managed applications and uses the current user's privilege level for unmanaged applications.

Because this policy permits users to install applications that require access to directories and registry keys for which the user may not have permission to view or change, you should consider whether it provides your users with an appropriate level of security. Setting this policy directs Windows Installer to use system permissions when it installs the application on the system. If this policy is not set, applications not distributed by the administrator are installed using the user's privileges and only managed applications get elevated privileges.

Note that once the per-machine policy for AlwaysInstallElevated is enabled, any user can set their per-user setting.

## Remarks

For information about the interaction of this policy with installation sources, see [Managing Installation Sources](#).

## Data Type

**REG\_DWORD**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DisableMedia

If this per-user [system policy](#) is set to "1", users and administrators running a maintenance installation of one product are prevented from using the Browse Dialog to browse media sources, such as CD-ROM, for the sources of other installable products. Browsing for other products is prevented regardless of whether the installation is done with elevated privileges. It is still possible for the user to reinstall the product from media if the user has a correctly labeled media source.

## Registry Key

HKEY\_CURRENT\_USER\Software\Policies\Microsoft\Windows\Instal

## Data Type

REG\_DWORD

## Remarks

Note that the **DISABLEMEDIA** property has a different effect than the DisableMedia policy. Setting the DisableMedia system policy, only disables browsing to media sources. Setting the **DISABLEMEDIA** property prevents the installer from registering any media source, such as a CD-ROM, as a valid source for the product. If browsing is enabled however, a user may still browse to a media source.

## See Also

[Source Resiliency](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DisableRollback

If this [system policy](#) is set to "1", the installer does not store rollback files during installation and disables installation rollback. By default, rollback is enabled. Administrators are advised to not use this policy unless it is absolutely essential. For more information, see [Rollback Installation](#).

## Registry Key

To disable rollback for per-user installations, set DisableRollback to "1" under the following registry key:

**HKEY\_CURRENT\_USER\Software\Policies\Microsoft\Windows\Instal**

To disable rollback for per-computer installations, set DisableRollback to "1" under the following registry key.

**HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta**

## Data Type

**REG\_DWORD**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SearchOrder

Setting this per-user [system policy](#) specifies the order in which the installer searches three types of sources. The types of sources are:

"n" – network

"m" – media (CD-ROM or DVD)

"u" – Uniform Resource Locator (URL)

For example, to search network sources first, media sources second, and URL sources last set this policy to a value of "nmu". To omit searching for a particular source type, leave out the corresponding letter from the value.

If SearchOrder is not set, the default search order is network, media, and then URL.

## Registry Key

HKEY\_CURRENT\_USER\Software\Policies\Microsoft\Windows\Instal

## Data Type

REG\_SZ

## See Also

[Source Resiliency](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# TransformsAtSource Policy

If this per-user [system policy](#) is set to "1"; the installer searches for transform files in the root of any network sources in the source list for the product. By default, transforms are stored in the Application Data folder of a user's profile.

Windows Installer interprets the TransformsAtSource policy to be the same as the [TransformsSecure policy](#).

## Registry Key

HKEY\_CURRENT\_USER\Software\Policies\Microsoft\Windows\Instal

## Data Type

REG\_SZ

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Machine Policies

The following machine policies can be configured under:

**HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta**

Policy	Value data type	Description
<a href="#">AlwaysInstallElevated</a>	<b>REG_DWORD</b>	If this policy value is set to 1 and the corresponding <a href="#">AlwaysInstallWithElevatedPrivileges</a> policy is also set, the installation will install with elevated privileges. Otherwise, the installation will use the current user's privileges to install managed applications and unmanaged applications.
<a href="#">AllowLockdownBrowse</a>	<b>REG_DWORD</b>	If this policy value is set to 1, non-administrative users can browse for new sources while running an installation. The default is that only administrators can browse for new sources during an elevated installation. Setting this policy to 1 also enables non-administrative users to run programs with LocalSystem privileges during an elevated installation.
<a href="#">AllowLockdownMedia</a>	<b>REG_DWORD</b>	If this policy value is set to 1, non-administrative users can access media sources, such as CD-ROM, while running an installation at elevated privileges. The default is that only administrators can access media sources during an elevated installation. Setting this policy to 1

		also enables non-administrative users to run programs at LocalSystem privileges during an elevated installation.
<a href="#">AllowLockdownPatch</a>	<b>REG_DWORD</b>	If this per-machine registry value is not set, only administrators can patch products that were installed with elevated privileges. If the value is set to "1", non-administrative users can, in some cases, apply patches while running an installer with elevated privileges. If the policy is set, the patch <a href="#">minor upgrades</a> will be installed using elevated privileges; the patch <a href="#">major upgrades</a> . Setting this policy also enables non-administrative users to run programs at LocalSystem privileges during an installation.
<a href="#">Debug</a>	<b>REG_DWORD</b>	If this policy value is set to "1", the installer will support common debugging features. If the debugger using the <b>OutputDebugString</b> API, this value exists and the installer writes a log of debugging messages to the debugger using the <b>OutputDebugString</b> API. This policy is for debugging purposes only and may not be supported in future versions of Windows Installer.

<a href="#">DisableAutomaticApplicationShutdown</a>	<b>REG_DWORD</b>	<p>If this policy value is set to "1", Windows Explorer will not interact with Resilient File System (ReFS) but will use the <a href="#">Files and Storage</a> functionality.</p> <p><b>Windows Installer:</b> Not supported.</p>
<a href="#">DisableBrowse</a>	<b>REG_DWORD</b>	<p>If this policy value is set to "1", users are prevented from browsing to local drives and network sources. The <b>Use feature packs</b> combo box for direct access is locked and the <b>Browse</b> button is disabled. For more information, see <a href="#">Source browsing Resiliency</a>.</p>
<a href="#">DisableFlyWeightPatching</a>	<b>REG_DWORD</b>	<p>If this per-machine system registry value is set to "1", all Windows Installer <a href="#">Optimization</a> options are turned off during the installation.</p> <p><b>Windows Installer:</b> Not supported.</p>
<a href="#">DisableLUAPatching</a>	<b>REG_DWORD</b>	<p>If this per-machine system registry value is set to "1", this prevents non-administrators from using least-privilege (LUA) patching to a Windows Installer application installed on the computer. If this value is not set or is set to 0, administrators can apply patches to LUA-enabled applications.</p>
<a href="#">DisableMSI</a>	<b>REG_DWORD</b>	<p>If this policy value is set to "1", Windows Installer will not be used to install applications.</p>

		<p>"Null", "absent", or any value other than "1" or "2" disables the Windows Installer on the operating system. On Windows Server 2003 and Windows Server 2008 using Windows Installer on Windows 2000 Server, the Windows Installer is enabled for all applications and disallows unmanaged applications. On Windows XP the Windows Installer is enabled for all applications.</p> <p>If this policy value is "1", Windows Installer is enabled for all applications. All operations are allowed.</p> <p>If this policy value is "2", Windows Installer is disabled for unmanaged applications. Windows Installer is enabled for managed applications. Non-elevated per-user installations are blocked. Elevated and per-machine installations are allowed.</p> <p>If this policy value is "3", Windows Installer is disabled for all applications. All installs are allowed including repairs, reinstalls, or re-installs.</p>
<p><a href="#">DisablePatch</a></p>	<p><b>REG_DWORD</b></p>	<p>If this policy value is "1", the installer does not apply patches. This policy can be used to improve security in environments where patching must be rescheduled.</p>

DisablePatchUninstall

**REG\_DWORD**

If this policy value is 1, patches cannot be reinstalled on the computer by a user without administrator privileges. The Windows Installer can still remove products that are no longer available on the computer.

**Windows Installer**  
supported.

DisableRollback

**REG\_DWORD**

If this policy value is 1, the installer does not delete rollback files during uninstallation. By default, rollback files are deleted. Administrators can use this policy unless absolutely essential.

DisableSharedComponent

**REG\_DWORD**

If this per-machine policy is set to 1, no package system gets the shared component functionality enabled. The `msidbComponentAttributes` attribute in the `Component`

DisableUserInstalls

**REG\_DWORD**

If this policy value is 1, the installer searches for products in the following order: products that are registered as per-user products, products that are registered as per-machine products, and finally products that are registered as per-machine products. If this policy value is 0, the installer ignores products that are registered as per-machine products and only searches for products that are registered as per-user products.

		<p>registered as per-man attempt to perform a installation causes th display an error mes stops the installation</p>
<a href="#">EnforceUpgradeComponentRules</a>	<b>REG_DWORD</b>	<p>Set this policy value apply upgrade comp during <a href="#">small updates</a> upgrades of all prod computer.</p> <p><b>Windows Install</b> supported.</p>
<a href="#">EnableAdminTSRemote</a>	<b>REG_DWORD</b>	<p>Setting this policy en administrators to per installations from a c of a server running t Server role service.</p>
<a href="#">EnableUserControl</a>	<b>REG_DWORD</b>	<p>If this policy value is then the installer can <a href="#">public properties</a> to t during a managed in using elevated privil this policy has the sa setting the <b>EnableU</b> property.</p>
<a href="#">LimitSystemRestoreCheckpointing</a>	<b>REG_DWORD</b>	<p>This policy turns off of checkpoints by W Installer.</p> <p>If the policy value is "absent", Windows I normal checkpointin or uninstall.</p> <p>If the policy value is Windows Installer ci</p>

		checkpoints.
Logging	<b>REG_SZ</b>	This policy value is used to control whether logging has not been enabled. If the value is set to the "/L" command-line option, the <b>MsiEnableLog</b> policy value is used. If a value is set in this case, a log file is created in the temp directory with a random name: MSI*. Specify the logging mode by setting the policy value to a string of characters. Use the following characters to specify logging mode policy as used in the command-line option: <b>C</b> for console information, see <b>Console Options</b> . Note that you can use "+" and "*" for the p
MaxPatchCacheSize	<b>REG_DWORD</b>	If this policy value is set to a value greater than "0", Windows Installer saves old version files in a cache. The value specifies the maximum amount of disk space that can be used for the file cache. For example, a value of "15" and setting the policy value to maximum to 15%. Setting the value to 0 saves no files. When the value is not set, the default is 0.
MsiDisableEmbeddedUI	<b>REG_DWORD</b>	To disable embedded UI on the computer, set the policy value to 1.  <b>Windows Installer earlier:</b> Not supported.
SafeForScripting	<b>REG_DWORD</b>	If this policy value is set to 1, users are not prompted for script execution.

		scripts use installer a within a Web page. 5 useful for Web-based allow silent installati applications without knowledge or conse
<a href="#">TransformsSecure policy</a>	<b>REG_DWORD</b>	Setting the Transform policy value to "1" in installer that transform cached locally on the computer in a location user does not have w
<a href="#">DisableLoggingFromPackage</a>	<b>REG_DWORD</b>	Set this policy value disable the logging s the package by the <b>N</b> property for all users computer.  <b>Windows Install earlier:</b> Not supp

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# AllowLockdownBrowse

Setting the value of this per-machine [system policy](#) to "1" enables nonadministrative users to use a [Browse Dialog](#) to locate sources of *managed applications*. Sources may include media, such as CD-ROM, URLs, and network locations. For more information, see [Source Resiliency](#). The default on Windows Installer is that nonadministrative users cannot browse for new sources of managed applications. The only sources available are those that are already registered in the source list of the product. If this policy is set, a nonadministrative user may browse for new sources of assigned or published applications or applications being installed for all users. Setting AllowLockdownBrowse also enables nonadministrative users to run programs at LocalSystem privileges during an elevated installation.

The default setting is recommended to ensure a secure environment.

## Registry Key

HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta

## Data Type

REG\_DWORD

## Remarks

Setting this policy also enables nonadministrative users to run arbitrary programs at LocalSystem privileges if they have a Windows Installer package that installs or launches those programs.

[DisableBrowse](#) overrides AllowLockdownBrowse and prevents browsing even if AllowLockdownBrowse is set.

For information about the interaction of this policy with installation sources, see [Managing Installation Sources](#).

## See Also

[Source Resiliency](#)  
[AllowLockdownMedia](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# AllowLockdownMedia

Setting the value of this per-machine [system policy](#) to "1", enables nonadministrative users to install *managed applications* from sources stored on media, such as a CD-ROM. See [Source Resiliency](#). For example, if this policy is set, a nonadministrative user may use a media source to install assigned or published applications or applications being installed for all users. Setting this policy also enables nonadministrative users to run programs at LocalSystem privileges during an elevated installation.

The default value of this policy is 1 only on computers running Windows Vista and that are not joined to a domain. The default on other computers is that nonadministrative users cannot install managed applications from a source located on media.

Because this policy enables users that are not an administrator to install with privileges they do not have by default, before setting this policy you should consider whether it provides an appropriate level of security for your user. The default setting is recommended to ensure a secure environment.

For more information about securing installations and using digital certificates see [Guidelines for Authoring Secure Installations](#) and [Digital Signatures and Windows Installer](#) and [Downloading an Installation from the Internet](#).

## Registry Key

HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta

## Data Type

REG\_DWORD

## Remarks

Setting this policy also enables nonadministrative users to run arbitrary programs at LocalSystem privileges if they have a Windows Installer

package that installs or launches those programs.

## **See Also**

[Source Resiliency](#)  
[AllowLockdownBrowse](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# AllowLockdownPatch

If this per-machine [system policy](#) is not set, only administrators can patch existing products that were installed using elevated privileges. If AllowLockdownPatch is set to "1", nonadministrative users can, in some cases, apply patches to products while running an installation using elevated privileges. With the policy set, the patch can install [minor upgrades](#) while running an installation using elevated privileges, the patch cannot install [major upgrades](#). Setting this policy also enables nonadministrative users to run programs at LocalSystem privileges during an elevated installation.

The default setting is recommended to ensure a secure environment.

## Registry Key

HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta

## Data Type

REG\_DWORD

## Remarks

Any user can apply a patch during a nonelevated installation. Setting this per-machine [system policy](#) to "1" gives nonadministrative users the additional flexibility of applying patches to any product during an elevated installation. If this policy is not set, nonadministrative users cannot apply a patch to assigned or published applications.

Setting this policy also enables nonadministrative users to run arbitrary programs at LocalSystem privileges if they have a Windows Installer patch package that installs or launches those programs.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Debug

If this per-machine [system policy](#) is set to "1", the installer writes common debugging messages to the debugger using the **OutputDebugString** function. If this value is set to "2", the installer writes all valid debugging messages to the debugger using the **OutputDebugString** function. This policy is for debugging purposes only and may not be supported in future versions of Windows Installer.

Windows Installer only writes command lines into the log file if the third (0x04) bit is set in the value of the Debug policy. Therefore, to display command lines in the log, set the Debug policy value to 7. This does not display properties associated with an [Edit Control](#) having the [Password Control Attribute](#). This will make properties set on the command line visible in the log even if the property is included in the [MsiHiddenProperties](#) property. For more information, see [Preventing Confidential Information from Being Written into the Log File](#).

## Registry Key

HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta

## Data Type

REG\_DWORD

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DisableAutomaticApplicationShutdown

If this per-machine system policy is set to 1 (one), Windows Installer does not interact with Restart Manager, but it will use the [FilesInUse Dialog](#).

If this per-machine system policy is set to 2 (two), Windows Installer uses the FilesInUse Dialog. This setting disables attempts by the Restart Manager to mitigate restarts when installing a Windows Installer package that has not been authored to use the Restart Manager. The installer still uses the Restart Manager to detect files in use by applications.

The DisableAutomaticApplicationShutdown policy is available beginning with Windows Installer version 4.0 on Windows Vista.

## Registry Key

**HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta**

## Data Type

**REG\_DWORD**

## See Also

**[MSIRESTARTMANAGERCONTROL](#)**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DisableBrowse

Setting the value of this per-machine [system policy](#) to "1" prevents nonadministrative users from using a [Browse Dialog](#) to locate sources of [managed applications](#) stored on media, such as CD-ROM. The "Use feature from:" combo box for direct input is locked and the "Browse..." button is disabled. For more details on source browsing, see [source resiliency](#).

DisableBrowse overrides AllowLockdownBrowse and prevents browsing even if AllowLockdownBrowse is set.

## Registry Key

HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta

## Data Type

REG\_DWORD

## Remarks

In some cases with DisableBrowse set, a nonadministrative user may still be capable of installing managed applications from sources on correctly labeled media. Setting the DisableBrowse policy only disables the capability to browse to sources. It can be used to prevent a nonadministrative user from browsing to a new source during an install-on-demand installation, reinstallation, or repair. If [AllowLockdownMedia](#) is set, the nonadministrative user could still install a managed application from correctly labeled media.

DisableBrowse prevents the nonadministrative user from browsing to a new media location or any other source location. For details of how to secure media sources of managed applications see [Source Resiliency](#).

For information about the interaction of this policy with installation sources, see [Managing Installation Sources](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DisableFlyWeightPatching

If this per-machine system policy is set to 1 (one), all [Patch Optimization](#) options are turned off during the installation.

The patch optimization options and DisableFlyWeightPatching policy are available beginning with Windows Installer version 3.0.

## Registry Key

**HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta**

## Data Type

**REG\_DWORD**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DisableLoggingFromPackage

The value of this per-machine [system policy](#) is set to "1" to disable the logging that is specified for the package by the [MsiLogging](#) property for all users of the computer.

The [MsiLogging](#) property and the DisableLoggingFromPackage policy require Windows Installer 4.0.

## Registry Key

HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta

## Data Type

REG\_DWORD

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DisableLUAPatching

If this per-machine system policy is set to "1", the installer prevents non-administrators from using [User Account Control \(UAC\) Patching](#) for any application installed on the computer. When the per-machine system policy is not set or set to 0, non-administrators can apply least-privilege user patches to applications that are enabled for least-privilege user account patching.

Use the **MSIDISABLELUAPATCHING** property to prevent the least-privilege patching of an application.

The DisableLUAPatching policy is available beginning with Windows Installer version 3.0.

## Registry Key

**HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta**

## Data Type

**REG\_DWORD**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DisableMSI

If the value of this per-machine [system policy](#) is set to "2" the installer is always disabled for all applications. If this policy value is set to "1", the installer is disabled for unmanaged applications but is still enabled for managed applications.

The following table lists the possible configurations.

<b>DisableMSI</b>	<b>Description</b>
<i>Default</i>	<p>On Windows Server 2003, Windows Server 2003 R2, Windows Server 2008, and Windows Server 2008 R2, if the policy value is Null, absent, or any number other than 1 or 2, the Windows Installer is enabled for managed applications. Unmanaged application installs are blocked.</p> <p>On Windows XP, Windows Vista, and Windows 7, the Windows Installer is enabled for all applications. All install operations are allowed.</p>
0	Windows Installer is enabled for all applications. All install operations are allowed.
1	The Windows Installer is disabled for unmanaged applications but is still enabled for managed applications. Non-elevated per-user installations are blocked. Per-user elevated and per-machine installs are allowed.
2	Windows Installer is always disabled for all applications. No installs are allowed including repairs, reinstalls, or on-demand installations.

## Registry Key

HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta

## Data Type

## **REG\_DWORD**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DisablePatch

If this per-machine [system policy](#) is set to "1", the installer does not apply patches. This policy can be used to provide security in environments where patching must be restricted.

## Registry Key

HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta

## Data Type

REG\_DWORD

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DisablePatchUninstall

If this per-machine [system policy](#) is set to 1, patches cannot be removed from the computer by a user or an administrator. The Windows Installer can still remove a patch that is no longer applicable to a product. If this policy is not set, a user can remove a patch from the computer only if the user has been granted privileges to remove the patch. This can depend on whether the user is an administrator, whether [DisableMsi](#) and [AlwaysInstallElevated](#) policy settings are set, and whether the patch was installed in a per-user managed, per-user unmanaged, or per-machine context.

## Registry Key

HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta

## Data Type

REG\_DWORD

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DisableSharedComponent

If this per-machine [system policy](#) is set to 1, no package on the system gets the shared component functionality enabled by the `msidbComponentAttributesShared` attribute in the [Component table](#). The default value is 0, which enables the shared component functionality for components marked with `msidbComponentAttributesShared` in all packages.

**Windows Installer 4.0 and earlier:** Not supported. This function is available beginning with Windows Installer 4.5.

## Registry Key

HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta

## Data Type

**REG\_DWORD**

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DisableUserInstalls

This is a per-machine [system policy](#) that can be used when the administrator only wants per-machine applications installed.

If this policy is not set, the installer searches the registry for applications in the following order: managed applications registered as per-user, unmanaged applications registered as per-user, and finally applications registered as per-machine.

If this policy is set to 1, the installer ignores all applications registered as per-user and only searches for applications registered as per-machine. Calls to the Windows Installer application programming interface or system ignore per-user applications. An attempt to perform an installation in the per-user [installation context](#) causes the installer to display an error message and stops the installation. In this case, the Windows Installer also prevents per-user installations from a terminal server.

## Registry Key

HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta

## Data Type

REG\_DWORD

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# EnableAdminTSRemote

Beginning with Windows Server 2008 and Windows Vista, this policy no longer has any effect. An administrator can perform an installation from the console session of a terminal server. An administrator can also perform an installation from a client session of the terminal server.

For more information, see Terminal Services in the Microsoft Windows Software Development Kit (SDK).

## **Operating systems earlier than Windows Server 2008 and Windows Vista:**

Setting this per-machine [system policy](#) enables administrators to perform installations from a client session of the terminal server. If this policy is not set, administrators can only perform installations from the console session. Nonadministrative users can never perform installations from a client session. The default value for this policy allows only administrators to perform installations from the console session. Administrators can always do [Administrative installations](#) from a client session of the terminal server, or from the console session, regardless of whether this policy is set.

## **Registry Key**

HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta

## **Data Type**

REG\_DWORD

## **Remarks**

For more information see also, [Using Windows Installer with a Terminal Server](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# EnableUserControl

In the case of a managed installation, the package author may need to limit which public properties are passed to the server side and can be changed by a user that is not a system administrator. Some restrictions are commonly necessary to maintain a secure environment when the installation requires the installer to use elevated privileges.

If this per-machine [system policy](#) is set to "1", then the installer can pass all [public properties](#) to the server side during a managed installation using elevated privileges. Setting this policy has the same effect as setting the **EnableUserControl** property. Setting this policy allows all public properties to be passed to the service and to be changeable by nonadministrative users. By default, this policy is not enabled; only restricted public properties are passed to the server side and changeable by a nonadministrative user. All other public properties are ignored.

If the operating system is Windows 2000, the user is not a system administrator, and the application or product is being installed with elevated privileges, then a user that is not a system administrator can only override an approved list of restricted public properties. For more information see [Restricted Public Properties](#).

## Registry Key

**HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta**

## Data Type

**REG\_DWORD**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# EnforceUpgradeComponentRules

This is a per-machine [system policy](#) that can be used to apply upgrade component rules during [small updates](#) and [minor upgrades](#).

**Windows Installer 2.0:** This policy is not supported.

Set the EnforceUpgradeComponentRules policy to 1 to apply upgrade component rules during small updates and minor upgrades of all products on the computer. To apply the rules during small updates and minor upgrades of a particular product, set the **MSIENFORCEUPGRADECOMPONENTRULES** property to 1 on the command line or in the [Property table](#).

When the property or policy has been set to 1, small updates and minor upgrades can fail because the update attempts to do the following:

- Add a new feature to the top or middle of an existing feature tree. The new feature must be added as a new leaf feature to an existing feature tree.

In this case, the **ProductCode** of the product can be changed and the updates can be treated as a [major upgrade](#).

- Remove a component from a feature. This can also occur if you change the GUID of a component. The component identified by the original GUID appears to be removed and the component as identified by the new GUID appears as a new component.

**Windows Installer 4.5 and later:** The component can be removed correctly using Windows Installer 4.5 or later by setting the msidbComponentAttributesUninstallOnSupersedence attribute in the [Component table](#) or by setting the **MSIUNINSTALLSUPERSEDEDCOMPONENTS** property.

Alternatively, the **ProductCode** of the product can be changed and

the updates can be treated as a major upgrade.

## **Registry Key**

**HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta**

## **Data Type**

**REG\_DWORD**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# LimitSystemRestoreCheckpointing

This per-machine [system policy](#) turns off the creation of checkpoints by Windows Installer.

Set to 0 or absent, the installer does normal checkpointing for install or uninstall. Set to 1, the installer creates no checkpoints.

This policy affects only checkpoints set by Windows Installer. On Windows XP computers, administrators may decide to disable checkpointing from within Windows Installer to improve performance. **System Restore** also creates additional checkpoints. For more information, see [System Restore Points and the Windows Installer](#) and [Setting a Restore Point from a Custom Action](#).

## Registry Key

Set the value named LimitSystemRestoreCheckpointing under the following registry key.

**HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta**

## Data Type

**REG\_DWORD**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Logging

This per-machine [system policy](#) is used only if logging has not been enabled by the "/L" command line option or by [MsiEnableLog](#). If the policy is set in this case, the installer creates a log file in %temp% with the random name: MSI\*.LOG. Specify the logging mode by setting the policy value to a string of characters. Use the same characters to specify logging mode policy as used by the "/L" [command line option](#). Note that you cannot use "+" and "\*" for the policy.

The logging mode can be set using policy, a command line option, or programmatically. For more information about all the methods that are available for setting the logging mode, see [Normal Logging](#) in the [Windows Installer Logging](#) section.

You can prevent confidential information, for example passwords, from being entered into the log file and made visible. For more information, see [Preventing Confidential Information from Being Written into the Log File](#)

## Registry Key

Set the value named Logging under the following registry key.

**HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta**

## Data Type

**REG\_SZ**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MaxPatchCacheSize

If this per-machine [system policy](#) is set to a value greater than 0, Windows Installer saves older versions of files in a cache when a patch is applied to an application. Caching can increase the performance of future installations that otherwise need to obtain the old files from a original application source.

The value of the MaxPatchCacheSize policy is the maximum percentage of disk space that the installer can use for the cache of old files. For example, a value of 20 specifies no more than 20% be used. If the total size of the cache reaches the specified percentage of disk space, no additional files are saved to the cache. The policy does not affect files that have already been saved.

If the value of the MaxPatchCacheSize policy is set to 0, no additional files are saved.

If the MaxPatchCacheSize policy is not set, the default value is 10 and a maximum of 10% of the disk space can be used to save old files.

**Windows Installer 2.0:** Not supported. The MaxPatchCacheSize policy is available beginning with Windows Installer 3.0.

## Registry Key

HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta

## Data Type

REG\_DWORD

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiDisableEmbeddedUI

If this per-machine [system policy](#) is set to 1, the capability to use embedded UI is disabled on the computer. The default value is 0, which enables the embedded UI handlers capability.

To disable the embedded UI for a single package, you can use the [MSIDISABLEEEEUI](#) property.

**Windows Installer 4.0 and earlier:** Not supported. This function is available beginning with Windows Installer 4.5.

## Registry Key

**HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta**

## Data Type

**REG\_DWORD**

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SafeForScripting

If this per-machine [system policy](#) is set to "1", the installer does not prompt users when scripts within a Web page use the installer's [automation interface](#).

Before setting this policy, you should consider whether foregoing the user prompt provides an appropriate level of security for your users. If this policy is not set, when a script hosted by an Internet browser tries to install an application, the system warns the user and asks them to accept or refuse the installation. Setting SafeForScripting suppresses this warning and may allow the installation of applications on the system without the user's knowledge. This policy may be useful for an enterprise that uses web-based tools to distribute programs.

For more information, see [Guidelines for Authoring Secure Installations](#) and [Digital Signatures and Windows Installer](#) and [Downloading an Installation from the Internet](#).

## Registry Key

**HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta**

## Data Type

**REG\_DWORD**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# TransformsSecure Policy

Setting the TransformsSecure policy to 1 informs the installer that transforms are to be cached locally on the user's computer in a location where the user does not have write access. Setting this policy is the same as setting the [TRANSFORMSSECURE property](#) except the scope is different. Setting TransformsSecure policy applies to all packages installed to a given computer. Setting the TRANSFORMSSECURE property applies to the package regardless of the computer.

The purpose of this policy is to provide for secure transform storage with traveling users of Windows 2000. Beginning with Windows Server 2003, the default value of this policy is 1.

When this policy is set, a [maintenance installation](#) can only use the transform from the secured cache. In the event that the installer finds that the transform is missing on the local computer, the installer attempts to restore the transform. In order for the installer to restore the transform, the secure transform must reside at an authorized source in the installation package's source list. For more information, see [Source Resiliency](#). The maintenance installation fails if the transform is unavailable or cannot be loaded.

On Windows Server 2003, if this policy is not set, the default behavior is to store the transforms in a secure location. On all versions prior to Windows Server 2003, if the policy is not set, the default behavior is to store the transforms in the users profile.

See also [Secured Transforms](#) for more information.

Windows Installer interprets the [TransformsAtSource policy](#) to be the same as the TransformsSecure policy.

## Registry Key

**HKEY\_LOCAL\_MACHINE\Software\Policies\Microsoft\Windows\Insta**

## Data Type

**REG\_DWORD**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Source Resiliency

Applications that rely on network resources for [installation-on-demand](#) are susceptible to source failures if the source location should change for any reason or become damaged. The Windows Installer provides source resiliency for features that are installed on-demand by using a source list. The source list contains the locations searched by the installer for installation packages. The entries in this list can be network locations, Uniform Resource Locators (URLs), or compact discs. If one of these sources fails, the installer can quickly and seamlessly try the next.

The application developer does not need to incorporate any special information into the installer package to ensure source resiliency. Once the application is installed, the installer has the behavior of adding the last successfully used source as an entry in the source list. By default, this source is the location from which the installer package is initially installed, and is the same as the [SourceDir](#) property.

A system administrator can change the source list by applying a [transform](#) or by changing the [SOURCELIST](#) property from the command line or in the [Property table](#).

The installer begins searching for a source by checking the most recently used source location in the source list. If this search fails, the installer searches the list of network sources, then media sources, and finally URL sources. System administrators can change this search order using the [SearchOrder](#) system policy. If these searches fail, the installer may present a [Browse Dialog](#) so that the user can search for the source manually. The browse dialog box cannot be displayed if the user interface level is set to None. For details, see [User Interface Levels](#).

Commonly, the installer should only display a browse dialog box if the current user is an administrator or if the installation does not require elevated privileges. An administrator can control the display of the browse dialog box to users with the [DisableBrowse](#) and [AllowLockDownBrowse](#) policies. An administrator also controls whether users can install applications from sources located on media by using the [DisableMedia](#) and [AllowLockDownMedia](#) policies. The use of these policies depends on the Windows Installer version. For details, see the following:

- [Source Resiliency Policy](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Source Resiliency Policy

The installer begins searching for a source by checking the most recently used source location in the source list, then the list of network sources, then media sources, and finally URL sources. For more information, see [Source Resiliency](#). If the search fails, the installer may present a [Browse Dialog](#) enabling the user to search for the source manually. The browse dialog box cannot be displayed if the [user interface levels](#) is set to None. An administrator controls the display of the browse dialog box to users by setting system policy.

With Windows Installer, nonadministrators by default cannot use the browse dialog box to locate managed application sources on media and cannot install managed applications. An administrator enables a nonadministrator to browse or install managed applications from media by using the [AllowLockdownBrowse](#) and [AllowLockdownMedia](#) policies. An administrator disables the capability to browse or install applications from media by using the [DisAbleBrowse](#) and [DisAbleMedia](#) policies.

The following table summarizes the use of these system policies in Windows Installer.

System Policy	NonAdministrator User NonManaged Application	NonAdministrator User Managed Application	Administrator Managed Application NonManaged Application
<a href="#">AllowLockdownBrowse</a>	Users can install unmanaged applications using sources located on media. Users can browse media for the sources of unmanaged applications.	Users cannot install managed applications using sources located on media. Users can browse media for the sources of managed applications.	Administrator can install applications using sources located on media. Administrator can browse media for the sources of

			applications
<a href="#">AllowLockdownMedia</a>	<p>Users can install unmanaged applications using sources located on media.</p> <p>Users can browse media for the sources of unmanaged applications.</p>	<p>Users can install managed applications using sources located on media.</p> <p>Users cannot browse media for the sources of managed applications.</p>	<p>Administrators can install applications using sources located on media.</p> <p>Administrators can browse media for the sources of applications</p>
<i>Default</i>	<p>Users can install unmanaged applications using sources located on media.</p> <p>Users can browse media for the sources of unmanaged applications.</p>	<p>Users cannot install managed applications using sources located on media.</p> <p>Users cannot browse media for the sources of managed applications.</p>	<p>Administrators can install applications using sources located on media.</p> <p>Administrators can browse media for the sources of applications</p>
<a href="#">DisAbleBrowse</a>	<p>Users can install unmanaged applications using sources located on media.</p> <p>Users cannot browse media for the sources of unmanaged applications.</p>	<p>Users cannot install managed applications using sources located on media.</p> <p>Users cannot browse media for the sources of managed applications. .</p>	<p>Administrators can install applications using sources located on media.</p> <p>Administrators cannot browse media for the sources of applications</p>
<a href="#">DisAbleMedia</a>	Users cannot install	Users cannot install	Administrators

	<p>unmanaged applications using sources located on media. Users can browse media for the sources of unmanaged applications.</p>	<p>managed applications using sources located on media. Users cannot browse media for the sources of managed applications.</p>	<p>cannot install applications using sources located on media. Administrators can browse media for the sources of applications.</p>
--	---	--	---

## See Also

[Source Resiliency](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Windows Installer and Windows Resource Protection

Windows Installer adheres to Windows Resource Protection (WRP) when installing essential system files, folders, and registry information in Windows Server 2008 and later and Windows Vista and later.

WRP in Windows Server 2008 and Windows Vista replaces Windows File Protection (WFP) in Windows Server 2003, Windows XP, and Windows 2000. Windows Installer developers should note the following changes in how the installer handles protected resources in Windows Server 2008 and later and Windows Vista and later:

- When running on Windows Server 2008 and later or Windows Vista and later, the Windows Installer skips the installation of any file that is protected by WRP, the installer enters a warning in the log file, and continues with the remainder of the installation without an error. In Windows Server 2003, Windows XP, and Windows 2000, when the Windows Installer encountered a WFP-protected file, the installer would request that WFP install the file.
- WRP on Windows Server 2008 and later or Windows Vista and later can protect registry keys in addition to files. If the Windows Installer encounters a WRP-protected registry key, the installer skips the installation of that registry key, the installer enters a warning in the log file, and continues with the remainder of the installation without an error.
- Note that if a Windows Installer component contains a file or registry key that is protected by WRP, this resource must be used as the KeyPath for the component. In this case, Windows Installer does not install, update, or remove the component. You should not include any protected resources in an installation package. Instead, you should use the supported resource replacement mechanisms for Windows Resource Protection.

For more information about WRP, see Windows Resource Protection and information that is provided on [Microsoft Technet](#).

## **WFP for Windows Server 2003 and Windows XP/2000**

Windows Installer adheres to Windows File Protection (WFP) when installing essential system files on Windows Server 2003, Windows XP and Windows 2000. If a protected system file is modified by an unattended installation of an application, WFP restores the file to the verified file version.

Windows Installer never attempts to install or replace a protected file. When the [InstallFiles](#) action or any other action scheduled before InstallFiles attempts to install a file protected on Windows Server 2003, Windows XP or Windows 2000, the installer calls WFP with a request to install or replace the protected file. The installer requests the file installation from WFP immediately after executing the InstallFiles action. WFP installs or replaces the file on the user's system with a cached version of the protected file. Note that this does not guarantee that the version of the file installed from the cache is the version required by the application. After WFP has installed the file, the installer determines whether this version matches the version in the package. If the file version in the package is greater than the installed version, the installer informs the user that it cannot update the system and that an update of the operating system may be required for the application.

If any action sequenced after InstallFiles attempts to install or replace a protected file not already installed on the system, the installer cannot call WFP to install the file. In this case, the installer informs the user that it cannot update the system and that an update of the operating system may be required for the application.

The installer also checks with WFP when removing files and never attempts to remove protected system files.

## **Component Key Files Protected by WFP**

Note that if a Windows Installer component contains a WFP file, this file must be specified as the key path for the component.

When the installer attempts to install a component's key file on Windows

Server 2003, Windows XP or Windows 2000, it first calls WFP to determine if the key file is protected. When the key file of a component is protected by WFP, and that key file is already installed, the installer updates the component only if the version of the key file in the package is greater than the installed version. If the installation package specifies that a component be installed, and the key file of the component is not currently installed, then regardless of whether the key file is protected the installer installs the component. Once any component having a key file protected by WFP is installed, it is permanently installed, and the installer never removes or replaces the component.

## **Installation of Assemblies by WFP**

WFP for assemblies differs from WFP for system files.

WFP protects Windows Server 2003, Windows XP and Windows 2000 system files by detecting attempts to replace protected system files. This protection is triggered after WFP receives a directory change notification for a file in a protected directory. When WFP receives this notification, it determines which file has changed. If the file is protected, WFP looks up the file signature in a static catalog file to determine if the new file is the correct version. If the file version is not correct, the system replaces the file with the correct version from either the cache or distribution media.

In contrast, WFP of assemblies is dynamic. WFP is extended to files as they are added to the shared side-by-side assembly cache. If an assembly becomes corrupted, WFP will request that the installer replace the file. Windows Installer may or may not be able to replace the file depending on whether the source package is accessible. If the source package is inaccessible, WFP will put up a dialog box stating that it is unable to restore the file.

Note that unmanaged shared side-by-side assemblies, installed in %windir%\winsxs, are protected by WFP. Unmanaged private assemblies, installed in the application directory, are not protected by WFP. Managed global assemblies installed in the application directory or %windir%\assembly\gac are not protected by WFP.

## **See Also**

# Windows Resource Protection

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# System Restore Points and the Windows Installer

System Restore automatically monitors and records key system changes on a user's computer. For more information, see [System Restore](#).

System restore points are created by the system and are also created by Windows Installer when software is installed or removed.

On Windows XP, the installer may create checkpoints during the first installation of an application, and during its removal. The installer only creates checkpoints in these cases when the change is run with at least a *basic UI*. Installations having the [user interface level](#) set to None are usually initiated by the system or an application that should handle creating a checkpoint. For more information, see [System Restore](#).

In corporations with many small applications, administrators may decide to disable checkpointing from within the installer to improve performance. For more information, see [LimitSystemRestoreCheckpointing](#) or [Setting a Restore Point from a Custom Action](#).

Beginning with Windows Installer 5.0, the **MSIFASTINSTALL** property can be set to prevent an installation from generating a system restore point.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# File Versioning Rules

At the core of any installer is the actual installation of files. Determining whether to install a file is a complex process. At the highest level, this determination depends on whether the component to which a file belongs is marked for installation. Once determined that a file should be copied, the process is complicated if another file with the same name exists in the target folder. In such situations, making the determination requires a set of rules involving the following properties:

- Version
- Date
- Language

The installer only uses these rules when trying to install a file to a location that already contains a file with the same name. In this case, the Windows Installer uses the following rules, all other things being equal, to determine whether to install.

**Highest Version Wins**—All other things being equal, the file with the highest version wins, even if the file on the computer has the highest version.

**Versioned Files Win**—A versioned file gets installed over a nonversioned file.

**Favor Product Language**—If the file being installed has a different language than the file on the computer, favor the file with the language that matches the product being installed. Language-neutral files are treated as just another language so the product being installed is favored again.

**Mismatched Multiple Languages**—After factoring out any common languages between the file being installed and the file on the computer, any remaining languages are favored according to what is needed by the product being installed.

**Preserve Superset Languages**—Preserve the file that supports multiple languages regardless of whether it is already on the computer or is being installed.

Nonversioned Files are User Data—If the Modified date is later than the Create date for the file on the computer, do not install the file because user customizations would be deleted. If the Modified and Create dates are the same, install the file. If the Create date is later than the Modified date, the file is considered unmodified, install the file.

The installation of a [Companion File](#) depends not on its own file versioning information, but on the versioning of its companion parent. In the case of Companion Files, the installation is skipped only if the parent file has a higher version. Note that a file that is the key path for its component must not be a companion file because this results in the versioning logic of the key path file being determined by the companion parent file.

Nonversioned Files Using Companion Files-A nonversioned file that is associated with a versioned file using the companion mechanism abides by the rules for the versioned file. The only exception is if the versioned file on the computer and the versioned file being installed have the same version and language but the companion file is missing on the computer. In this case the companion file being installed is used even though the versioned file on the computer is used. Additionally, a nonversioned file using a companion file is installed if the **REINSTALLMODE** property includes the overwrite older versions options ("o" or "e") and the companion file's version is equal to a file already on the machine.

Rules are Global—The rules for determining when to install a file reside in one place within the installer and are global, meaning they apply to all files equally.

For examples of the format used for file versions, see the [Version](#) data type. For more specific information, see [Replacing Existing Files](#) or [Default File Versioning](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Default File Versioning

The flow diagrams in the following sections illustrate the default file versioning rules used when the key file of a component being installed has the same name as a file already installed in the target location. Default file versioning is also illustrated in [Replacing Existing Files](#).

Note that with Windows Installer, file hashing is available to optimize the copying of files. For details, see [MsiGetFileHash](#) and the [MsiFileHash table](#). The MsiFileHash table can only be used with unversioned files.

- [Both Files Have a Version](#)
- [Neither File Has a Version](#)
- [Neither File Has a Version with File Hash Check](#)
- [One File Has a Version](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

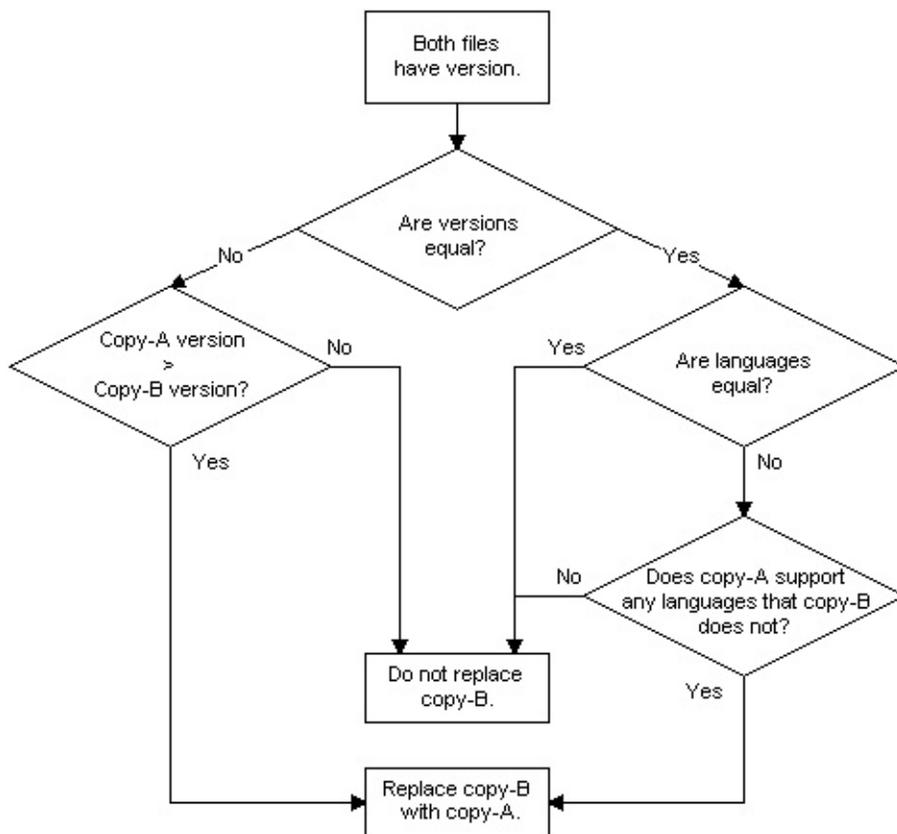
© 2009 Microsoft Corporation. All rights reserved.

# Both Files Have a Version

If the key file of a component being installed (copy-A) has the same name as a file already installed in the target location (copy-B), the installer compares the version number and language of the two files.

If both files have a version number, the installer uses the logic illustrated by the following flow diagram to determine whether to replace all of the installed files belonging to the component. Because the installer only installs entire components, if the installed key file is replaced then all of the component's files are replaced.

Note that this diagram illustrates the default [File Versioning Rules](#), which can be overridden by setting the [REINSTALLMODE](#) property. The default value of the **REINSTALLMODE** property is "omus".



The previous diagram can also be used with files with no language specified. If copy-A has a specified language and copy-B has no specified language, copy-B is replaced with copy-A. If copy-A and copy-B both have no language specified, then copy-B is not replaced.

See examples of default file versioning in [Replacing Existing Files](#).

- [Neither File Has a Version](#)
- [Neither File Has a Version with File Hash Check](#)
- [One File Has a Version](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

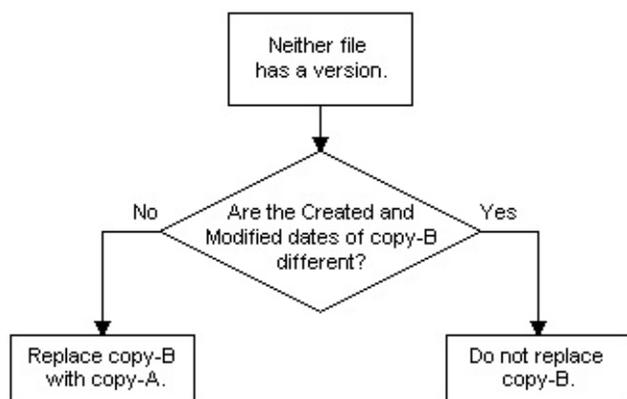
© 2009 Microsoft Corporation. All rights reserved.

# Neither File Has a Version

If the key file of a component being installed (copy-A) has the same name as a file already installed in the target location (copy-B), the installer compares the version number, date, and language of the two files.

If neither file has a version number, the installer uses the logic illustrated by the following flow diagram to determine whether to replace all of the installed files belonging to the component. Because the installer only installs entire components, if the installed key file is replaced, then all of the component's files are replaced.

Note that this diagram illustrates the default [File Versioning Rules](#), which can be overridden by setting the **REINSTALLMODE** property. The default value of the **REINSTALLMODE** property is "omus".



See the examples of default file versioning in [Replacing Existing Files](#).

- [Both Files Have a Version](#)
- [Neither File Has a Version with File Hash Check](#)
- [One File Has a Version](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

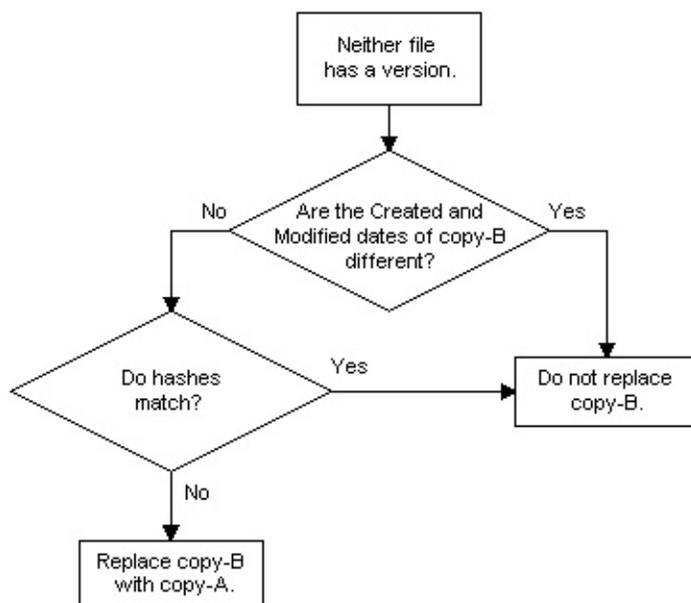
# Neither File Has a Version with File Hash Check

File hashing is available with Windows Installer. For more information, see [MsiGetFileHash](#) and the [MsiFileHash table](#). The MsiFileHash table can only be used with unversioned files.

If the key file of a component being installed (copy-A) has the same name as a file already installed in the target location (copy-B), the installer compares the version number, date, and language of the two files.

If neither file has a version number, the installer uses the logic illustrated by the following flow diagram to determine whether to replace all of the installed files belonging to the component. Because the installer only installs entire components, if the installed key file is replaced then, all of the component's files are replaced.

Note that this diagram illustrates the default [File Versioning Rules](#), which can be overridden by setting the [REINSTALLMODE](#) property. The default value of the **REINSTALLMODE** property is "omus".



See the examples of default file versioning in [Replacing Existing Files](#).

- [Both Files Have a Version](#)
- [Neither File Has a Version](#)
- [One File Has a Version](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

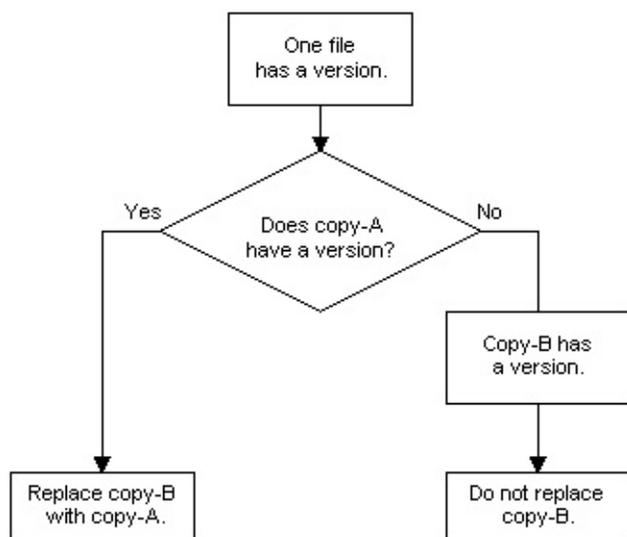
© 2009 Microsoft Corporation. All rights reserved.

# One File Has a Version

If the key file of a component being installed (copy-A) has the same name as a file already installed in the target location (copy-B), the installer compares the version number, date, and language of the two files.

If only one of the files has a version number, the installer uses the logic illustrated by the following flow diagram to determine whether to replace all of the installed files belonging to the component. Because the installer only installs entire components, if the installed key file is replaced, then all of the component's files are replaced.

Note that this diagram illustrates the default [File Versioning Rules](#), which can be overridden by setting the **REINSTALLMODE** property. The default value of the **REINSTALLMODE** property is "omus".



See the examples of default file versioning in [Replacing Existing Files](#).

- [Both Files Have a Version](#)
- [Neither File Has a Version](#)
- [Neither File Has a Version with File Hash Check](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Product Codes

The product code is a GUID that is the principal identification of an application or product. For more information, see the [ProductCode](#) property. If significant changes are made to a product then the product code should also be changed to reflect this. It is not however a requirement that the product code be changed if the changes to the product are relatively minor.

The 32-bit and 64-bit versions of an application's package must have different product codes. If any 32-bit component of an application is recompiled into a 64-bit component, a new product code must be assigned.

If a server exposed in the [PublishComponent Table](#) is recompiled from 32-bits to 64-bits, the GUID in this table may also need to be changed so that 32-bit and 64-bit clients can identify the appropriate qualified component category. In this case, the product code must also be changed.

Note that letters in product code GUIDs must be uppercase. Utilities such as GUIDGEN generate GUIDs containing lowercase letters. The lowercase letters in these GUIDs must be changed to uppercase to be used as a product code or package code. For more information, see [Changing the Product Code](#).

The package code is a GUID identifying a particular Windows Installer [package](#). The package code associates an .msi file with an application or product and can also be used for the verification of sources. The product and package codes are not interchangeable. No two nonidentical .msi files should ever have the same package code. Although it is common to ship an application that has the same package code and product code, the two values can diverge as the application is updated. For more information, see [Package Codes](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Package Codes

The package code is a GUID identifying a particular Windows Installer *package*. The package code associates an .msi file with an application or product and can also be used for the verification of sources. The product and package codes are not interchangeable. For details, see [Product Codes](#).

Nonidentical .msi files should not have the same package code. It is important to change the package code because it is the primary identifier used by the installer to search for and validate the correct package for a given installation. If a package is changed without changing the package code, the installer may not use the newer package if both are still accessible to the installer.

The package code is stored in the [Revision Number Summary](#) Property of the [Summary Information Stream](#). Note that letters in product code and package code GUIDs must be uppercase. Utilities such as GUIDGEN generate GUIDs containing lowercase letters. The lowercase letters in these GUIDs must be changed to uppercase to be used as a product code or package code.

Although it is common to ship an application that has the same package code and product code, the two values can diverge as the application is updated. For example, including a new file with the application would require updating the installation database to install the file. If the changes are minor a developer may choose not to change the product code, however, a different .msi file is needed to install the new file and so the package code must be incremented. Conversely, a single package can be used to install more than one product. For example, the installation of a package without a language transform could install the English version of the application and the installation of the same package with a language transform could install the French version. The transform is distinct from the .msi file that determines the package code. The English and French versions could have different product codes and the same package code because they are both installed with the same .msi file.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merges and Transforms

The Windows Installer keeps all information about the installation in a relational database. You can modify this database, and therefore the installation, by using transforms and merges.

## Transforms

A [database transform](#) adds or replaces elements in the original database. For example, a transform can change all of the text in an application's user interface from French to English.

Primary uses for transforms include:

- Customization of base installation packages for particular groups of users.

Transforms can be used to encapsulate the various customizations of a single base package that are required by different groups of users. For example, this is useful in organizations where the finance and staff support departments require different installations of a particular product. A product's base package can be available to everyone at one administrative installation point with appropriate customizations distributed to each group of users separately.

- Synchronization of applications across languages.

Transforms are useful for keeping packages authored at widely separated locations synchronized during authoring. For example, if an upgrade is first developed for an English version of an application that exists in English and French, a transform can be applied to the upgraded English version that converts it into an upgraded French version.

Multiple transforms can be applied to a base package and then applied on-the-fly during installation. This extends the capabilities of the installer to create custom packages and provides a mechanism

for efficiently assigning the most appropriate installations to different groups of users.

- Patching applications.

Transforms can be used to apply a minor fix to an application that does not warrant a major upgrade. For more information about patches, see [Patch Packages](#).

## Merges

A merge combines two databases into one database, and adds, rather than replaces, information. If the same information exists in both databases, a merge conflict occurs. Merges are useful to development teams because they allow a large application to be divided into parts that can be recombined later. For example, the database elements for the installation of a new component can be developed separately and later merged into the main installation database. For more information, see [Merge Modules](#).

A development team might apply a merge operation in the following way:

1. Separate into groups and work simultaneously on different components of a large application.
2. Each development group then populates a database with installation information for its own component, without being concerned with the other components of the application.
3. After the development of a component is complete, that component's database can be merged into the main installation database for the entire application.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Qualified Components

A qualified component is a method of single-level indirection, similar to a pointer. Qualified components are primarily used to group components with parallel functionality into categories. For example, if you have 30 components listed in the [Component table](#) that are the same Microsoft Word fax template localized into 30 languages, you can group these together into a category of qualified components by using the [PublishComponent table](#).

Qualified components are entered in the Component table in the same way as ordinary components. Every component must have a unique component ID GUID and component identifier specified in the Component table. In addition, qualified components are associated with a category GUID and a text-string qualifier in the PublishComponent table. Qualified components are referenced by the category GUID and the qualifier, which just points to the ordinary component in the Component table.

For example, a qualified component ID GUID can point to different language versions of a resource DLL. In this case, the group of localized resource DLLs comprises the category and the numeric locale identifiers (LCID) strings are commonly used as the qualifiers. A developer could author an installation package that uses these qualified components to do the following:

- Find the path to a particular language version of the resource DLL using [MsiProvideQualifiedComponent](#) or [MsiProvideQualifiedComponentEx](#) and install the resource.
- Determine all of the language versions of the resource DLL that are present by calling [MsiEnumComponentQualifiers](#).
- Prepare the application to support additional languages. A future language pack for the application can use the qualified component to add more language versions of the resource DLL.

For more information, see [Using Qualified Components](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Windows Installer Logging

Windows Installer records errors and events in its own error log and in the Event log. The diagnostic information that the installer writes to these logs can help users and administrators understand the cause of a failed installation.

- [Normal Logging](#)
- [Event Logging](#)
- [Preventing Confidential Information from Being Written into the Log File](#)
- [Willogutl.exe](#) assists the analysis of log files from a Windows Installer installation, and displays suggested solutions to errors that are found in a log file.

For more information about interpreting Windows Installer log files, see the white paper available on the TechNet site: [Windows Installer: Benefits and Implementation for System Administrators](#).

## See Also

[Logging of Action Return Values](#)

[Logging of Reboot Requests](#)

[Checking the Installation of Features, Components, Files](#)

[willogutl.exe](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Normal Logging

The installer records errors and events in its own error log. The type of logging that is performed by the installer is determined by the setting of the logging mode. Logging is enabled and the mode can be set by using the following methods:

- The logging mode of an installation launched from the command line can be specified by using the /L option of the [Command Line Options](#). If the logging mode is not specified by using the /L command-line option, the default logging mode will be used.
- The logging mode of an installation process can be specified programmatically by using the [MsiEnableLog](#) function or the [EnableLog](#) method. If the logging mode is not specified by using the [MsiEnableLog](#) function or the [EnableLog](#) method, the default logging mode will be used.
- The default logging mode of a particular installation package can be specified by setting the [MsiLogging](#) property in the [Property table](#) of the package. This property is available starting with Windows Installer 4.0.
- If the [MsiLogging](#) property is present in the Property table, the default logging mode of the package can be modified by changing the value by using a [database transform](#). The default logging mode cannot be changed by using [Patch Packages](#) (a .msp file.)
- If the [MsiLogging](#) property has not been set, the default logging mode for all users of the computer can be specified by using the [Logging](#) policy.
- If the [MsiLogging](#) property has been set, the default logging mode for all users of the computer can be specified by setting both the [DisableLoggingFromPackage](#) policy and [Logging](#) policy.
- If the logging mode has not been specified by the /L option, [MsiEnableLog](#), [EnableLog](#), [MsiLogging](#) property, or the [Logging](#)

policy, then the default logging mode for the package is the same mode that is obtained as setting the **MsiLogging** property to 'iwearmo'.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Event Logging

Windows Events provides a standard, centralized way for applications (and the operating system) to record important software and hardware events. The event-logging service stores events from various sources in a single collection called an *event log*. Prior to Windows Vista, you would use either Event Tracing for Windows (ETW) or Event Logging to log events. Windows Vista introduced a new eventing model that unifies both ETW and the Windows Event Log API.

The installer also writes entries into the event log. These record events such as following:

- Success or failure of the installation; removal or repair of a product.
- Errors that occur during product configuration.
- Detection of corrupted configuration data.

If a large amount of information is written, the Event Log file can become full and the installer displays the message, "The Application log file is full."

The installer may write the following entries in the event log. All event log messages have a unique event ID. All general errors authored in the [Error table](#) that are returned for an installation that fails are logged in the Application Event Log with a message ID equal to the Error + 10,000. For example, the error number in the Error table for an installation completed successfully is 1707. The successful installation is logged in the Application Event Log with a message ID of 11707 (1707 + 10,000).

For information about how to enable verbose logging on a user's computer when troubleshooting deployment, see [Windows Installer Best Practices](#).

<b>Event ID</b>	<b>Message</b>	<b>Remarks</b>
1001	Detection of product '%1', feature '%2' failed during request for component '%3'	A warning message. For details, see <a href="#">Searching For a Broken Feature or Component</a> .

1002	Unexpected or missing value (name: '%1', value: '%2') in key '%3'	Error message that there was an unexpected or missing value.
1003	Unexpected or missing subkey '%1' in key '%2'	Error message that there was an unexpected or missing subkey.
1004	<p>Detection of product '%1', feature '%2', component '%3' failed</p> <p><b>Note:</b> Beginning with Windows Installer version 2.0, this message is: Detection of product '%1', feature '%2', component '%3' failed. The resource '%4' does not exist.</p>	A warning message. See also <a href="#">Searching For a Broken Feature or Component</a> .
1005	Install operation initiated a reboot	Informational message that the installation initiated a reboot of the system.
1006	<p>Verification of the digital signature for cabinet '%1' cannot be performed. WinVerifyTrust is not available on the computer.</p>	Warning message. A cabinet was authored in the <a href="#">MsiDigitalSignature table</a> to have a <b>WinVerifyTrust</b> check performed. This action could not be performed because the computer does not have the proper cryptography DLLs installed.
1007	The installation of %1 is not permitted by software restriction policy. The Windows	An error message indicating that the administrator has configured software restriction policy to disallow this install.

	<p>Installer only allows execution of unrestricted items. The authorization level returned by software restriction policy was %2.</p>	
1008	<p>The installation of %1 is not permitted due to an error in software restriction policy processing. The object cannot be trusted.</p>	<p>An error message indicating that there were problems attempting to verify the package according to software restriction policy.</p>
1012	<p>This version of Windows does not support deploying 64-bit packages. The script '%1' is for a 64-bit package.</p>	<p>Error message indicating that scripts for 64-bit packages can only be executed on a 64-bit computer.</p>
1013	<p>{Unhandled exception report}</p>	<p>Error message for an unhandled exception, this is the report.</p>
1014	<p>Windows Installer proxy information not registered correctly</p>	<p>Error message that proxy information was not registered correctly.</p>
1015	<p>Failed to connect to server. Error: %d</p>	<p>Informational message that the installation failed to connect to server.</p>
1016	<p>Detection of product '%1', feature '%2', component '%3' failed. The resource '%4' in a run-from-source component could not be located because no valid and accessible source could be found.</p>	<p>Warning message. For more information, see <a href="#">Searching for a Broken Feature or Component</a>.</p>

1017	User SID had changed from '%1' to '%2' but the managed app and the user data keys cannot be updated. Error = '%3'.	Error message indicating that an error occurred while attempting to update the user's registration after the user's SID changed.
1018	The application '%1' cannot be installed because it is not compatible with this version of Windows.	Error message indicating that the installation is incompatible with the currently running version of Windows. Contact the manufacturer of the software being installed for an update.
1019	Product: %1 - Update '%2' was successfully removed.	Informational message that the installer has removed the update.  <b>Windows Installer 2.0:</b> Not available.
1020	Product: %1 - Update '%2' could not be removed. Error code %3. Additional information is available in the log file %4.	Error message indicating that the installer was unable to remove the update. Additional information is available in the log file.  <b>Windows Installer 2.0:</b> Not available.
1021	Product: %1 - Update '%2' could not be removed. Error code %3.	Error message indicating that the installer was unable to remove the update. For information on how to turn on logging, see <a href="#">Enable verbose logging on user's computer when troubleshooting deployment</a> .  <b>Windows Installer 2.0:</b> Not available.
1022	Product: %1 - Update '%2' installed successfully.	Informational message that the installer has installed the update successfully.  <b>Windows Installer 2.0:</b> Not available.
1023	Product: %1 - Update	Error message indicating that the installer was

	'%2' could not be installed. Error code %3. Additional information is available in the log file %4.	unable to install the update. Additional information is available in the log file.  <b>Windows Installer 2.0:</b> Not available.
1024	Product: %1 - Update '%2' could not be installed. Error code %3.	Error message indicating that the installer was unable to install the update. For information on how to turn logging on, see <a href="#">Enable verbose logging on user's computer when troubleshooting deployment</a> .  <b>Windows Installer 2.0:</b> Not available.
1025	Product: %1. The file %2 is being used by the following process: Name: %3 , Id %4.	<b>Windows Installer 2.0:</b> Not available.
1026	Windows Installer has determined that its configuration data registry key was not secured properly. The owner of the key must be either Local System or Builtin\Administrators. The existing key will be deleted and re-created with the appropriate security settings.	Warning message.  <b>Windows Installer 3.1 and earlier:</b> Not available.
1027	Windows Installer has determined that a registry sub key %1 within its configuration data was	Warning message.  <b>Windows Installer 3.1 and earlier:</b> Not available.

	<p>not secured properly. The owner of the key must be either Local System or Builtin\Administrators. The existing sub key and all of its contents will be deleted.</p>	
1028	<p>Windows Installer has determined that its configuration data cache folder was not secured properly. The owner of the key must be either Local System or Builtin\Administrators. The existing folder will be deleted and re-created with the appropriate security settings.</p>	<p>Warning message</p> <p><b>Windows Installer 3.1 and earlier:</b> Not available.</p>
1029	<p>Product: %1. Restart required.</p>	<p>Warning message indicating that a system restart is required to complete the installation and the restart has been deferred to a later time.</p> <p><b>Windows Installer 3.1 and earlier:</b> Not available.</p>
1030	<p>Product: %1. The application tried to install a more recent version of the protected Windows file %2. You may need to update your</p>	<p>Warning message indicating that the installation tried to replace a critical file that is protected by <a href="#">Windows Resource Protection</a>. An update of the operating system may be required to use this application.</p> <p><b>Windows Installer 3.1 and earlier:</b> Not available.</p>

	operating system for this application to work correctly. (Package Version: %3, Operating System Protected Version: %4).	
1031	Product: %1. The assembly '%2' for component '%3' is in use.	Warning message indicating that the installation tried to update an assembly currently in use. The system must be restarted to complete the update of this assembly.  <b>Windows Installer 3.1 and earlier:</b> Not available.
1032	An error occurred while refreshing environment variables updated during the installation of '%1'.	Warning message indicating that some users who are logged on to the computer may need to log off and back on to complete the update of environment variables.  <b>Windows Installer 3.1 and earlier:</b> Not available.
1033	Product: %1. Version: %2. Language: %3. Installation completed with status: %4. Manufacturer: %5.	Field 1 - <b>ProductName</b> Field 2 - <b>ProductVersion</b> Field 3 - <b>ProductLanguage</b>  <b>Windows Installer 3.1 and earlier:</b> Not available.  Field 5 - <b>Manufacturer</b>  <b>Windows Installer 4.5 and earlier:</b> Field 5 not available.
1034	Product: %1. Version: %2. Language: %3.	Field 1 - <b>ProductName</b> Field 2 - <b>ProductVersion</b>

	Removal completed with status: %4. Manufacturer: %5.	Field 3 - <b>ProductLanguage</b>  <b>Windows Installer 3.1 and earlier:</b> Not available.  Field 5 - <b>Manufacturer</b>  <b>Windows Installer 4.5 and earlier:</b> Field 5 not available.
1035	Product: %1. Version: %2. Language: %3. Configuration change completed with status: %4. Manufacturer: %5.	Field 1 - <b>ProductName</b> Field 2 - <b>ProductVersion</b> Field 3 - <b>ProductLanguage</b> Field 5 - <b>Manufacturer</b>  <b>Windows Installer 4.5 and earlier:</b> Field 5 not available.
1036	Product: %1. Version: %2. Language: %3. Update: %4. Update installation completed with status: %5. Manufacturer: %6.	Field 1 - <b>ProductName</b> Field 2 - <b>ProductVersion</b> Field 3 - <b>ProductLanguage</b> Field 4 - This is the user friendly name if the <a href="#">MsiPatchMetadata Table</a> is present in the patch package. Otherwise, this is the patch code GUID of the patch. Field 5 - Status of update installation.  <b>Windows Installer 3.1 and earlier:</b> Not available.  Field 6 - <b>Manufacturer</b>  <b>Windows Installer 4.5 and earlier:</b> Field 6 not available.
1037	Product: %1. Version:	Field 1 - <b>ProductName</b>

	<p>%2. Language: %3. Update: %4. Update removal completed with status: %5. Manufacturer: %6.</p>	<p>Field 2 - <b>ProductVersion</b></p> <p>Field 3 - <b>ProductLanguage</b></p> <p>Field 4 - This is the user friendly name if the <a href="#">MsiPatchMetadata Table</a> is present in the patch package. Otherwise, this is the patch code GUID of the patch.</p> <p>Field 5 - Status of update removal.</p> <p><b>Windows Installer 3.1 and earlier:</b> Not available.</p> <p>Field 6 - <b>Manufacturer</b></p> <p><b>Windows Installer 4.5 and earlier:</b> Field 6 not available.</p>
1038	<p>Product: %1. Version: %2. Language: %3. Reboot required. Reboot Type: %4. Reboot Reason: %5. Manufacturer: %6.</p>	<p>Field 1 - <b>ProductName</b></p> <p>Field 2 - <b>ProductVersion</b></p> <p>Field 3 - <b>ProductLanguage</b></p> <p>Field 4 - A constant indicating the type of restart:</p> <p>msirbRebootImmediate (1) - There was an immediate restart of the computer.</p> <p>msirbRebootDeferred (2) - A user or admin has deferred a required restart of the computer using the UI or <b>REBOOT=ReallySuppress</b>.</p> <p>Field 5 - A constant indicating the reason for the restart:</p> <p>msirbRebootUndeterminedReason (0)- Restart required for an unspecified reason.</p> <p>msirbRebootInUseFilesReason (1)- A restart was required to replace</p>

		<p>files in use.</p> <p>msirbRebootScheduleRebootReason (2)- The package contains a <a href="#">ScheduleReboot</a> action.</p> <p>msirbRebootForceRebootReason (3)- The package contains a <a href="#">ForceReboot</a> action.</p> <p>msirbRebootCustomActionReason (4)- A custom action called the <a href="#">MsiSetMode</a> function.</p> <p><b>Windows Installer 3.1 and earlier:</b> Not available.</p> <p>Field 6 - <a href="#">Manufacturer</a></p> <p><b>Windows Installer 4.5 and earlier:</b> Field 6 not available.</p>
10005	The installer has encountered an unexpected error installing this package. This may indicate a problem with this package. The error code is [1]. {{The arguments are: [2], [3], [4]}}	Error message indicating an internal error occurred. The text of this message is based upon the text authored for error 5 in the Error table.
11707	Product [2] – Installation operation completed successfully	Informational message that the installation of the product was successful.
11708	Product [2] – Installation operation failed	Error message that the installation of the product failed.
11728	Product [2] -- Configuration	Informational message that configuration of the product was successful.

completed successfully.
----------------------------

---

You can import localized errors strings for events into your database by using Msidb.exe or **MsiDatabaseImport**. The SDK includes localized resource strings for each of the languages listed in the [Localizing the Error and ActionText Tables](#) section. If the error strings corresponding to events are not populated, the installer loads localized strings for the language specified by the **ProductLanguage** property.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Logging of Action Return Values

The Installer writes the following values into the log when an action returns these error codes. For the entire list of error codes returned by the Windows Installer function calls MsiExec.exe, and InstMsi.exe, see [Error Codes](#).

<b>Error Code</b>	<b>Values returned by function calls MsiExec.exe, and InstMsi.exe</b>	<b>Values that appear in the Log.</b>	<b>Descript</b>
ERROR_FUNCTION_NOT_CALLED	1626	0	A function executed
ERROR_SUCCESS	0	1	An action successful
ERROR_INSTALL_USEREXIT	1602	2	A user canceled installation
ERROR_INSTALL_FAILURE	1603	3	A fatal error occurred
ERROR_INSTALL_SUSPEND	1604	4	The installation is suspended
ERROR_SUCCESS	0	5	The action successful
ERROR_INVALID_HANDLE_STATE	1609	6	The handle is in an invalid state
ERROR_INVALID_DATA	1626	7	The data is invalid
ERROR_INSTALL_ALREADY_RUNNING	1618	8	Another installation is already running. Action will not run. See <a href="#">InstallEx AdminE</a>

		or <a href="#">AdvtExe</a> tables.
--	--	--

## See Also

[Windows Installer Logging](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Preventing Confidential Information from Being Written into the Log File

When using the Windows Installer, you can prevent confidential information, for example passwords, from being entered into the log file and made visible.

- The Installer never writes the information in the Password column of the [ServiceInstall table](#) into the log.
- You can prevent the Installer from writing the property that is associated with an [Edit Control](#) into the log by setting the [Password Control Attribute](#). The property associated with an Edit Control that has the Password Control Attribute is hidden even if the [Debug](#) policy is set to a value of 7.
- You can prevent the Installer from writing a private property into the log by including the property in the [MsiHiddenProperties](#) property. **Note** This method can make confidential information entered on a command line visible in the log. When the Debug policy is set to a value of 7, the installer will write information entered on a command line into the log. This makes the property entered on a command line visible even if the property is included in the **MsiHiddenProperties** property.
- You can prevent the information in the Target column of the [CustomAction](#) Table from being written into the log by including the HideTarget bit flag in the Type field of the CustomAction table. The value of this flag is 8192 (0x2000). For more information, see [Custom Action Hidden Target Option](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Companion Files

The installation state of a companion file depends not on its own file versioning information, but on the versioning of its companion parent. See the [File Versioning Rules](#). To specify a companion file, the primary key of the companion parent in the [File table](#) must be authored into the Version column of the record for the companion.

In the following example, FileA is the companion parent and FileB is the companion file.

## [File Table](#) (partial)

File	Version
FileA	1.0.0.0
FileB	FileA

In this example, the installation state of FileB depends on the [File Versioning Rules](#) and the versioning information for FileA. If the installer determines that the version of FileA in the package should be installed over an older version of FileA that already exists on the user's computer, it will also install FileB from the package regardless of the version of any installed FileB.

Note that a file that is the key path for its component must not be a companion file. This would result in the versioning logic of the key path file being determined by the companion parent file.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Isolated Components

Authors of installation packages can specify that the installer copy the shared files (commonly shared DLLs) of an application into that application's folder rather than to a shared location. This private set of files (DLLs) are then used only by the application. Isolating the application together with its shared components in this manner has the following advantages:

- The application always uses the versions of the shared files with which it was deployed.
- Installing the application does not overwrite other versions of the shared files by other applications.
- Subsequent installations of other applications using different versions of the shared files cannot overwrite the files used by this application.

Because the current implementation of COM keeps a single full path in the registry for each CLSID/Context pair, it forces all applications to use the same version of a shared DLL. To enable an application to keep a private copy of a COM server, the system loader in Windows 2000 checks for the presence of a .LOCAL file in the application's folder. If the system loader detects a .LOCAL file, it alters its search logic to prefer DLLs located in the same folder as the application.

When Windows Installer runs the [IsolateComponents action](#) they copy the files of the component (commonly a shared DLL) specified in the Component\_Shared column of the [IsolatedComponent table](#) into the same folder as the component (commonly an .exe file) specified in the Component\_Application column. The installer creates a file in this directory, zero bytes in length, having the short file name of the key file for Component\_Application (typically the name is the same as the application's .exe) appended with .LOCAL. The installer uses the registration for the component in its shared location and does not write any registration information for the copy of the component in the private location.

For more information, see:

- [Installation of Isolated Components](#)
- [Reinstallation of Isolated Components](#)
- [Removal of Isolated Components](#)
- [Using Isolated Components](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installation of Isolated Components

Windows Installer performs the following actions during installation of an application when the package contains isolated components. Typically, Component\_Shared is a DLL that is shared by Component\_Application and other client executables.

## Installation

- Copy the files of Component\_Shared into the same folder as Component\_Application only if Component\_Application is also being installed.
- Create a zero-byte file with the short file name of the key file of Component\_Application. Locate this file in the same folder as Component\_Application. Append the extension .LOCAL to this file name.
- Increment the SharedDLL refcount if the msidbComponentAttributesSharedDllRefCount bit is set in the Attributes column of the [Component table](#).
- Register Component\_Application as a client of Component\_Shared and register a key path pointing to the shared location of Component\_Shared.
- Install all of the resources of Component\_Application as usual.

If Component\_Shared or its key file is already installed on the computer do not copy files to the shared location of Component\_Shared.

If Component\_Shared or its key file is not yet installed on the computer:

- Copy the files of Component\_Shared to the shared location.
- Process all install actions for Component\_Shared.
- If Component\_Shared is a COM component, register the full COM path such that the syntax [\$Component] and [#FileKey] point to the shared location of Component\_Shared.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Reinstallation of Isolated Components

Windows Installer performs the following actions during reinstallation of an application when the package contains isolated components. Typically, Component\_Shared is a DLL that is shared by Component\_Application and other client executables.

## Reinstallation

- Reinstall of the files of Component\_Shared into the same folder as Component\_Application only if Component\_Application is also being reinstalled.
- Do not increment the client list of Component\_Shared and do not increment the SharedDLL count.
- Recreate the zero-byte file with the short file name of the key file of Component\_Application. This file must be located in the same folder as Component\_Application and have the extension .LOCAL.
- Reinstall all of the resources of Component\_Application as usual.

If the SharedDLL refcount for Component\_Shared is more than 1, or if other products remain on the client list of Component\_Shared:

- Reinstall no files to the shared location of Component\_Shared.

If the SharedDLL refcount for Component\_Shared equals 1, or if there are no other remaining clients of Component\_Shared:

- Reinstall of the files of Component\_Shared into the shared location using the [File Versioning Rules](#).
- Process all reinstall actions for Component\_Shared.
- If Component\_Shared is a COM component, register the full COM path such that the installer syntaxes [\$Component] and [#FileKey] point to the shared location of Component\_Shared.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Removal of Isolated Components

Windows Installer performs the following actions during the removal of an application when the package contains isolated components. Typically, Component\_Shared is a DLL that is shared by Component\_Application and other client executables.

## Uninstall

- Remove the files of Component\_Shared from the folder containing Component\_Application only if Component\_Application is also being removed.
- If the msidbComponentAttributesSharedDllRefCount bit is set in the [Component table](#) decrement the SharedDLL refcount.
- Remove the .LOCAL zero-byte file from the folder containing Component\_Application.
- Remove Component\_Application from the client list of Component\_Shared.
- Remove all of the resources of Component\_Application as usual.

If there are other products remaining on the client list of Component\_Shared:

- Remove no files from the shared location of Component\_Shared.

If the SharedDLL refcount for Component\_Shared is 0 after being decremented, or if there are no other remaining clients of Component\_Shared:

- Remove the files of Component\_Shared from the shared location.
- Process all uninstall actions with respect to this component.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installation Context

Windows Installer can install a package on a computer into two installation contexts: per-machine and per-user. A per-machine installation of the package is required to enable all users of the computer to access and use the application. Because a per-machine installation makes changes to the system that affect all users, standard users having limited privileges may be prevented from installing a package into the per-machine context without first obtaining permission.

You can specify installation context by authoring the package for per-user or per-machine installation and using the **ALLUSERS** and **MSIINSTALLPERUSER** properties. Based on these properties, Windows Installer automatically redirects the values of folder properties and registrations to locations for the per-user or per-machine context.

**Note** The **MSIINSTALLPERUSER** property, available beginning with Windows Installer 5.0 and Windows Server 2008 R2 and Windows 7, can facilitate the development of a single package capable of being installed in either the per-machine or per-user context. For information about developing a dual-purpose package that gives the user the capability to choose the installation context at installation time, see [Single Package Authoring](#). Windows Installer ignores the **MSIINSTALLPERUSER** property if the value of **ALLUSERS** is not 2. Windows Installer always resets the value of **ALLUSERS** to 1 when it installs in the per-machine context and resets the value of **ALLUSERS** to an empty string ("") when it installs in the per-user context.

## Shortcut Redirection

The following table compares the locations of shortcuts for per-machine and per-user installation contexts.

<b>Per-Machine Installation Context (ALLUSERS=1)</b>	<b>Per-User Installation Context (ALLUSERS="")</b>
Applications appear under Add/Remove Programs on Control Panel for all users of the computer.	Applications appear only under Add/Remove Programs on Control Panel for users that have installed the applications.

Shortcuts are installed to the All Users profile.	Shortcuts are installed only to that user's profile.
Icons and transforms are stored in %WINDOWS%\Installer\{ProductCode}.	Icons and transforms are stored in %USERPROFILE%\Application Data\Microsoft\Installer\{ProductCode GUID}

## Registry Redirection

The following table compares the locations of registry entries for the per-machine and per-user installation contexts.

<b>Per-Machine Installation Context (ALLUSERS=1)</b>	<b>Per-User Installation Context (ALLUSERS="")</b>
Windows Installer writes or removes registry values entered in the <a href="#">Registry</a> table and <a href="#">RemoveRegistry</a> table, with the value -1 in the Root column, under HKEY_LOCAL_MACHINE.	Windows Installer writes or removes registry values entered in the Registry table and RemoveRegistry table, with the value -1 in the Root column, under HKEY_CURRENT_USER.
Windows Installer writes or removes registry values entered in the Registry table and RemoveRegistry table, with the value msidbRegistryRootClassesRoot (0) in the Root column, under <b>HKLM\Software\Classes</b> .	Windows Installer writes or removes registry values entered in the Registry table and RemoveRegistry table, with the value msidbRegistryRootClassesRoot (0) in the Root column, under <b>HKCU\Software\Classes</b> .
COM registration is written to <b>HKLM\Software\Classes</b> .	COM registration is written to <b>HKCU\Software\Classes</b> .

## Folder Redirection

Windows Installer sets the values of the folder properties to the full path of the respective folder for the installation context.

**Note** Folders are identified by their **KNOWNFOLDERID** and **CSIDL** constants. Beginning with Windows Vista, applications should use the **SHGetKnownFolderPath** function and the **KNOWNFOLDERID** to determine the full path to the special folders. Existing applications that use the **SHGetFolderPath** function and constant special item IDs (**CSIDL**) will continue to work.

The following table compares the locations of folders that are used when Windows Installer installs the package in the per-machine or per-user installation contexts.

<b>Per-Machine Installation Context (ALLUSERS=1)</b>	<b>Per-User Installation Context (ALLUSERS="")</b>
<p><b>DesktopFolder</b> The full path of the Desktop folder for all users. FOLDERID_PublicDesktop (CSIDL_COMMON_DESKTOPDIRECTORY)</p>	<p><b>DesktopFolder</b> The full path of the Desktop folder for the current user. FOLDERID_Desktop (KNOWNFOLDERID) (CSIDL_DESKTOPDIRECTORY)</p>
<p><b>ProgramMenuFolder</b> The full path of the Program Menu folder for all users. FOLDERID_CommonPrograms (CSIDL_COMMON_PROGRAMS)</p>	<p><b>ProgramMenuFolder</b> The full path of the Program Menu folder for the current user. FOLDERID_Programs (CSIDL_PROGRAMS)</p>
<p><b>StartMenuFolder</b> The full path of the Start Menu folder for the all users. FOLDERID_CommonStartMenu (CSIDL_COMMON_STARTMENU)</p>	<p><b>StartMenuFolder</b> The full path of the Start Menu folder for the current user. FOLDERID_StartMenu (CSIDL_STARTMENU)</p>
<p><b>StartUpFolder</b> The full path of the Start Up folder for all users. FOLDERID_CommonStartup (CSIDL_COMMON_STARTUP)</p>	<p><b>StartUpFolder</b> The full path of the Start Up folder for the current user. FOLDERID_Startup (KNOWNFOLDERID) (CSIDL_STARTUP)</p>

<p><b>TemplateFolder</b> The full path of the Templates folder for all users. FOLDERID_CommonTemplates (CSIDL_COMMON_TEMPLATES)</p>	<p><b>TemplateFolder</b> The full path of the Temp current user. FOLDERID_Templates (CSIDL_TEMPLATES)</p>
<p><b>AdminToolsFolder</b> The full path of the Admin Tools folder for all users. FOLDERID_CommonAdminTools (CSIDL_COMMON_ADMINTOOLS)</p>	<p><b>AdminToolsFolder</b> The full path of the Admin Tools folder for the current user. FOLDERID_AdminTools (CSIDL_ADMINTOOLS)</p>
<p><b>AppDataFolder</b> The full path of the Program Menu folder.  <b>Windows Vista and later:</b> The full path of the Roaming folder.  FOLDERID_RoamingAppData (CSIDL_APPDATA)</p>	<p><b>AppDataFolder</b> The full path of the Program Menu folder.  <b>Windows Vista and later:</b> The full path of the Roaming folder.  FOLDERID_RoamingAppData (CSIDL_APPDATA)</p>
<p><b>CommonAppDataFolder</b> The full path of the folder that contains application data for all users. FOLDERID_ProgramData (CSIDL_COMMON_APPDATA)</p>	<p><b>CommonAppDataFolder</b> The full path of the folder that contains application data for all users. FOLDERID_ProgramData (CSIDL_COMMON_APPDATA)</p>
<p><b>FavoritesFolder</b> The full path of the Favorites folder for the current user. FOLDERID_Favorites (CSIDL_FAVORITES)</p>	<p><b>FavoritesFolder</b> The full path of the Favorites folder for the current user. FOLDERID_Favorites (CSIDL_FAVORITES)</p>
<p><b>PersonalFolder</b> The full path of the My Documents folder or Personal folder for the current user.</p>	<p><b>PersonalFolder</b> The full path of the My Documents folder or Personal folder for the current user.</p>

<p><b>Windows Vista and later:</b> The full path of the Documents folder for the current user.</p> <p>FOLDERID_Documents (CSIDL_PERSONAL)</p>	<p><b>Windows Vista and later:</b> The full path of the Documents folder for the current user.</p> <p>FOLDERID_Documents (CSIDL_PERSONAL)</p>
<p><b>SendToFolder</b> The full path of the SendTo folder.</p> <p>FOLDERID_SendTo (CSIDL_SENDTO)</p>	<p><b>SendToFolder</b> The full path of the SendTo folder.</p> <p>FOLDERID_SendTo (CSIDL_SENDTO)</p>
<p><b>FontsFolder</b> The full path of the System Fonts folder.</p> <p>FOLDERID_Fonts (CSIDL_FONTS)</p>	<p><b>FontsFolder</b> The full path of the System Fonts folder.</p> <p>FOLDERID_Fonts (CSIDL_FONTS)</p>
<p><b>ProgramFilesFolder</b></p> <p><b>32-bit version of Windows:</b> The property value is the full path to the Program Files folder for all users (for example, %ProgramFiles%.) The identifier for this folder is FOLDERID_ProgramFiles (CSIDL_PROGRAM_FILES.) The identifiers FOLDERID_ProgramFiles and FOLDERID_ProgramFilesX86 represent the same folder. Files in this folder can be accessed by all users.</p> <p><b>64-bit version of Windows:</b> The property value is the full path to the Program Files (x86) folder for all users (for example, %ProgramFiles(x86)%.) The identifier for this folder is FOLDERID_ProgramFilesX86 (CSIDL_PROGRAM_FILESX86.) Files in this folder can be accessed by all users.</p>	<p><b>ProgramFilesFolder</b></p> <p><b>Windows Server 2008 and Windows 7:</b> The property value is the full path of the Program Files folder for the current user (for example, %LocalAppData%\Program Files.) The identifier for this folder is FOLDERID_UserProgramFiles and FOLDERID_UserProgramFilesX86 for 32-bit and 64-bit systems. The equivalent CSIDL identifier is CSIDL_USER_PROGRAM_FILES. Files in this folder can be accessed by the current user that installed this folder.</p> <p><b>Windows Server 2008 and Windows Vista and earlier:</b> The property value is the full path of the Program Files folder for the current user (for example, %ProgramFiles% or %ProgramFiles(x86)%). Files in this folder can be accessed by all users.</p>

## CommonFilesFolder

**32-bit version of Windows:** The property value is the full path to the Common Files folder for all users (for example, %ProgramFiles%\Common Files.) The identifier for this folder is FOLDERID\_ProgramFilesCommon (CSIDL\_PROGRAM\_FILES\_COMMON.) The identifiers FOLDERID\_ProgramFilesCommon and FOLDERID\_ProgramFilesCommonX86 represent the same folder. Files in this folder can be accessed by all users.

**64-bit version of Windows:** The property value is the full path to the Common Files folder for all users (for example, %ProgramFiles(x86)%\Common Files.) The identifier for this folder is FOLDERID\_ProgramFilesCommonX86 (CSIDL\_PROGRAM\_FILES\_COMMONX86.) Files in this folder can be accessed by all users.

## ProgramFiles64Folder

The property value is the full path to the Program Files folder for all users (for example, %ProgramFiles%.) The identifier for this folder is FOLDERID\_ProgramFilesX64. There is no equivalent CSIDL identifier to FOLDERID\_ProgramFilesX64. This is the pre-defined folder for 64-bit components and applies to 64-bit systems. Files in this folder can be accessed by all users.

## CommonFilesFolder

**Windows Server 2008**  
**Windows 7:** The property value is the full path of the Common Files folder for the current user (for example, %LocalAppData%\Program Files.) The identifier for this folder is FOLDERID\_UserProgramFiles. On 32-bit and 64-bit systems, there is no equivalent CSIDL identifier. Files in this folder can be accessed by the user that installed this application.

**Windows Server 2008**  
**Windows Vista and earlier:** The user capable folder is a user-capable folder. The identifier for this folder is the same as in the previous version (for example, %ProgramFiles% or %ProgramFiles(x86)%.) Files in this folder can be accessed by all users.

## ProgramFiles64Folder

**Windows Server 2008**  
**Windows 7:** The property value is the full path of the Program Files folder for the current user (for example, %LocalAppData%\Program Files.) The identifier for this folder is FOLDERID\_UserProgramFiles. There is no equivalent CSIDL identifier. Files in this folder can be accessed by the user that installed this application.

**Windows Server 2008**

**Windows Vista and earlier:** The user capable folder is the same as for the per-user context (for example, %ProgramFiles%\Common Files). Files in this folder can be accessed by all users.

**CommonFiles64Folder**  
The property value is the full path to the Common Files folder for all users (for example, %ProgramFiles%\Common Files.) This is the pre-defined folder for 64-bit components and applies to 64-bit systems. The identifier for this folder is FOLDERID\_ProgramFilesCommonX64. There is no equivalent CSIDL identifier to FOLDERID\_ProgramFilesCommonX64. Files in this folder can be accessed by all users.

**CommonFiles64Folder**  
**Windows Server 2008 Windows 7:** The property value is the full path of the Common Files folder for the current user (for example, %LocalAppData%\Program Files). The identifier for this folder is FOLDERID\_UserProgramFiles. There is no equivalent CSIDL identifier for FOLDERID\_UserProgramFiles. Files in this folder can be accessed by the user that installed the application.

**Windows Server 2008 Windows Vista and earlier:** The user capable folder is the same as for the per-user context (for example, %ProgramFiles%\Common Files). Files in this folder can be accessed by all users.

**WindowsFolder**  
The full path of the Windows folder.  
FOLDERID\_Windows (CSIDL\_WINDOWS)

**WindowsFolder**  
The full path of the Windows folder.  
FOLDERID\_Windows (CSIDL\_WINDOWS)

**SystemFolder**  
The full path of the System folder.

**SystemFolder**  
The full path of the System folder.

FOLDERID_SystemX86 (CSIDL_SYSTEMX86)	FOLDERID_SystemX86 (CSIDL_SYSTEMX86)
<p><b>LocalAppDataFolder</b></p> <p>The full path of the folder that contains local (nonroaming) applications.</p> <p>FOLDERID_LocalAppData (CSIDL_LOCAL_APPDATA)</p>	<p><b>LocalAppDataFolder</b></p> <p>The full path of the folder (nonroaming) application</p> <p>FOLDERID_LocalAppD (CSIDL_LOCAL_APPD</p>
<p><b>MyPicturesFolder</b></p> <p>The full path of the Pictures or My Pictures folder.</p> <p>FOLDERID_Pictures (CSIDL_MYPICTURES)</p>	<p><b>MyPicturesFolder</b></p> <p>The full path of the Pictur folder.</p> <p>FOLDERID_Pictures (CSIDL_MYPICTURES)</p>
<p><b>PrintHoodFolder</b></p> <p>The full path of the PrintHood folder.</p> <p>FOLDERID_PrintHood (CSIDL_PRINTHOOD)</p>	<p><b>PrintHoodFolder</b></p> <p>The full path of the PrintH folder.</p> <p>FOLDERID_PrintHood (CSIDL_PRINTHOOD)</p>
<p><b>NetHoodFolder</b></p> <p>The full path of the NetHood folder.</p> <p>FOLDERID_NetHood (CSIDL_NETHOOD)</p>	<p><b>NetHoodFolder</b></p> <p>The full path of the NetH folder.</p> <p>FOLDERID_NetHood (C</p>
<p><b>RecentFolder</b></p> <p>The full path of the Recent folder.</p> <p>FOLDERID_Recent (CSIDL_RECENT)</p>	<p><b>RecentFolder</b></p> <p>The full path of the Recer folder.</p> <p>FOLDERID_Recent (CS</p>

**Note** An application can call the [MsiEnumProducts](#) or [MsiEnumProductsEx](#) functions to enumerate all the products installed on the system. The application can then retrieve information about the installation context of these products by calling the [MsiGetProductInfoEx](#) or [MsiGetProductInfo](#) functions. For information see [Determining Installation Context](#).

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Windows Installer

This section describes how to use Windows Installer to organize applications into components and download an application from an Internet location:

- [Organizing Applications into Components](#)
- [Downloading an Installation from the Internet](#)
- [Multiple-Package Installations](#)
- [Concurrent Installations](#)
- [Configuring Add/Remove Programs with Windows Installer](#)
- [Using Cabinets and Compressed Sources](#)
- [Using Qualified Components](#)
- [Using Transitive Components](#)
- [Localizing a Windows Installer Package](#)
- [Searching for Existing Applications, Files, Registry Entries, or .ini File Entries](#)
- [Authoring a Large Package](#)
- [Determining Installation Context](#)
- [Checking the Installation of Features, Components, and Files](#)
- [CRC Checking During an Installation](#)
- [Installing Permanent Components, Files, Fonts, Registry Keys](#)
- [Removing Stranded Files](#)
- [Searching for a Broken Feature or Component](#)
- [Replacing Existing Files](#)
- [Editing Installer Shortcuts](#)
- [Installing Multiple Instances of Products and Patches](#)
- [Using Windows Installer with a Terminal Server](#)
- [Controlling Feature Selection States](#)
- [Guidelines for Authoring Secure Installations](#)

- [Using Isolated Components](#)
- [Determining the Windows Installer Version](#)
- [Installing a COM+ Application with the Windows Installer](#)
- [Adding or Removing Registry Keys on the Installation or Removal of Components](#)
- [Adding and Removing an Application and Leaving No Trace in the Registry](#)
- [Reducing the Size of an .msi File](#)
- [Changing the Target Location for a Directory](#)
- [Hiding the Cancel Button During an Installation](#)
- [Using Windows Installer with UAC](#)
- [Using Windows Installer with Restart Manager](#)
- [Using Windows Installer with WMI](#)
- [Enumerating Components](#)
- [Using Services Configuration](#)
- [Single Package Authoring](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Organizing Applications into Components

Windows Installer installs and removes an application or product in parts referred to as [components](#). Components are collections of resources that are always installed or removed as a unit from a user's system. A resource can be a file, registry key, shortcut, or anything else that may be installed. Every component is assigned a unique component code [GUID](#).

Authors of installation packages should only create components, and versions of components, that can be installed and removed without damaging other components. Also, the removal of a component should not leave behind any orphaned resources on the user's computer, such as unused files, registry keys, or shortcuts. To ensure this, authors should adhere to the following general rules when organizing resources into components:

- Never create two components that install a resource under the same name and target location. If a resource must be duplicated in multiple components, change its name or target location in each component. This rule should be applied across applications, products, product versions, and companies.
- Note that the previous rule means that two components must not have the same key path file. The key path value points to a particular file or folder belonging to the component that the installer uses to detect the component. If two components had the same key path file, the installer would be unable to distinguish which component is installed. Two components however may share a key path folder.
- Do not create a version of a component that is incompatible with all previous versions of the component. The component may be shared by other applications, products, product versions, and companies. Instead create a new component.
- Do not create components containing resources that will need to be installed into more than one directory on the user's system. The

installer installs all of the resources in a component into the same directory. It is not possible to install some resources into subdirectories.

- Do not include more than one COM server per component. If a component contains a COM server, this must be the key path for the component.
- Do not specify more than one file per component as a target for the **Start** menu or a Desktop shortcut.

When organizing an application into components, package authors may need to add, remove, or modify the resources in an existing installation. In this case, the author must decide whether to provide the resources by introducing a new component or by modifying existing components and changing them into a new version of the component. Because a unique component code must be assigned when a new component is introduced, authors must determine whether their changes require changing the component code. For more information, see [Changing the Component Code](#), [What happens if the component rules are broken?](#), and [Defining Installer Components](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Changing the Component Code

When specifying the [components](#) for an installation, package authors should follow the general rules for component organization described in [Organizing Applications into Components](#). Authors may need to introduce new components or modify existing components. If the addition, removal, or modification of resources effectively creates a new component, then the component code must also be changed.

## Creating a New Component

Introduce a new component and assign it a unique component code when making any of the following changes:

- Any change that has not been shown by testing to be compatible with previous versions of the component. In this case, you must also change the name or target location of every resource in the component.
- A change in the name or target location of any file, registry key, shortcut, or other resource in the component. In this case, you must also change the name or target location of every resource in the component.
- The addition or removal of any file, registry key, shortcut, or other resource from the component. In this case, you must also change the name or target location of every resource in the component.
- Recompiling a 32-bit component into a 64-bit component.

When introducing a new component, authors need to do one of the following to ensure that the component does not conflict with any existing components:

- Change the name or target location of any resource that may be installed under the same name and target location by another component.

- Otherwise guarantee that the new component is never installed into the same folder as another component which has a resource under a common name and location. This includes localized versions of files with the same file name. For more information, see [What happens if the component rules are broken?](#).
- When changing the component code of an existing component, also change the name or target location of every file, registry key, shortcut, and other resource in the component.

## Creating a New Version of a Component

A new version of a component is assigned the same component code as another existing component. Modifying a component without changing the component code is only optional in the following cases:

- The changes to the component have been proven by testing to be backward compatible with all previous versions of the component.
- The author can guarantee that the new version of the component will never be installed on a system where it would conflict with previous versions of the component or applications requiring a previous version. For more information, see [What happens if the component rules are broken?](#).

The component code of a new version of a component must not be changed when it would result in two components sharing resources, such as registry values, files, or shortcuts.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# What happens if the component rules are broken?

In certain cases, authors may decide they need to break the rules for creating components as discussed in [Organizing Applications into Components](#) and [Changing the Component Code](#). Authors need to be aware of the possible consequences of doing this and must otherwise guarantee that their components are never installed where they can damage other applications or components on the user's system.

The following list describes ways that authors sometimes break the recommended component rules and the possible consequences.

An author adds resources to a component without the changing the component code.

- Products installed with the old component have no information about the added resources in their installation database.
- If both a new product that has the added resources and an old product are installed on the same computer, the resources can be left behind if the new product is uninstalled first.
- An old product without the added resources cannot repair the newer version of the component. Reinstalling the old product does not restore the added resources.

An author removes resources from a component without changing the component code.

- Products installed with the new component have no information about the removed resources in their installation database.
- If both an old product, having the resource information, and a new product are installed on the same computer, the resources can be left behind if the old product is uninstalled first.
- A new product with the removed resources cannot repair the older version of the product. Reinstalling the new product does not restore

the removed resources.

An author includes a file that is incompatible with previous versions without changing the component code.

If an incompatible file is included in a component without changing the component code, [default file versioning](#) causes the installer to overwrite the original file with the more recent incompatible file. This can damage old products needing the original file. It may also prevent the installer from repairing the old product because the version of a component's key path file determines the version of the component. If a newer version of the key path file is already installed, the installer does not install an older version of the component. For more information, see [File Versioning Rules](#). In this case, the new product must be removed before the old product can be reinstalled.

- [Default file versioning](#) causes the installer to overwrite the original file with the more recent incompatible file.
- Old products that need the original file are damaged.
- It may also prevent the installer from repairing the old product because the version of a component's key path file determines the version of the component. If a newer version of the key path file is already installed, the installer does not install the older version of the component. For more information, see [File Versioning Rules](#). In this case, the new product must be removed before the old product can be reinstalled.

An author includes the same resource in two different components.

If two components have a resource under the same name and location and both components are installed into the same folder, then the removal of either component removes the common resource, which damages the remaining component.

- Uninstalling either component removes the resource and breaks the other component.
- The component reference-counting mechanism is damaged.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Defining Installer Components

The following outlines how to organize your application into Windows Installer components.

## ▶ To organize an application into components

1. Begin by obtaining a directory and file tree for all of the files and other resources used in your application.
2. Identify any files, registry keys, shortcuts, or other resources that are shared across applications and can be provided by existing components available as [merge modules](#). You must not include any of these resources in the components you author. Instead obtain these components by merging the merge modules into your installation package. The following steps describe how to organize the remaining resources of the application into components.
3. Define a new component for every .exe, .dll, and .ocx file. Designate these files as the key path files of their components. Assign each component a component code GUID.
4. Define a new component for every .hlp or .chm help file. Designate these files as the key path files of their components. Add the .cnt or .chi files to the components holding their associated .hlp and .chm files. Assign each component a component code GUID.
5. Define a new component for every file that serves as a target of a shortcut. Designate these files as the key path files of their components. Assign each component a component code GUID.
6. Group all of the remaining resources into folders. All resources in each folder must ship together. If there is a possibility that a pair of resources may ship separately in the future, put these in separate folders. Define a new component for every folder. Try to

keep the total number of components low to improve performance. Divide the application into many components when it is necessary to have the installer check the validity of the installation thoroughly. Designate any file in the component as the key path file. Assign each component a component code GUID.

7. Add registry keys to the components. Any registry key that points to a file should be included in that file's component. Other registry keys should be logically grouped with the files that require them.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Downloading an Installation from the Internet

Windows Installer accepts a Uniform Resource Locator (URL) as a valid source for an installation. Windows Installer can install packages, patches, and transforms from a URL location.

If the installation database is at a URL, the installer downloads the database to a cache location before starting the installation. The installer also downloads the files and cabinet files from the Internet source that are appropriate for the user's selections. See [A URL-based Windows Installer Installation Example](#) for more information.

For example, to install a package with a source located on a Web server at `http://server/share/package.msi`, you can use the [command line](#) options to install the package and set [public](#) properties.

```
msiexec /i http://server/share/package.msi PROPERTY=VALUE
```

A command line like the one previously shown should be passed to the installer to start an installation from a Web browser. In general, you should not download and install the package simply by double-clicking the .msi file from within the browser. This downloads the .msi file to the temporary Internet files folder and passes the following command to the installer:

```
msiexec /i c:\windows\temporary internet files\package.msi
```

The installation fails if the package requires any external source files or cabinets because these are not located in the same location as the .msi file.

Note that because the [Installer](#) object is not marked as [SafeForScripting](#) on the user's computer, users need to adjust their browser security settings for the example to work correctly.

The [InstallProduct](#) method could be used to run the previous command from a browser as an on-click event.

```
'Downloading an Installation from the Internet  
'The InstallProduct method could be used to run  
'the previous command from a browser as an on-click event.
```

```
<SCRIPT LANGUAGE="VBScript">
<!--
Dim Installer
On Error Resume Next
set Installer=CreateObject("WindowsInstaller.Installer")
Installer.InstallProduct "http://server/share/package.msi",
set Installer=Nothing
-->
</SCRIPT>
```

Note that because some Web servers are case sensitive, the FileName field in the [File](#) table must match the case of the source files exactly to ensure support of Internet downloads.

See [Downloading and Installing a Patch from the Internet](#). For more information about securing installations and using digital certificates, see [Guidelines for Authoring Secure Installations](#) and [Digital Signatures and Windows Installer](#). For more information about how to create a Web installation of a Windows Installer package, see [Internet Download Bootstrapping](#).

## Available Internet Protocols

Beginning with Windows Server 2003 and Windows XP, the installer can use the HTTP, HTTPS and FILE protocols. The installer does not support the FTP and GOPHER protocols.

Windows Installer version 2.0 can use the HTTP, FILE, and FTP protocols and cannot use the HTTPS and GOPHER protocols.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Multiple-Package Installations

Windows Installer can install multiple packages using *transaction processing*. This capability is available beginning with Windows Installer 4.5. The installer will install all the packages belonging to a multiple-package transaction or none of the packages. If all the packages in the transaction cannot be installed successfully, or if the user cancels the installation, the Windows Installer can roll back changes and restore the computer to its original state.

A multiple-package installation package can contain a [MsiEmbeddedChainer table](#) that references a user-defined function that uses the [MsiBeginTransaction](#), [MsiJoinTransaction](#), and [MsiEndTransaction](#) functions.

The [MsiPackageCertificate Table](#) lists digital signature certificates used to verify the identity of the installation packages that make a multiple-package installation. You can use this table to reduce the number of times your multiple-package installation displays a *User Account Control* (UAC) prompt that requires a response by an administrator.

The following Windows Installer functions can make changes to the user's computer when the Windows Installer installs, repairs, updates, or removes applications. Beginning with Windows Installer 4.5, the installer can roll back changes made by these functions during the transaction processing of a multiple-package installation:

- [MsiAdvertiseProduct](#)
- [MsiAdvertiseProductEx](#)
- [MsiApplyMultiplePatches](#)
- [MsiApplyPatch](#)
- [MsiConfigureFeature](#)
- [MsiConfigureProduct](#)
- [MsiConfigureProductEx](#)
- [MsiInstallMissingComponent](#)
- [MsiInstallMissingFile](#)
- [MsiInstallProduct](#)
- [MsiProvideAssembly](#)
- [MsiProvideComponent](#)
- [MsiProvideQualifiedComponent](#)

**MsiProvideQualifiedComponentEx**  
**MsiReinstallFeature**  
**MsiReinstallProduct**  
**MsiRemovePatches**

There is an exception if the Windows Installer encounters a package belonging to a multiple-package installation that contains a [ForceReboot](#) or [ScheduleReboot](#) action. In this case, Windows Installer does not install only that package. Other packages belonging to the multiple-package installation, that do not contain a ForceReboot or ScheduleReboot action, can be installed.

**Windows Installer 4.0 and earlier:** Transaction processing of multiple-package Windows Installer installations is not supported. These versions of the Windows Installer are unable to roll back the installation of multiple packages as a single transaction.

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Concurrent Installations

Concurrent Installations, also called Nested Installations, is a deprecated feature of the Windows Installer. Applications installed with concurrent installations can eventually fail because they are difficult for customers to service correctly. Do not use concurrent installations to install products that are intended to be released to the public. Concurrent installations can have limited applicability in controlled corporate environments when used to install applications that are not intended for public release. The concurrent installations documentation is provided for package authors that wish to use concurrent installations with applications that are not for public distribution.

A concurrent installation action installs another Windows Installer package during a currently running installation. A concurrent installation is added to a package by authoring a concurrent installation action into the [CustomAction table](#) and scheduling this custom action into the sequence tables. The Target field of the CustomAction table contains a string of public property settings used by the concurrent installation. The Source field of the CustomAction table identifies the concurrent package. A concurrent installation action can only reinstall or remove an application that has been installed by the current application's installation package.

The type of concurrent installation action is specified in the Type field of the CustomAction table. Depending upon the custom action type, the package for the concurrent application can reside in a substorage of the main package, as a file at a location specified by a property, or as an advertised application on the user's machine. The following types of custom actions perform a concurrent installation.

<b>Custom action type</b>	<b>Description</b>
<a href="#">Custom Action Type 7</a>	Concurrent installation of a product residing in the installation package.
<a href="#">Custom Action Type 23</a>	Concurrent installation of an installer package within the current source tree.
<a href="#">Custom Action Type 39</a>	Concurrent installation of an advertised installer package.

A concurrent installation shares the same user interface and logging settings as the main installation.

Concurrent installation actions should be placed between the [InstallInitialize action](#) and [InstallFinalize action](#) of the main installation's action sequence. Upon rollback of the main installation, the installer will then rollback the concurrent installation as well. The use of [deferred execution](#) with concurrent installation actions is unnecessary because the installer combines rollback information from the concurrent and main installations. All changes are reversed upon a rollback installation.

The return values for concurrent installation actions are the same as for other custom actions. See [Custom Action Return Values](#).

Standard or custom actions that specify an automatic restart of the system, or request the user to restart, can also perform restart or request from within a concurrent installation.

Once the installer begins a concurrent installation, it locks out all other installations until the concurrent installation is complete and before continuing the main installation. The installer can only execute concurrent installations as synchronous custom actions. See [Synchronous and Asynchronous Custom Actions](#). The option flags described in [Custom Action Return Processing Options](#) must be set to none (+0) or `msidbCustomActionTypeContinue` (+64).

A concurrent installation action can install an application to be run locally, to run from source, to be reinstalled, or to be removed in the same manner as when using [MsiInstallProduct](#) for a regular installation. To specify the type of installation, pass either the [ADDLOCAL](#), [ADDSOURCE](#), [REINSTALL](#), or [REMOVE](#) property to the concurrent installation action.

Concurrent installation actions can be authored in pairs, one action used for installing and the other action used for removing the concurrent installation. A Custom Action Type 7 or Custom Action Type 23 is typically used to install. A Custom Action Type 39 is typically used to remove the concurrent installation when the parent product is uninstalled. The record for the removal custom action in the CustomAction table can have the product code GUID in the Source field and "REMOVE=ALL" in the Target field. The two custom actions need to be authored in the action

sequence table with mutually exclusive conditions. For example, the custom action that installs the product can have "NOT Installed" in its Condition field and the custom action removes the concurrent installation can have REMOVE="ALL" in its Condition field.

There is no method for querying a package for its cost. This makes the costing of a concurrent installations difficult. Rows must be added to the [ReserveCost table](#) to indicate the folders and worst-case costs of the component associated with the concurrent install.

The only [Custom Action Return Processing Options](#) available with concurrent installation actions are none (+0) or msidbCustomActionTypeContinue (+64).

Note that a parent installation cannot call its own package as a concurrent installation action.

Note that if a per-machine installation attempts to run a per-user concurrent installation, the installer registers the parent installation as per-user by default. This can cause the installer to incorrectly remove the application because the installer attempts to uninstall the application per-machine when it is actually registered as per-user. To force the state of a concurrent installation to track the state of its parent installation, enter ALLUSERS="[ALLUSERS]" in the Target column of the [CustomAction table](#). In this case, the concurrent installation is per-machine if the parent is per-machine, and the concurrent installation is per-user if the parent is per-user.

Developers should note the following warnings when authoring concurrent installations.

- Concurrent installations cannot share components.
- An administrative installation cannot also contain a concurrent installation.
- Patching and upgrading may not work with concurrent installations.
- The installer may not properly cost a concurrent installation.
- Integrated ProgressBars cannot be used with concurrent installations.
- Resources that are to be advertised cannot be installed by the

concurrent installation.

- A package that performs a concurrent installation of an application should also uninstall the concurrent application when the parent product is uninstalled.

To prevent a package from ever being installed as a concurrent installation, add either of the following conditional statements to the [LaunchCondition](#) table. This prevents the package from ever being installed by a concurrent installation action run by another installation. This does not prevent the package from being removed by the [RemoveExistingProducts](#) action. See also the [ParentOriginalDatabase](#) property and [ParentProductCode](#) property.

```
"Not ParentProductCode"
```

```
"Not ParentOriginalDatabase"
```

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Type 7

The Custom Action Type 7 is used with concurrent installations. Concurrent installations are not recommended for the installation of applications intended for release to the public. For more information about concurrent installations please see [Concurrent Installations](#).

This custom action installs another installer package that is nested inside of the first package.

## Source

The database of the concurrent application is stored as a substorage of the package, and the name of the substorage is designated in the Source field of the [CustomAction](#) table.

## Numeric Type

Type name	Value
msidbCustomActionTypeInstall + msidbCustomActionTypeBinaryData	7

## Target

The Target field of the [CustomAction](#) table contains property settings to be passed to the concurrent installation. These property settings can specify features.

## Return Processing Options

The concurrent installation session runs as a separate thread in the current process. A concurrent installation cannot run asynchronously.

See [Custom Action Return Processing Options](#).

## Execution Scheduling Options

Options flags are available to control the potential multiple execution of custom actions. See [Custom Action Execution Scheduling Options](#).

## In-Script Execution Options

This custom action does not use this option.

## Return Values

The return status of user exit, failure, suspend, or success from a concurrent installation is processed in the same way as any other action. Note however that Windows Installer translates the return values from all actions when it writes the return value into the log file. For example, if the action return value appears as 1 in the log file, this means that the action returned `ERROR_SUCCESS`. For more information about this translation see [Logging of Action Return Values](#).

Note that if a concurrent install has `msidbCustomActionTypeContinue` set, then a return of `ERROR_INSTALL_USEREXIT`, `ERROR_INSTALL_REBOOT`, `ERROR_INSTALL_REBOOT_NOW`, or `ERROR_SUCCESS_REBOOT_REQUIRED` is treated as `ERROR_SUCCESS`. This means that if you set `msidbCustomActionTypeContinue` and your concurrent installation requires a restart, the requirement for the restart will be ignored. Additionally, the error code from the concurrent installation custom action will be ignored.

If `msidbCustomActionTypeContinue` is not set, the following return codes plus `ERROR_SUCCESS` are treated as success and have the following meanings. Other return codes are treated as failure.

Message	Meaning
<code>ERROR_INSTALL_REBOOT</code>	The restart flag will be set to restart at end of the installation.
<code>ERROR_INSTALL_REBOOT_NOW</code>	A restart is required before completing the installation. The restart will be processed immediately.

ERROR_SUCCESS_REBOOT_REQUIRED	A restart was required, but was suppressed.
-------------------------------	---

## Remarks

A conditional expression is required to enable the concurrent installation at either installation or removal of the associated component or feature.

## See Also

[Concurrent Installations](#)

[Custom Action Reference](#)

[About Custom Actions](#)

[Using Custom Actions](#)

[Custom Action Return Values](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Type 23

The Custom Action Type 23 is used with concurrent installations. Concurrent installations are not recommended for the installation of applications intended for release to the public. For information about concurrent installations please see [Concurrent Installations](#).

This custom action installs another installer package that resides in the application's source tree.

## Source

The location of the concurrent installation package is specified relative to the root of the source location shown in the Source field of the [CustomAction table](#).

## Numeric Type

Type name	Value
msidbCustomActionTypeInstall + msidbCustomActionTypeSourceFile	23

## Target

The Target field of the [CustomAction table](#) contains property settings that are to be passed to the concurrent installation. These property settings can specify features.

## Return Processing Options

The concurrent installation session runs as a separate thread in the current process. A concurrent installation cannot run asynchronously.

For more information, see [Custom Action Return Processing Options](#).

## Execution Scheduling Options

Options flags are available to control the potential multiple execution of custom actions. For more information, see [Custom Action Execution Scheduling Options](#).

## In-Script Execution Options

Not used.

## Return Values

The return status of user exit, failure, suspend, or success from a concurrent installation is processed in the same way as any other action. Note however, that Windows Installer translates the return values from all actions when it writes the return value into the log file. For example, if the action return value appears as 1 in the log file, this means that the action returned `ERROR_SUCCESS`. For more information, see [Logging of Action Return Values](#).

Note that if a concurrent installation has `msidbCustomActionTypeContinue` set, then a return of `ERROR_INSTALL_USEREXIT`, `ERROR_INSTALL_REBOOT`, `ERROR_INSTALL_REBOOT_NOW`, or `ERROR_SUCCESS_REBOOT_REQUIRED` is treated as `ERROR_SUCCESS`. This means that if you set `msidbCustomActionTypeContinue` and your concurrent installation requires a restart, the requirement for the restart will be ignored. Additionally, the error code from the concurrent installation custom action will be ignored.

If `msidbCustomActionTypeContinue` is not set, the following return codes plus `ERROR_SUCCESS` are treated as success and have the following meanings. Other return codes are treated as failure.

Message	Meaning
<code>ERROR_INSTALL_REBOOT</code>	The restart flag will be set to restart at end of the installation.
<code>ERROR_INSTALL_REBOOT_NOW</code>	A restart is required before completing the installation.

	The restart will be processed immediately.
ERROR_SUCCESS_REBOOT_REQUIRED	A restart was required, but was suppressed.

## Remarks

A conditional expression is required to enable the concurrent installation at either installation or removal of the associated component or feature.

## See Also

[Concurrent Installations](#)  
[Custom Action Reference](#)  
[About Custom Actions](#)  
[Using Custom Actions](#)  
[Custom Action Return Values](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Type 39

The Custom Action Type 39 is used with concurrent installations. Concurrent installations are not recommended for the installation of applications intended for release to the public. For information about concurrent installations please see [Concurrent Installations](#).

Type 39 custom action installs an application that is advertised or already installed. This custom action type may be used to reinstall or remove a product that has been installed as a concurrent installation by the current product's installation package. The Type 39 custom action cannot be used to reinstall or remove any product previously installed by any other means. For example, if the secondary product is installed using a Type 39, Type 23, or Type 7 custom action during the installation of the primary product, a Type 39 custom action may be used to remove the secondary product when the primary product is uninstalled.

## Source

The Source field of the [CustomAction table](#) contains the product code for the application.

## Numeric Type

Type name	Value
msidbCustomActionTypeInstall + msidbCustomActionTypeDirectory	39

## Target

The Target field of the [CustomAction table](#) contains property settings that are to be passed to the concurrent installation. These property settings can specify features.

## Return Processing Options

The custom action type 39 fails if the application is not advertised or

installed. To avoid this failure, you must set the `msidbCustomActionTypeContinueflag`.

A concurrent install cannot run asynchronously.

See [Custom Action Return Processing Options](#).

## Execution Scheduling Options

Options flags are available to control the potential multiple execution of custom actions. See [Custom Action Execution Scheduling Options](#).

## In-Script Execution Options

The custom action does not use this option.

## Return Values

The return status of user exit, failure, suspend, or success from a concurrent installation is processed in the same way as any other action. Note however that Windows Installer translates the return values from all actions when it writes the return value into the log file. For example, if the action return value appears as 1 in the log file, this means that the action returned `ERROR_SUCCESS`. For more information, see [Logging of Action Return Values](#).

Note that if a concurrent installation has `msidbCustomActionTypeContinue` set, then a return of `ERROR_INSTALL_USEREXIT`, `ERROR_INSTALL_REBOOT`, `ERROR_INSTALL_REBOOT_NOW`, or `ERROR_SUCCESS_REBOOT_REQUIRED` is treated as `ERROR_SUCCESS`. This means that if you set `msidbCustomActionTypeContinue` and your concurrent installation requires a restart, the requirement for the restart will be ignored. Additionally, the error code from the concurrent installation custom action will be ignored.

If `msidbCustomActionTypeContinue` is not set, the following return codes plus `ERROR_SUCCESS` are treated as success and have the following meanings. Other return codes are treated as failure.

---

Message	Meaning
ERROR_INSTALL_REBOOT	The restart flag will be set to restart at end of the installation.
ERROR_INSTALL_REBOOT_NOW	A restart is required before completing the installation. The restart will be processed immediately.
ERROR_SUCCESS_REBOOT_REQUIRED	A restart was required, but was suppressed.

## Remarks

A conditional expression is required to enable the concurrent installation at either installation or removal of the associated component or feature.

## See Also

[Concurrent Installations](#)  
[Custom Action Reference](#)  
[About Custom Actions](#)  
[Using Custom Actions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE11

The ICE11 is used with concurrent installations. Concurrent installations are not recommended for the installation of applications intended for release to the public. For information about concurrent installations please see [Concurrent Installations](#).

## Result

ICE11 validates the Source column of the [CustomAction table](#) of concurrent installation custom actions. The Source column must contain a valid [GUID](#) (MSI product code).

ICE11 posts an error if the Source column of the CustomAction table is authored incorrectly for concurrent installation custom actions.

## Example

ICE posts the following error messages for the example shown.

```
CustomAction: CA4 is a nested install of an advertised MSI.  
CustomAction: CA1 is a nested install of an advertised MSI.  
CustomAction: CA2 is a nested install of an advertised MSI.
```

### [Property Table](#) (partial)

Property	Value
ProductCode	{BFB69273-F0AE-45C4-9853-0AF946714768}

### [CustomAction Table](#) (partial)

CustomAction	Type	Source
CA1	39	{BFB69273-F0AE-45C4-9853-0AF946714768}
CA2	39	{BFB69273-F0AE-55c5-9853-0AF946714768}
CA3	39	{BFB69273-F0AE-66C6-9853-0AF946714768}
CA4	39	ProductCode

To fix the errors, for CA1, you cannot do a concurrent installation of the "base package". This would result in a recursive installation. This entry should be removed or the Source column should be changed to a GUID for an advertised MSI that differs from the base package's GUID. For CA2, make all characters of the GUID uppercase. Lastly, change CA4's Source column to reference a valid GUID of an advertised MSI.

## **See Also**

[Concurrent Installations](#)  
[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# \_MSIExecute Mutex

The `_MSIExecute` Mutex is set only while processing the [InstallExecuteSequence table](#), [AdminExecuteSequence table](#), or [AdvtExecuteSequence table](#).

Because two installations cannot be run in the same process, an attempt to call the installer's application programming interface (API) returns `ERROR_INSTALL_ALREADY_RUNNING` (1618) in two cases:

- While the `_MSIExecute` Mutex is set.
- While the current process is processing the [InstallUISequence table](#) or [AdminUISequence table](#).

See the [Event Logging](#) messages for information about what application is being installed.

In cases where it is impractical to return an `ERROR_INSTALL_ALREADY_RUNNING` error, you can retrieve the current status of the Windows Installer service before attempting to start the installation by using the **QueryServiceStatusEx** function. The Windows Installer service is currently running if the value of the **dwControlsAccepted** member of the returned **SERVICE\_STATUS\_PROCESS** structure is `SERVICE_ACCEPT_SHUTDOWN`.

**Windows Installer 2.0:** Not supported. The use of the **QueryServiceStatusEx** function to retrieve the current status of the Windows Installer service requires Windows Installer version 3.0 or greater.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# **\_\_MsiPromptForCD Mutex**

The \_\_MsiPromptForCD Mutex exists when the installer prompts the user to insert a CD-ROM. Autoplay programs should check that the \_\_MsiPromptForCD mutex is not currently set before starting. Note that it is possible for a CD-ROM prompt to occur before either the InProgress key or the \_MSIExecute Mutex exist.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Configuring Add/Remove Programs with Windows Installer

You can supply all of the information needed to configure Add/Remove Programs in Control Panel by setting the values of certain installer properties in your application's Windows Installer package. Setting these properties automatically writes the corresponding values into the registry. If the installer detects that the product is marked for complete removal, operations are automatically added to the script to remove the Add/Remove Programs folder in Control Panel information for the product.

If an application is not registered, it is not listed in Add/Remove Programs in Control Panel. For more information, see [Adding and Removing an Application and Leaving No Trace in the Registry](#).

Applications that have been installed in the per-user [installation context](#) are displayed in the Add/Remove Programs of the current user.

Applications that have been installed in the per-machine installation context are displayed in the Add/Remove Programs of all users.

Applications that have not been installed per-machine, and have only been installed as per-user applications for users other than the current user, do not appear in the Add/Remove Programs of the current user.

Note that installation packages that use the [LIMITUI](#) property must also contain the [ARPNOMODIFY](#). This is required for a user to obtain the correct behavior from Add/Remove Programs in Control Panel utility when attempting to configure a product.

The installer uses the following [public properties](#) to manage Add/Remove Programs in Control Panel.

Property name	Brief description of property
<a href="#">ARPAUTHORIZEDCDFPREFIX</a>	URL of the update channel for the application. The value the installer writes under the <a href="#">Uninstall Registry Key</a> .
<a href="#">ARPCOMMENTS</a>	Provides Comments for the Add/Remove Programs in the Control

	Panel. The value the installer writes under the <a href="#">Uninstall Registry Key</a> .
<b>ARPCONTACT</b>	Provides the Contact for Add/Remove Programs in the Control Panel. The value the installer writes under the <a href="#">Uninstall Registry Key</a> .
<b>ARPINSTALLLOCATION</b>	Fully qualified path to the application's primary folder. The value the installer writes under the <a href="#">Uninstall Registry Key</a> .
<b>ARPHELPLINK</b>	Internet address, or URL, for technical support. The value the installer writes under the <a href="#">Uninstall Registry Key</a> .
<b>ARPHELPTELEPHONE</b>	Technical support phone numbers. The value the installer writes under the <a href="#">Uninstall Registry Key</a> .
<b>ARPNOMODIFY</b>	Prevents display of a Change button for the product in Add/Remove Programs in the Control Panel. <b>Note</b> This only affects the display in the ARP. The Windows Installer is still capable of repairing, installing-on-demand, and uninstalling applications through a command line or the programming interface.
<b>ARPNOREMOVE</b>	Prevents display of a Remove button for the product in the Add/Remove Programs in the Control Panel. The product can still be removed by selecting the Change button if the installation package has been authored with a user interface that provides product removal as an option. <b>Note</b> This only affects the display in the ARP. The Windows Installer is still

	capable of repairing, installing-on-demand, and uninstalling applications through a command line or the programming interface.
<b>ARNOREPAIR</b>	Disables the Repair button in the Add/Remove Programs in the Control Panel. <b>Note</b> This only affects the display in the ARP. The Windows Installer is still capable of repairing, installing-on-demand, and uninstalling applications through a command line or the programming interface.
<b>ARPPRODUCTICON</b>	Identifies the icon displayed in Add/Remove Programs. If this property is not defined, Add/Remove Programs specifies the display icon.
<b>ARPREADME</b>	Provides the ReadMe for Add/Remove Programs in Control Panel. The value the installer writes under the <a href="#">Uninstall Registry Key</a> .
<b>ARPSIZE</b>	Estimated size of the application in kilobytes.
<b>ARPSYSTEMCOMPONENT</b>	Prevents display of the application in the Programs List of the Add/Remove Programs in the Control Panel. <b>Note</b> This only affects the display in the ARP. The Windows Installer is still capable of repairing, installing-on-demand, and uninstalling applications through a command line or the programming interface.
<b>ARPURLINFOABOUT</b>	URL for application's home page. The value the installer writes under the

	<a href="#">Uninstall Registry Key.</a>
<b>ARPURLUPDATEINFO</b>	URL for application update information. The value the installer writes under the <a href="#">Uninstall Registry Key</a> .

**Note** For information regarding the Set Program and Defaults tool, please refer to the section [Working with Set Program Access and Computer Defaults](#) .

## See Also

[Uninstall Registry Key](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Cabinets and Compressed Sources

This section discusses compressed and uncompressed sources and adding cabinet files to an installation:

- [Compressed and Uncompressed Sources](#)
- [Cabinet Files](#)
- [Including a Cabinet File in an Installation](#)
- [Ordering File Sequence Numbers in a Cabinet, File Table and Media Table](#)
- [Digital Signatures and External Cabinet Files](#)
- [Cabinet Data Type](#)
- [Generate File Cabinet](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Compressed and Uncompressed Sources

Package authors can reduce the size of their installation packages by compressing the source files and including them in [cabinet files](#). The source file image can be compressed, uncompressed, or a mixture of both types.

## Compressed Sources

A source consisting entirely of compressed files should include the compressed flag bit in the **Word Count Summary** Property. The compressed source files must be stored in cabinet files located in a data stream inside the .msi file or in a separate cabinet file located at the root of the source tree. All of the cabinets in the source must be listed in the [Media table](#).

## Uncompressed Sources

A source consisting entirely of uncompressed source files should omit the compressed flag bit from the **Word Count Summary** Property. All of the uncompressed files in the source must exist in the source tree specified by the [Directory table](#).

## Mixed Sources

To mix compressed and uncompressed source files in the same package, override the **Word Count Summary** property default by setting the `msidbFileAttributesCompressed` or `msidbFileAttributesNoncompressed` bit flags on particular files. These bit flags are set in the Attributes column of the [File table](#) if the compression state of the file does not match the default specified by the **Word Count Summary** property.

For example, if the **Word Count Summary** property has the compressed flag bit set, all files are treated as compressed into a cabinet. Any uncompressed files in the source must include `msidbFileAttributesNoncompressed` in the Attributes column of the [File table](#). The uncompressed files must be located at the root of the source tree.

If the **Word Count Summary** property has the uncompressed flag set, files are treated as uncompressed by default and any

compressed files must include `msidbFileAttributesCompressed` in the `Attributes` column of the `File` table. All of the compressed files must be stored in cabinet files located in a data stream inside the `.msi` file or in a separate cabinet file located at the root of the source tree.

For more information, see [Using Cabinets and Compressed Sources](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Cabinet Files

A cabinet is a single file, usually with a .cab extension, that stores compressed files in a file library. The cabinet format is an efficient way to package multiple files because compression is performed across file boundaries, which significantly improves the compression ratio.

Developers can use a cabinet file creation tool such as Makecab.exe to make cabinet files for use with installer packages. The Makecab.exe utility is included in the [Windows SDK Components for Windows Installer Developers](#).

Developers can also use a cabinet file creation tool such as Cabarc.exe to make cabinet files for use with installer packages. This tool writes to the Diamond cabinet structure.

The file keys of the files stored inside of a cabinet file must match the entries in the File column of the [File table](#) and the sequence of files in the cabinet must match the file sequence specified in the Sequence column. For more information, see [Using Cabinets and Compressed Sources](#).

Large files can be split between two or more cabinet files. There can be no more than 15 files in any one cabinet file that spans to the next cabinet file. For example, if you have three cabinet files the first cabinet can have 15 files that span to the second cabinet file and the second cabinet file can have 15 files that span to the third cabinet file.

A cabinet file can be located inside or outside of the .msi file. To conserve disk space, the installer always removes any cabinets that are embedded in the .msi file before caching the installation package on the user's computer.

The installer extracts files from a cabinet as they are needed by the installation and installs them in the same order as they are stored in the cabinet file. The space requirements for installing a file stored in a cabinet are no different than for installing an uncompressed file.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Including a Cabinet File in an Installation

This section describes including cabinet files in installations. For more information, see [Using Cabinets and Compressed Sources](#).

## ► To include a cabinet file in a installation package

1. Use a cabinet creation tool to compress the source files into a cabinet file. See [Cabinet Files](#).
2. The cabinet file must either be located in a data stream inside the .msi file or in a separate cabinet file located at the root of the source tree specified by the [Directory Table](#).
3. Determine whether the source is to be a compressed type or a mixed type that has both uncompressed and compressed files. See [Compressed and Uncompressed Sources](#). Depending on the type of source image, set the compressed or uncompressed flag bits of the [Word Count Summary](#) Property.
4. Add a record to the [File table](#) for each of the files in the cabinet. Enter a file key in the File column that exactly matches the file key of the file in the cabinet. The file keys are case-sensitive. The file installation sequence in the File table and the cabinet must also be the same. The file sequence is specified by the sequence number in the Sequence column. To arrive at the sequence number for the first file in the cabinet, do the following. Find the existing record in the [Media table](#) having the greatest value in the DiskID column. The LastSequence field of this record gives the last file sequence number used on the media. In the File table, assign the first file of the new cabinet a sequence number that is greater than this. Assign sequence numbers to all of the remaining files in the same order as in the cabinet file. For a description of the remaining record fields, see [File table](#).

5. Add a record to the [Media table](#) for the cabinet. Specify a value in the DiskID field of this new record that is greater than the largest DiskID value already existing in the table. Put the name of the cabinet into the Cabinet field. This name must be in the form of a [Cabinet](#) data type. Prefix the name with a number sign "#" if the cabinet is a data stream stored in the .msi file. Note that if the cabinet is a data stream, the name of the cabinet is case-sensitive. If the cabinet is a separate file, the name of the file is not case-sensitive.
6. Determine the greatest file sequence number in the new cabinet by checking the Sequence column of the updated File table. Enter a value that is greater than this into the LastSequence field of the new record of the Media table. For a description of the remaining record fields, see [Media table](#).
7. You can store the cabinet file in the installation package either by using a tool such as Msidb.exe or by using the installer's [Database Functions](#). The following four steps explain how to add the cabinet from a program by using the database functions.
8. To add the cabinet to the installation package from a program open a view on the [\\_Streams table](#) of the database using [MsiDatabaseOpenView](#).
9. Use [MsiRecordSetString](#) to set the Name column of the [\\_Streams](#) table to the name appearing in the Cabinet column of the [Media table](#). Omit the number sign: #.
10. Use [MsiRecordSetStream](#) to set the Data column of the [\\_Streams](#) table to the cabinet's data.
11. Use [MsiViewModify](#) to update the record in the [\\_Streams](#) table.
12. To use Msidb.exe to add the cabinet file Mycab.cab to the installation package named Mydatabase.msi, use the following command line: Msidb.exe -d mydatabase.msi -a mycab.cab. In this case, the Cabinet column of the Media table should contain

the string: #mycab.cab.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Ordering File Sequence Numbers in a Cabinet, File Table and Media Table

The [File table](#) contains a complete list of all the source files for the installation. Files can be stored on the source media as individual files or compressed within [cabinet files](#). The sequence numbers in the Sequence column of the File table, together with the LastSequence field of the [Media table](#), specify both the order of installation for files and the source media on which each file is located. Each record in the Media table identifies the source disk containing all the files with sequence numbers less than or equal to the value shown in the LastSequence column and greater than the LastSequence value of the previous disk.

For example, suppose a file has a sequence number of 92 entered in the Sequence column of the File table. To determine on which source disk this file resides, the installer checks the record of the Media table for the entry with the smallest LastSequence value that is larger than 92. The DiskId column is the primary key for the Media table and this field uniquely identifies the disk in the table.

The maximum limit on the number of files that can be listed in the File table of a Windows Installer package is 32767 files. To create a Windows Installer package containing more files, see [Authoring a Large Package](#).

Package authors can reduce the size of their installation packages by compressing the source files and including them in cabinet files. The source file image can be compressed, uncompressed, or a mixture of both types. For more information about compressed and uncompressed sources see [Compressed and Uncompressed Sources](#). Compressed source files must be stored inside of a cabinet file. The compressed files inside a cabinet have their own internal sequence numbers. The values of these internal sequence numbers do not need to match the value of the sequence numbers within the File table. However, the sequence of the files specified in the File table must be identical to the actual sequence of the files within the cabinets. The sequence numbers of uncompressed files need not be unique. For example, if all the files are uncompressed and reside on one disk, all the files can have the same sequence number in the File table.

The Media table describes the set of disks that make up the source media for the installation. The first entry in the Media table must always have a 1 in the DiskId field. Files should be organized on the source media such that all the files on disk 1 have File table sequence numbers that are smaller than the sequence numbers of files on disk 2, and all of the sequence numbers on disk 2 should be smaller than on disk 3, and so on. This requirement also applies to a disk that contains both compressed and uncompressed sources. For example, if the media sources for the installation are located on two source disks, and if disk 1 contains both uncompressed files and a cabinet file, then both of the uncompressed files and the files in the cabinet must have sequence numbers smaller than the smallest file sequence number of any file stored on disk 2. If all files on disk 1 are compressed in a cabinet file, the Media table could be authored as shown in the following table.

#### Media Table (partial)

DiskId	LastSequence	DiskPrompt	Cabinet	VolumeLabel
1	5	1	mycab.cab	Disk 1
2	10	2		Disk 2

If some files on disk 1 are compressed in a cabinet and some are uncompressed, the Media table could be authored as follows.

#### Media Table (partial)

DiskId	LastSequence	DiskPrompt	Cabinet	VolumeLabel
1	5	1		Disk 1
2	10	1	mycab.cab	Disk 1
3	15	2		Disk 2

Note that the authoring in the following Media table is incorrect because it specifies some file sequence numbers on disk 2 that are smaller than some files inside the cabinet on disk 1.

#### Media Table



DiskId	LastSequence	DiskPrompt	Cabinet	VolumeLabel
1	5	1		Disk 1
2	10	2		Disk 2
3	15	1	mycab.cab	Disk 1

Large files can be split between two or more cabinet files. There can be no more than 15 files in any one cabinet file that spans to the next cabinet file. For example, if you have three cabinet files, the first cabinet can have 15 files that span to the second cabinet file, and the second cabinet can have 15 files that span to the third cabinet file. When you add a record to the File table for a file multiple cabinets, use the first part of the file to specify the file sequence number you enter in the Sequence column.

The File and Media tables could be authored as follows if there are three files, two cabinets, and two disks. In this example, c1.cab resides on disk1 and c2.cab resides on disk2. The file f2 spans both cabinets. The c1.cab cabinet contains the entire f1 file and the first part of file f2. The c2.cab cabinet contains the second part of f2 and the entire f3 file.

#### Media Table (partial)

DiskId	LastSequence	DiskPrompt	Cabinet	VolumeLabel
1	5	1	c1.cab	Disk 1
2	10	2	c2.cab	Disk 2

#### File Table (partial)

File	Sequence
f1	1
f2	2
f3	6

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Qualified Components

Qualified components are a method of indirection and can be used to group components with parallel functionality into categories.

To return the full path and install a [qualified component](#), call [MsiProvideQualifiedComponent](#) or [MsiProvideQualifiedComponentEx](#).

To enumerate all of the qualified component qualifiers and descriptive strings, call [MsiEnumComponentQualifiers](#).

## ► To group components together into a qualified-component category

1. There must be a record in the [Component table](#) for each component that is included in the new category of qualified components. Author the fields in the Component table the same as for ordinary components. Note that each qualified component must have a unique component ID GUID entered in the ComponentId column of the Component table.
2. Generate a qualifier text string for each qualified component. The qualifier must be unique text string that can be easily generated when searching for a qualified component. For example, if the components in the category are being qualified by language, the numeric locale identifier (LCID) is a reasonable qualifier string.
3. Add a record in the [PublishComponent table](#) for each qualified component. Enter the qualified-component identifiers from the Component column of the Component table into the Component\_ column of the PublishComponent table. Enter the qualifier strings for each qualified component into the Qualifier column. Enter a localized string to be displayed to the user and describing the qualified component into the optional AppData column. An explanatory string should be put in the AppData field, such as "French Dictionary," rather than just the numeric LCID. Enter the

name of the feature that uses this component into the Feature\_ column. The feature identifier in this field must also be listed in the Feature column of the [Feature table](#).

4. Generate a category GUID for this category of qualified components. This must be a valid [GUID](#). If you use a utility such as GUIDGEN to generate the GUID be sure that it contains only uppercase letters. For every qualified component in this category, enter the category GUID into the ComponentId field of the [PublishComponent table](#).

The following example illustrates how the "FAX Templates" category of qualified components are authored into the Component, Feature, and PublishComponent tables.

#### [PublishComponent table](#)

<b>ComponentId</b>	<b>Qualifier</b>	<b>AppData</b>	<b>Feature_</b>	<b>Component_</b>
{FAX Template Category GUID}	1033	US English template	FAXTemplate	FAXTemplateENU
	1041	Japanese template	FAXTemplate	FAXTemplateJPN
	1054	Thai template	FAXTemplate	FAXTemplateTHA
	1031	German template	FAXTemplate	FAXTemplateDEU

#### [Component table](#) (partial table)

<b>Component</b>	<b>ComponentId</b>
FAXTemplateENU	{FAX Template (US English) component GUID}
FAXTemplateJPN	{FAX Template (Japanese) component GUID}
FAXTemplateTHA	{FAX Template (Thai) component GUID}

FAXTemplateDEU

{*FAX Template (German) component GUID*}

### Feature table (partial table)

<b>Feature</b>
FAXTemplate
FAXTemplate
FAXTemplate
FAXTemplate

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Transitive Components

A typical use for transitive components is to prepare a product to reinstall during a system upgrade. The author of the installation package specifies those components that need to be swapped out during a system upgrade as having the transitive attribute. When the user later upgrades the system, the product must be reinstalled. Upon this reinstall, the installer removes the earlier components and installs the later components, without having to install the entire product.

## ► To include two transitive components in the installation package

1. Include both transitive components in the installation package.
2. Author both transitive components into the [Component table](#) the same as regular components. Each transitive component must have its own unique GUID specified in the ComponentId column.
3. Include the msidbComponentAttributesTransitive bit in the Attributes column of the Component table for each transitive component. If this bit is set, the installer reevaluates the value of the statement in the Condition column upon a reinstall.

If the value was previously False and has changed to True, the installer installs the component.

If the value was previously True and has changed to False the installer removes the component even if the component has other products as clients.

**Note** Unless the transitive bit is set, the component remains enabled once installed even if the conditional statement evaluates to False on a subsequent maintenance installation of the product. The conditions must be based only on computer states. Do not use with conditions based on user states or properties set on the command line because this can cause the installer to require a reinstallation of the product on each use by a different user.

4. Enter complementary conditional expressions into the Condition fields of the Control table such that when the condition on the first transitive component changes to False the condition on the second transitive component changes to True. This results in the removal of the first component and installation of the second component upon reinstallation of the application.

A reinstallation of the product is necessary to switch the transitive components. Package authors therefore need to provide users with a method for reinstalling the product and for setting the modes of the **REINSTALLMODE** property. There are basically three ways to trigger the reinstallation:

- Run and configure the reinstallation through the user interface by authoring a package that uses the *full UI*.
- Run the reinstallation from the command line by using **msiexec /f** and select the modes from the list for the */f command line option*.
- Have the application call **MsiReInstallProduct** or **MsiReInstallFeature**.

The bit should only be used with conditions based on computer states. Do not use with conditions based on user states or properties set on the command line because this can cause the installer to require a reinstallation of the product on each use by a different user.

**Note** Unless the Transitive bit in the Attributes column is set for a component, the component remains enabled once installed even if the conditional statement in the Condition column evaluates to False on a subsequent maintenance installation of the product.

In most cases, if an application includes transitive components, Windows Installer requires the application's source to repair or upgrade the application. In these cases, the system restoration CD-ROM shipped by an original equipment manufacturer does not work and an actual installation source for the application needs to be provided.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Localizing a Windows Installer Package

For general information about localization, see Globalization Services. Localizing a Windows Installer package requires modifying the strings displayed by the user interface and may also require adding or modifying product resources. For example, localization may include the addition of international DLLs and localized files to the product.

## ▶ To localize a Windows Installer package

1. Prepare for localization when authoring the original installation package. Design the layout of localized files such that different language versions can safely coexist when installed on the user's computer. Organize files requiring localization into separate components and install these files into separate directories. Author a base installation database that has a neutral control page. See [Preparing a Windows Installer package for localization](#).
2. Always set the code page of the database being localized prior to adding any localized data. If the code page of the database being localized is neutral, see [Setting the code page of a database](#). To determine the code page see [Determining an installation database's code page](#).
3. Import a localized [Error table](#) and [ActionText table](#) into the database. For more information, see [Localizing the Error and ActionText Tables](#) for a list of languages supported by the Microsoft Windows Software Development Kit (SDK). You may import these tables using Msidb.exe or [MsiDatabaseImport](#).
4. Modify any of the other localizable columns in the database using a table editor or SQL queries. For the SQL access functions, see [Working with Queries](#). The topics for the database tables identify which database columns can be localized. For more information,

see the list of tables in [Database Tables](#).

5. Set the **ProductLanguage** property in the [Property table](#) to the LANGID of the database. When authoring a package as language neutral, set the ProductLanguage property to 0 and use the MS Shell Dlg font as the text style for all authored dialog boxes. Because some fonts do not support all of the character sets, you can ensure that the text is correctly displayed on all localized versions of the operating system by using this font.
6. Set the language field of the **Template Summary** property to reflect the LANGID of the database.
7. If the text strings in the [summary information stream](#) are localized, set the **Codepage Summary** property to the code page.
8. Set the **ProductCode** property in the [Property table](#) and set the package code in the **Revision Number Summary** property to a new package code. A localized product is considered a different product. For example, the German and English versions of an application are considered two different products and must have different product codes.
9. Localization may require modifying resources that already exist or the addition of new resources such as files or registry keys. Check to be sure that the component code is changed for every existing component that has had a new resource added. Other modifications may also require changes to a component's code. For more information see [Changing the Component Code](#).
10. Be sure to save localization and other changes to the database by saving the package with the editing tool or by calling **MsiDatabaseCommit**.

For more information, see [A Localization Example](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Code Page Handling (Windows Installer)

The Windows Installer stores all database strings in a single shared string pool to reduce the size of the database, and to improve performance. For a list of numeric code pages, see [Localizing the Error and ActionText Tables](#).

For more information, [Determining an Installation Database's Code Page](#).

Windows Installer uses **IsValidCodePage** to determine whether the code page is valid.

## Localizing a Windows Installer Package

If you localize a Windows Installer package, it may involve modifying information in database tables, exporting the tables to ANSI text archive files, and then importing the archive files into the database that is being localized. You can also add localization changes to a database by using a database table editor or the [Database Functions](#). It is important to set the code page of the database that is being localized before you make any localization changes to the database. Do not set the code page of the database after localizing the database, because this can corrupt extended characters. For more information, see [Setting the Code Page of a Database](#).

The recommended approach for handling code pages is to author a neutral database that only contains characters that can be translated into any code page. For more information, see [Creating a Database with a Neutral Code Page](#).

If you add localization information with database archive files, you can use **MsiDatabaseExport** to export tables from a database that contains localization changes to ANSI text archive files, and then import these into the database being localized with **MsiDatabaseImport**. The code page of an exported archive file is always the same as its parent database. The code pages of an imported file and the database that is receiving the file must be identical, or at least one of the two code pages must be neutral. For more information, see [Code Page Handling of Imported and](#)

## Exported Tables.

If you add localization information with a text editor or the [Database Functions](#) be careful to only pass string parameters to the Windows Installer API that uses the code page of the database that is being localized. If a string parameter contains characters not represented by the code page of the database, an error occurs when calling [MsiDatabaseCommit](#). For more information, see [Code Page Handling of Parameter Strings](#).

If one package is used to install multiple language versions of a product, the transform that is used to localize strings can also change the code page of the database.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Code Page Handling of Imported and Exported Tables

You can add localization information to an installation database by importing and exporting ASCII text archive files using [MsiDatabaseExport](#) and [MsiDatabaseImport](#). Because the database string pool uses an ANSI code page, both the database and exported [Text Archive Files](#) have code pages.

When a Text Archive File is exported from a database, the code page of the archive file is the same as the parent database. For a list of numeric code pages, see [Localizing the Error and ActionText Tables](#).

**Note** Exporting a table to a text archive file translates the control characters to avoid conflicts with file delimiters.

## ASCII Text Archive Files

The ASCII Text Archive Files exported by [MsiDatabaseExport](#) are explained in the following format:

- The names of the table columns are written on the first line.
- The column formats are written on the second line.
- If the table contains only ASCII data, the third line of the text file is the table name followed by a list of the primary keys.
- If the table contains non-ASCII data and the database is stamped with a numeric code page, the code page number appears at the beginning of the third line.
- If the database contains non-ASCII data, but the database is not stamped with the numeric code page, the current system code page number is written at the beginning of the third line.
- The remaining lines of the text file are the data in the specified code page.
- If a table contains streams, [MsiDatabaseExport](#) exports each

stream in the table to a separate file.

## Neutral and Non-Neutral Code Pages

You can facilitate localization by starting with a database that has a neutral code page:

- A blank database has a neutral code page.
- A database that does not contain extended characters that require a code page to be represented in ASCII has a neutral code page.

For more information, see [Creating a Database with a Neutral Code Page](#).

Neutral and non-neutral code pages have the following characteristics:

- If a Text Archive File with a non-neutral code page is imported into a database that has a different non-neutral code page, the Installer returns an error when **MsiDatabaseImport** is called.
- A Text Archive File that has a neutral code page can be imported into a database that has any code page.
- A Text Archive File that has any code page can be imported into a database that has a neutral code page.
- Importing a Text Archive File into a database with a neutral code page sets the code page of the database to the archive file code page. All archive files subsequently imported into the database must have the same code page as the first file.

For more information, see [Determining an Installation Database Code Page](#) and [Setting the Code Page of a Database](#).

The Text Archive Files that are exported by **MsiDatabaseExport** can be used with version control systems. Use the [Database Functions](#) or a database table editor to edit the database.

You can add localization information to an installation database by using a database table editor or the Windows Installer API. For more information, see [Code Page Handling of Parameter Strings](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Code Page Handling of Parameter Strings

You can add localization information to an installation database by using a database table editor such as Orca that is provided with the Windows Installer SDK, or by calling the [Database Functions](#) from an application. Be careful to only pass string parameters that use the code page of the database that is being localized. If a string parameter contains characters that cannot be represented by the code page of the database, the Installer returns an error when calling [MsiDatabaseImport](#). For a list of numeric code pages, see [Localizing the Error and ActionText Tables](#).

For more information, see [Determining an Installation Database's Code Page](#).

## Adding Localization Information to a Database

When you add localization information to a database, the code page of the database must be supported by the operating system. It does not have to be the current code page of the system. **IsValidCodePage** must return TRUE for the database code page. Because the system converts ANSI strings to Unicode, there is an error if the current user code page is not the same as the database code page.

Calling the ANSI version of the Windows Installer API converts the localized string to Unicode by using the current system code page. When the database is committed, the Unicode string is converted to ANSI using the code page of the database. If the current system code page differs from the code page of the localized string, the result can be a loss of data and incorrect string conversion.

The following procedure shows you how to store the localization data.

### ► To store localization data

1. Set the code page of the database to the code page of the localized string.
2. Convert the ANSI string to Unicode by using the

**MultiByteToWideChar** function, and specify the code page of the localized data.

3. Call the Unicode version of the Windows Installer API by using the Unicode string to add the localized data.
4. Commit the localization changes to the database by using **MsiDatabaseCommit**.

You can also add localization information to an installation database by importing and exporting ASCII text archive files. For more information, see [Code Page Handling of Imported and Exported Tables](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Creating a Database with a Neutral Code Page

The recommended approach for handling code pages is to author a neutral base database that only contains characters that can be translated into any code page. The database may then be set to the code page of the localization and the localization information can be added as described in [Localizing a Windows Installer Package](#).

To author a neutral database, avoid extended characters that do not belong to the ASCII character set and therefore require a special code page. For example, the copyright sign, ©, and the registered trademark sign, ®, are not ASCII characters, and require a special ANSI code page for proper display. Instead use (c) and (r), because these characters can be translated or transformed to the symbols for the English-language version. This neutral database can then be localized by setting its code page and adding localization information by table editing or by importing text archive files.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Localizing the Error and ActionText Tables

The Microsoft Windows Software Development Kit (SDK) includes localized resource strings, localized [Error tables](#), and localized [ActionText tables](#) for the languages listed in the following table. The LANGIDs marked with asterisks indicate that the resource strings are stored as the base language and so can be used with all dialects of that language.

You can import the localized Error and ActionText tables into your database by using Msidb.exe or [MsiDatabaseImport](#).

<b>LANGID (decimal)</b>	<b>LANGID (hexadecimal)</b>	<b>ASCII code page</b>	<b>Abbreviation</b>	<b>Language</b>	<b>Language- Culture</b>
1025*	0x401	1256	ARA	Arabic - Saudi Arabia	ar-SA
1027	0x403	1252	CAT	Catalan	ca-ES
1028*	0x404	950	CHT	Chinese - Taiwan	zh-TW
2052	0x804	936	CHS	Chinese - China	zh-CN
1029	0x405	1250	CSY	Czech - Czech Republic	cs-CZ
1030	0x406	1252	DAN	Danish - Denmark	da-DK
1031*	0x407	1252	DEU	German - Germany	de-DE
1032	0x408	1253	ELL	Greek - Greece	el-GR
1033*	0x409	1252	ENU	English - United	en-US

				States	
3082*	0xC0A	1252	ESN	Spanish - Modern Sort - Spain	es-ES
1061	0x425	1257	ETI	Estonian	et-EE
1035	0x40B	1252	FIN	Finnish - Finland	fi-FI
1036*	0x40C	1252	FRA	French - France	fr-FR
1037	0x40D	1255	HEB	Hebrew - Israel	he-IL
1038	0x40E	1250	HUN	Hungarian - Hungary	hu-HU
1040*	0x410	1252	ITA	Italian - Italy	it-IT
1041	0x411	932	JPN	Japanese - Japan	jp-JP
1042	0x412	949	KOR	Korean - Korea	ko-KO
1063	0x427	1257	LTH	Lithuanian	lt-LT
1062	0x426	1257	LVI	Latvian	lv-LV
1043*	0x413	1252	NLD	Dutch - Netherlands	nl-NL
1044*	0x414	1252	NOR	Norwegian (Bokmål)- Norway	nb-NO
1045	0x415	1250	PLK	Polish - Poland	pl-PL
1046	0x416	1252	PTB	Portuguese - Brazil	pt-BR
2070	0x816	1252	PTG	Portuguese	pt-PT

				- Portugal	
1048	0x418	1250	ROM	Romanian - Romania	ro-RO
1049	0x419	1251	RUS	Russian - Russia	ru-RU
1050	0x41A	1250	HRV	Croatian - Croatia	hr-HR
1051	0x41B	1250	SKY	Slovak - Slovakia	sk-SK
1053*	0x41D	1252	SVE	Swedish - Sweden	sv-SE
1054	0x41E	874	THA	Thai - Thailand	th-TH
1055	0x41F	1254	TRK	Turkish - Turkey	tr-TR
1060	0x424	1250	SLV	Slovenian - Slovenia	sl-SI
1066	0x42A	1258	VIT	Vietnamese - Viet Nam	vi-VN
1069	0x42D	1252	EUQ	Basque	eu-ES

**Windows Vista:** In addition to the languages listed in the previous table, the languages in the following table are available beginning with Windows Vista.

<b>LANGID (decimal)</b>	<b>LANGID (hexadecimal)</b>	<b>ASCII code page</b>	<b>Abbreviation</b>	<b>Language</b>	<b>Language-Culture</b>
1026	0x402	1251	BGR	Bulgarian	bg-BG
1058	0x422	1251	UKR	Ukrainian	uk-UA
2074	0x81A	1250	SRL	Serbian	sr-Latn-CS

				(Latin)	
--	--	--	--	---------	--

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Determining an Installation Database's Code Page

To determine the code page of a database, call [MsiDatabaseExport](#) with *hDatabase* set to the handle of the database and *szTableName* set to `_ForceCodepage`. This exports a text file with an `.idt` extension. The first two lines of this file are blank. The third line is the ANSI code page number, followed by a tab, followed by the name `_ForceCodepage`. See also [Code Page Handling of Imported and Exported Tables](#).

An example of determining the code page by using the [Export method](#) is provided in the Windows Installer SDK as a part of the utility `WiLangId.vbs`. For more information about using `WiLangId.vbs` see the topic: [Manage Language and Codepage](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Localizing the Language Displayed by Dialogs

The Windows Installer service uses the current user's language in all dialogs until a Windows Installer package is opened. On Windows 2000 and later operating systems, the Windows Installer determines this by using [GetUserDefaultUILanguage](#). On operating systems earlier than Windows 2000, it uses [GetUserDefaultLangID](#). When a Windows Installer package is opened, the installer displays dialogs using the package's language as specified by the [Template Summary](#) Property.

For more information about localizing the language of a Windows Installer package, see also [A Localization Example](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Setting the Code Page of a Database

Always set the code page of a database prior to adding any localization information. Attempting to set the code page after entering data into the database is not recommended because this could corrupt extended characters. Localization can be greatly facilitated by starting with a database that is code page neutral. For details, see [Creating a database with a neutral code page](#). You may determine the current code page of a database as described in [Determining an installation database's code page](#). See [Localizing the Error and ActionText Tables](#) for a list of numeric code pages.

You may set the code page of a blank database, or a database with a neutral code page, by importing a text archive file having a non-neutral code page with [MsiDatabaseImport](#). This sets the code page of the database to the imported file's code page. All archive files subsequently imported into the database must then have the same code page as the first file. If a text archive file is exported from a database, the code page of the archive file is the same as the parent database. See [Code Page Handling of Imported and Exported Tables](#).

The code page of any database may be set to a specified numeric code page by using [MsiDatabaseImport](#) to import a text archive file with the following format: Two blank lines; followed by a line containing the numeric code page, a tab delimiter, and the exact string: `_ForceCodepage`. Note that with Windows 2000, this translates all of the strings in the database to the code page of `_ForceCodepage`. This may be intended when localizing an existing database and translating all non-neutral strings to the new code page. However, this causes an error if the database contains non-neutral strings that are not to be translated.

The utility `WiLangId.vbs` provides an example of how to set the code page of a package using the [Import method](#). A copy of `WiLangId.vbs` is provided in the Windows Installer SDK. You can use this utility to determine the language versions that are supported by the database (Package), the language the installer uses for any strings in the user interface that are not authored into the database (Product), or the single ANSI code page for the string pool (Codepage). For information on using `WiLangId.vbs` see the help page: [Manage Language and Codepage](#).

To determine the values of Product, Package, and Codepage, run WiLangId.vbs as follows.

**cscript wilangid.vbs** *[path to database]*

To set the Codepage of the package run the following command line.

**cscript wilangid.vbs** *[path to database]* **Codepage** *[value]*

For example, to set the Codepage of test.msi to the numeric ANSI code page value 1252, run the following command line.

**cscript wilangid.vbs c:\temp\test.msi Codepage 1252**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Preparing a Windows Installer Package for Localization

Localization of a Windows Installer package into multiple languages can be greatly facilitated by doing the following:

- Author a base installation database that is code page neutral. See [Creating a database with a neutral code page](#). The code page of the localized database can then be set by importing a text archive table with a non-neutral code page into the base database. See [Setting the code page of a database](#).
- Organize files requiring localization into separate components and install these files into separate directories. This ensures that two localized packages never install identically named files into the same directory.

For example, a worldwide application using the following resources may have three components.

Component	Resource
WORLD	worldwide.exe
WORLD	worldwide registry entries
WORLD	worldwide shortcut
ENG	engui.dll
ENG	readme.txt
FRA	fraui.dll
FRA	readme.txt

The files that need to be localized may be installed into the following directory locations:

- [ProgramFilesFolder]\World\worldwide.exe
- [ProgramFilesFolder]\World\English\engui.dll
- [ProgramFilesFolder]\World\English\readme.txt
- [ProgramFilesFolder]\World\French\fraui.dll
- [ProgramFilesFolder]\World\French\readme.txt

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Searching for Existing Applications, Files, Registry Entries or .ini File Entries

Windows Installer can search for a specific file or directory during an installation. File or directory searches are used to determine whether a user has already installed a version of an application.

[AppSearch Action](#) searches a user system for file signatures that are specified in the [AppSearch Table](#). If the AppSearch action finds an installed file or directory with the specified signature, it sets a corresponding property, also specified in the AppSearch Table, to the location of the file or directory. When searching for a file, the file signature must also be listed in the [Signature Table](#). If a file signature is listed in the AppSearch Table and is not listed in the Signature Table, the search looks for a directory, registry entry, or .ini file entry.

To expedite the search of a user computer, the Installer queries the following locator database tables in the order listed for a suggested search location:

- If the file signature is listed in the [CompLocator Table](#), the suggested search location is the key path of a component. If the signature is not listed in this table or not installed at the suggested location, the Installer queries the [RegLocator Table](#) for a suggested location.
- If the file signature is listed in the RegLocator Table, the suggested search location is a key path written in the user registry. If the signature is not listed in this table or not installed at the suggested location, the Installer queries the [IniLocator Table](#) for a suggested location.
- If the file signature is listed in the IniLocator Table, the suggested search location is a key path written in an .ini file present in the default Windows directory of a user system. If the signature is not listed in this table or not installed at the suggested location, the

Installer queries the [DrLocator Table](#) for a suggested location.

- If the file signature is listed in the DrLocator Table, the suggested search location is a path in the user directory tree. The depth of subdirectory levels to search below this location is also specified in this table.

The first time the Installer finds the file signature at a suggested location, it stops searching for this file or directory, and sets the corresponding property in the AppSearch Table. For more information, see the following:

- [Searching All Fixed Drives for a File](#)
- [Searching for a Directory and a File in the Directory](#)
- [Searching for a File and Creating a Property Holding the File's Path](#)
- [Searching for a File in a Specific Location](#)
- [Searching for a Registry Entry and Creating a Property Holding the Value of the Registry](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Searching All Fixed Drives for a File

## ► To search all fixed drives for a file

1. Enter the file signature and name in the [Signature Table](#). The remaining fields in this record can be null to search for any version of MyApp.exe.

Signature Table (partial)

Signature	File Name
AppFile	MyApp.exe

2. Enter the property that the Installer is to set if MyApp.exe is installed.

[AppSearch Table](#)

Property	Signature
MYAPP	AppFile

3. Use the [DrLocator Table](#). Leave the Parent and Path fields empty to search all fixed drives of the user system. Specify in the Depth column the number of subdirectory levels to search. For example, setting Depth to 0 detects c:\MyApp.exe, but does not detect the file at a depth of 2, for example: c:\Program Files\MyApps\MyApp.exe.

DrLocator Table

Signature	Parent	Path	Depth
AppFile			3

4. Include the AppSearch action in the action sequence. If MyApp.exe is installed, the Installer sets the property MYAPP to the location of the file. If the file is installed, MYAPP evaluates as True in a conditional expression.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Searching for a File in a Specific Location

► To search for a file in a specific location on a user system

1. List the file signature and name in the [Signature Table](#). The remaining fields in this record can be null to search for any version of MyApp.exe.

Signature Table

Signature	File name
AppFile	MyApp.exe

2. Enter the property that the Installer is to set if MyApp.exe is installed.

[AppSearch Table](#) (partial)

Property	Signature
MYAPP	AppFile

3. Use the [DrLocator Table](#). Enter the full path to the file on the user system in the Path field. Enter a value of 0 into the Depth column to search the bin folder.

DrLocator Table

Signature	Parent Path	Depth
AppFile	C:\Program Files\MyProducts\Projects\bin	0

4. Include the AppSearch action in the action sequence. If MyApp.exe is installed in C:\Program Files\MyProducts\Projects\bin, the Installer sets the property MYAPP to the location of file.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Searching for a Directory and a File in the Directory

► To search for a directory and then a file in that directory

1. First search for the directory.

AppDir must be defined as the valid signature of the directory. If AppDir is not defined as a valid signature, AppSearch does not have a place to find the file, for example, if the search is for c:\MyDir\MyApp.exe, AppDir should be defined as c:\MyDir. AppDir might be defined by including a record in the [DrLocator Table](#), or by some other method. No record is included in the [Signature Table](#) for the directory search. For the file search, list the file signature and name in the Signature Table. The remaining fields in this record can be null to search for any version of MyApp.exe.

Signature Table (partial)

Signature	File Name
AppFile	MyApp.exe

2. Use the [AppSearch Table](#).

Enter the property that the Installer is to set if the directory with the signature AppDir is installed. If the Installer finds this directory is installed, it sets MYDIR to the directory path. Enter the property that the Installer is to set if MyApp.exe is installed.

AppSearch Table (partial)

Property	Signature

MYDIR	AppDir
MYAPP	AppFile

3. Use the DrLocator Table.

Enter in the Parent column the signature, AppDir, that is defined as the path of the directory. Specify in the Depth column the number of subdirectory levels to search in this directory. AppDir must be defined as the directory signature. AppDir may be defined by including a record as shown here or by another method.

DrLocator Table

Signature	Parent	Path	Depth
AppDir		C:\MyDir	0
AppFile	AppDir		0

4. Include the AppSearch action in the action sequence.

If MyApp.exe is found to be installed in AppDir, the Installer sets the property MYAPP to the location of file.

## See Also

[Searching for Existing Applications, Files, Registry Entries or .ini File Entries](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Searching for a File and Creating a Property Holding the File's Path

► To search for a file and create a property holding the path of that file

1. First do a search for the file by listing the file signature and name in the [Signature Table](#).

The remaining fields of this record can be left empty to specify a search for any version of MyApp.exe.

Signature Table (partial)

Signature	File name
AppFile	MyApp.exe

2. Next, specify the path of the file that is being searched for in the [DrLocator Table](#).

Because AppFolder is not listed in the Signature Table, the Installer determines that AppFolder is a folder rather than a file.

DrLocator Table

Signature	Parent	Path	Depth
AppFile			
AppFolder	AppFile		

3. Finally, populate the [AppSearch Table](#) so that the [AppSearch Action](#) returns the path of AppFolder.

After the Installer executes the AppSearch action, the value of MYFOLDER is the full path of AppFolder.

AppSearch Table (partial)

<b>Property</b>	<b>Signature</b>
MYFOLDER	AppFolder

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Searching for a Registry Entry and Creating a Property Holding the Value of the Registry

► To search for a registry entry and create a property holding the value of that file

1. Do not add the signature to the [Signature Table](#) or the [CompLocator Table](#). If a file signature is listed in the [AppSearch Table](#) and is not listed in the Signature or CompLocator tables, the Installer looks in the [RegLocator Table](#).
2. Specify the registry entry to be searched for in the RegLocator Table. If the signature is absent from the Signature Table and the value of the Type column is msidbLocatorTypeRawValue, then the search is assumed to be for the specific registry key name pointed to by the RegLocator Table.

RegLocator Table (partial)

Signature_	Root	Key	Name	Type
AppValue	2	SOFTWARE\Microsoft\MyApp	Myname	msidb

3. Finally, populate the AppSearch Table so that the [AppSearch Action](#) returns the value of AppValue. After the Installer executes the AppSearch Action, the value of MYREGVAL is the value of AppValue.

AppSearch Table (partial)

Property	Signature
MYREGVAL	AppValue

---

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring a Large Package

Use this guideline to author a Windows Installer package that contains more than 32767 files.

If your Windows Installer package contains more than 32767 files, you must change the schema of the database to increase the limit of the following columns:

- The Sequence column of the [File table](#).
- The LastSequence column of the [Media table](#).
- The Sequence column of the [Patch table](#).

For more information, see [Column Definition Format](#).

## ► To increase the limit of a database column

1. Export the table to an .idt file. For details, see [Msidb.exe](#), [Export Files](#), and [Importing and Exporting](#).
2. Edit the .idt file to change the column type from i2 to i4, or from I2 to I4.
3. Export the [\\_Validation](#) table to an .idt file.
4. Edit the .idt file to change the values in the MaxValue column of the [\\_Validation](#) table to accommodate the increased column widths.
5. Import the .idt files back into the database.

Note that transforms and patches cannot be created between two packages with different column types.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Determining Installation Context

An application can call the [MsiEnumProducts](#) or [MsiEnumProductsEx](#) functions to enumerate products that are installed or advertised on the system. This function can enumerate all the products installed in the per-machine [installation context](#). It can enumerate the products installed in the per-user context for the current user. The application can retrieve information about the context of these products by calling the [MsiGetProductInfoEx](#) or [MsiGetProductInfo](#) functions.

The Windows Installer can install products to run with elevated (system) privileges for non-administrator users. This requires the permission of an administrator user. A product that is installed with elevated privileges is called "managed." All products installed per-machine are managed. Products installed per-user are only managed if a local system agent performs an advertisement while impersonating a user. This is the method used by software deployment through Group Policy. Per-user applications installed while the [AlwaysInstallElevated](#) policies are set are not considered managed. By calling [MsilsProductElevated](#), an application can check whether a particular product is managed.

The following sample demonstrates how an application determines context by using [MsiEnumProducts](#), [MsiGetProductInfo](#), and [MsilsProductElevated](#).

```
#ifndef UNICODE
#define UNICODE
#endif //UNICODE

#ifdef _WIN32_MSI
#define _WIN32_MSI 200
#endif //_WIN32_MSI

#include <stdio.h>
#include <windows.h>
#include <msi.h>
#pragma comment(lib, "msi.lib")

const int cchGUID = 38;
```

```

UINT DetermineContextForAllProducts()
{
    WCHAR wszProductCode[cchGUID+1] = {0};
    WCHAR wszAssignmentType[10] = {0};
    DWORD cchAssignmentType =
        sizeof(wszAssignmentType)/sizeof(wszAssignmentType[0]);
    DWORD dwIndex = 0;

    DWORD cchProductName = MAX_PATH;
    WCHAR* lpProductName = new WCHAR[cchProductName];
    if (!lpProductName)
    {
        return ERROR_OUTOFMEMORY;
    }

    UINT uiStatus = ERROR_SUCCESS;

    // enumerate all visible products
    do
    {
        uiStatus = MsiEnumProducts(dwIndex,
                                   wszProductCode,
                                   cchProductCode,
                                   cchAssignmentType,
                                   cchAssignmentType,
                                   &wszAssignmentType,
                                   &cchAssignmentType,
                                   &fPerMachine,
                                   &fManaged);
        if (ERROR_SUCCESS == uiStatus)
        {
            cchAssignmentType =
                sizeof(wszAssignmentType)/sizeof(wszAssignmentType[0]);
            BOOL fPerMachine = FALSE;
            BOOL fManaged = FALSE;

            // Determine assignment type of product
            // This indicates whether the product
            // instance is per-user or per-machine
            if (ERROR_SUCCESS ==
                MsiGetProductInfo(wszProductCode,
                                  &wszProductCode,
                                  &cchProductCode,
                                  &wszAssignmentType,
                                  &cchAssignmentType,
                                  &fPerMachine,
                                  &fManaged))
            {
                if (L'1' == wszAssignmentType[0])
                    fPerMachine = TRUE;
            }
            else
            {
                // This halts the enumeration
                // could be logged and enumerated
            }
        }
    } while (uiStatus == ERROR_SUCCESS);
}

```

```

        // remainder of the product
        uiStatus = ERROR_FUNCTION_F
        break;
    }

    // determine the "managed" status of
    // If fManaged is TRUE, product is
    // and runs with elevated privilege
    // If fManaged is FALSE, product is
    // run as the user.
    if (ERROR_SUCCESS != MsiIsProductE

    {
        // This halts the enumerati
        // could be logged and enu
        // remainder of the product
        uiStatus = ERROR_FUNCTION_F
        break;
    }

    // obtain the user friendly name of
    UINT uiReturn = MsiGetProductInfo(v
    if (ERROR_MORE_DATA == uiReturn)
    {
        // try again, but with a la
        delete [] lpProductName;

        // returned character count
        // terminating NULL
        ++cchProductName;

        lpProductName = new WCHAR[c
        if (!lpProductName)
        {
            uiStatus = ERROR_OU
            break;
        }

        uiReturn = MsiGetProductInf
    }

    if (ERROR_SUCCESS != uiReturn)

```

```

        {
            // This halts the enumerati
            // could be logged and enur
            // remainder of the product
            uiStatus = ERROR_FUNCTION_F
            break;
        }

        // output information
        wprintf(L" Product %s:\n", lpProduct
        wprintf(L"\t%s\n", wszProductCode);
        wprintf(L"\tInstalled %s %s
        fPerMachine ? L"per-machine
        fManaged ? L"managed" : L"r
    }
    dwIndex++;
}
while (ERROR_SUCCESS == uiStatus);

if (lpProductName)
{
    delete [] lpProductName;
    lpProductName = NULL;
}

return (ERROR_NO_MORE_ITEMS == uiStatus) ? ERROR_SU
}

```

## See Also

[MsiEnumProducts](#)

[MsiGetProductInfo](#)

[MsiGetProductInfoEx](#)

[MsilsProductElevated](#)

[Installing a Package with Elevated Privileges for a Non-Admin](#)

[Advertising a Per-User Application To Be Installed with Elevated Privileges](#)

[Installation Context](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Checking the Installation of Features, Components, Files

If after running an installation, you need to verify that a particular feature, component, or file has been installed, turn on the verbose logging option during the installation. See [Windows Installer Logging](#) and [Command Line Options](#).

The verbose log includes an entry for each feature and component the installation package may install. The log tells what the state of that feature or component was prior to the installation, what state was requested by the installation, and in what state the installer left the feature or component. Feature and component entries in the log appear as the following examples.

```
MSI (s) (40:A4): Feature: QuickTest; Installed: Absent; Requested: Absent; Local; Action: Local
MSI (s) (40:A4): Component: QuickTest; Installed: Absent; Requested: Absent; Local; Action: Local
```

This verbose log indicates that:

- the installation state of the QuickTest feature and component was absent before running the package
- the package requested a local installation of these
- the feature and component were both left in the locally installed state after running the package.

The label "Installed" in the log refers to the current install state of the feature or component, "Request" refers to the requested install state of the feature or component. "Action" refers to the actual action state of the feature or component.

The following table lists the possible component or feature states that can appear in the log.

Log Entry	Description

Request: Null	No request.
Action: Null	No action taken.
Installed: Absent	Component or feature is not currently installed.
Request: Absent	Installation requests component or feature be uninstalled.
Action: Absent	Installer actually uninstalls component or feature.
Installed: Local	Component or feature is currently installed to run local.
Request: Local	Installation requests component or feature be installed to run local.
Action: Local	Installer actually installs component or feature to run local.
Installed: Source	Component or feature is currently installed to run from source.
Requested: Source	Installation requests that component or feature be installed to run from source.
Action: Source	Installer actually installs the component or feature to run from source.
Installed: Advertise	Feature is currently advertised. Components are never advertised.
Request: Advertise	Installation requests feature be installed as an advertised feature.
Action: Advertise	Installer actually installs the feature as an advertised feature.
Request: Reinstall	Installation requests feature be reinstalled. Components do not use reinstall state.
Action: Reinstall	Installer actually reinstalls feature.
Installed: Current	Feature is currently installed in the default authored install state.
Request: Current	Installation requests feature be installed in the default authored install state.
Action: Current	Installer actually installs the feature in the default

	authored install state.
Action: FileAbsent	Installer actually uninstalls component's files and leaves all other resources of the component installed.
Action: HKCRAbsent	Installer actually removes component's HKCR information. File and non-HKCR information remain.
Action: HKCRFileAbsent	Installer actually removes component's HKCR information and files. All other resources of the component remain.

The verbose log has an entry for each file that may be installed by the package. The log tells what was done to the file and provides some explanation. File entries in the log appear as in the following example.

```
MSI (s) (40:A4): File: C:\Test\TESTDB.EXE; Won't Overwrite;
file is of an equal version
```

This log indicates that the installer will not overwrite the existing Testdb.exe file because the existing file is the same as the version being installed.

**Note** If you need to author an installation package that searches for an existing file or directory on the user's computer during an installation, use the method described in [Searching for Existing Applications, Files, Registry Entries or .ini File Entries](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# CRC Checking During an Installation

A Cyclic Redundancy Check (CRC) of files is available with Windows Installer. CRC checking is an error-checking mechanism, similar to a checksum, that enables an application to determine whether the information in a file has been modified. After the Windows Installer finishes copying a file, it gets a CRC value from both the source and destination files. The installer checks the original CRC stamped into the file and compares this to the CRC calculated from the copy. The CRC check fails if the original CRC value is non-null and is different from the CRC calculated on the copy. If the original CRC is null, no check occurs.

The Windows Installer does a CRC check on a file in the following cases:

- If the MSICHECKCRCS property is set and msidbFileAttributesChecksum is included in the Attributes field of the file's record in the [File table](#). The installer does the CRC check once after installing, duplicating, or moving the file.
- If the MSICHECKCRCS property is set and msidbFileAttributesChecksum is included in the Attributes field of the file's record in the [File table](#), the installer does a CRC check after patching the file.
- If the msidbFileAttributesChecksum is included in the Attributes field of the file's record in the [File table](#), the installer does a CRC check before binding images.

If the check fails before binding an image, the installer reports the following two errors in the log file and continues the installation without binding the file.

Code	Message
2941	Unable to compute the CRC for file [2].
2942	BindImage action has not been executed on [2] file.

If the check fails after an uncompressed file had been copied, duplicated,

or patched, the installer reports the following error. This error is also reported if the check fails after a compressed file is copied. If the file has the msidbFileAttributesVital attribute, the file is considered vital to the installation and the user gets the option to retry or cancel the installation. If the file is marked as nonvital in the Attributes column of the [File table](#), the user may ignore the error and continue, retry, or cancel the installation.

Code	Message
1331	Failed to correctly copy [2] file: CRC error.

Note that only uncompressed files are moved. If the check fails after an uncompressed file is moved, the installer displays the following error. If the file has the msidbFileAttributesVital attribute, the file is considered vital to the installation and the installation fails. If the file is marked as nonvital in the Attributes column of the [File table](#), the user gets the option to cancel or to ignore the error and continue the installation.

Code	Message
1332	Failed to correctly move [2] file: CRC error.

If the check fails after an uncompressed file is patched, the installer displays the following error. If the file has the msidbFileAttributesVital attribute, the file is considered vital to the installation and the installation fails. If the file is marked as nonvital in the Attributes column of the [File table](#), the user gets the option to cancel or to ignore the error and continue the installation.

Code	Message
1333	Failed to correctly patch [2] file: CRC error.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installing Permanent Components, Files, Fonts, Registry Keys

To install a file, font, or registry key so that it is not removed when the product is uninstalled, the entire component containing the file, font, or registry key must be made permanent. To make a component permanent, set **msidbComponentAttributesPermanent** in the Attributes column of the [Component table](#).

Note that the installer removes a registry key after removing the last value or subkey under the key. To prevent an empty registry key from being removed on uninstall, write a dummy value under the key you need keep, and enter + in the Name column of the [Registry table](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Removing Stranded Files

If a file that should have been removed from the user's computer remains installed after running an uninstall, the installer may not be removing the component containing the file for one or more of the following reasons:

- The `msidbComponentAttributesPermanent` bit was set for the component in the `Attributes` column of the [Component table](#).
- No value was entered for the component in the `ComponentId` column of the `Component` table.
- The component is used by another application or feature that is still installed.
- There is a condition specified in the [Condition](#) table that enables a feature during installation and disables the feature during uninstallation.
- The key file for the component has a previous reference count under **HKLM\Software\Microsoft\Windows\CurrentVersion\SharedDLLs**
- The component is installed in the `System` folder and any file in the component has a previous reference count under **HKLM\Software\Microsoft\Windows\CurrentVersion\SharedDLLs**
- The Windows Installer does not remove any files or registry keys that are protected by Windows Resource Protection (WRP). For more information, see [Using Windows Installer and Windows Resource Protection](#). On Windows Server 2003, Windows XP, and Windows 2000, the installer does not remove any files that are protected by Windows File Protection (WFP). If a component's key path file or registry key is protected by WFP or WRP, the installer does not remove the component.

**Note** Because Windows Installer does not install, update, or remove any resource that is protected by WRP, you should not include protected resources in an installation package. Instead, use only the supported resource replacement mechanisms described in the

Windows Resource Protection section.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Searching for a Broken Feature or Component

The installer can increase application [resiliency](#) by automatically reinstalling damaged components. Specifically, the installer reinstalls a component or feature if it finds that the file or registry key specified in the KeyPath column of the [Component table](#) is missing.

If the KeyPath of a feature's component is damaged in the source, or if there is an error in how the KeyPath is authored in the database, the installer may attempt to open an installation package and reinstall the feature each time the feature's shortcut is activated.

To determine the cause of repeated attempts to reinstall a feature or application, check the Event Log for two entries such as the following.

```
Detection of product 'MyProduct', feature 'MyFeature' failed  
request for component 'MyComponent'  
Detection of product 'MyProduct', feature 'MyFeature', component  
'MyComponent' failed
```

The first message states which component in the product's package was being installed. This is the component referenced in the Component\_ column of the [Shortcut table](#).

The second message states which component is failing detection. This is the component with the missing or damaged KeyPath that's triggering the reinstall.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Replacing Existing Files

Because unnecessary file copying slows an installation, the Windows Installer determines whether the component's key file is already installed before attempting to install the files of any component. If the installer finds a file with the same name as the component's key file installed in the target location, it compares the version, date, and language of the two key files and uses file versioning rules to determine whether to install the component provided by the package. If the installer determines it needs to replace the component base upon the key file, then it uses the file versioning rules on each installed file to determine whether to replace the file.

Note that when authoring an installation package with versioned files, the version string in the Version column of the [File table](#) must always be identical to the version of the file included with the package.

The default file versioning rules can be overridden or modified by using the **REINSTALLMODE** property. The installer uses the file versioning rules specified by the **REINSTALLMODE** property when installing, reinstalling, or repairing a file. The following example shows how the installer applies the default [File Versioning Rules](#). The default value of the **REINSTALLMODE** property is "omus".

The following component key files are installed on the system before the component is reinstalled.

File	Version	Create date	Modified date	Language
FileA	1.0.0000	1/1/99	1/1/99	ENG
FileB	2.0.0000	1/1/99	1/1/99	ENG
FileC	1.0.0000	1/1/99	1/1/99	ENG
FileD	1.0.0000	1/1/99	1/2/99	ENG
FileE	none	1/1/99	1/1/99	none
FileF (modified > create)	none	1/1/99	1/2/99	none
FileG	1.0.0000	1/1/99	1/1/99	ENG

FileH	1.0.0000	1/1/99	1/1/99	ENG,FRN,SPN
FileI	1.0.0000	1/1/99	1/1/99	ENG,FRN
FileJ	1.0.0000	1/1/99	1/1/99	ENG,GER,ITN

The following component key files are included in the installer package.

<b>File</b>	<b>Version</b>	<b>Create date</b>	<b>Modified date</b>	<b>Language</b>
FileA (marked same)	1.0.0000	1/1/99	1/1/99	ENG
FileB (earlier version)	1.0.0000	1/1/99	1/1/99	ENG
FileC (later version)	2.0.0000	1/1/99	1/1/99	ENG
FileD (later version)	2.0.0000	12/31/98	1/10/99	FRN
FileE (marked same)	none	1/1/99	1/1/99	none
FileF (new file)	none	1/3/99	1/3/99	none
FileG (new language)	1.0.0000	1/1/99	1/1/99	FRN
FileH (new language)	1.0.0000	1/1/99	1/1/99	ITN,ENG,GER
FileI (more languages)	1.0.0000	1/1/99	1/1/99	ENG,FRN,SPN
FileJ (fewer languages)	1.0.0000	1/1/99	1/1/99	GER

The following component key files stay on the system after the component is reinstalled. The state of the key file determines the state of any other files in the component.

<b>File</b>	<b>Version</b>	<b>Create date</b>	<b>Modified date</b>	<b>Language</b>
FileA (original)	1.0.0000	1/1/99	1/1/99	ENG
FileB (original)	2.0.0000	1/1/99	1/1/99	ENG
FileC (replacement)	2.0.0000	1/1/99	1/1/99	ENG
FileD (replacement)	2.0.0000	12/31/98	1/10/99	FRN
FileE (replacement)	none	1/1/99	1/1/99	none
FileF (original)	none	1/1/99	1/2/99	none
FileG (replacement)	1.0.0000	1/1/99	1/1/99	FRN
FileH (replacement)	1.0.0000	1/1/99	1/1/99	ITN,ENG,GER
FileI (replacement)	1.0.0000	1/1/99	1/1/99	ENG,FRN,SPN
FileJ (original)	1.0.0000	1/1/99	1/1/99	ENG,GER,ITN

## See Also

[CRC Checking During an Installation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Editing Installer Shortcuts

The [Shortcut table](#) holds the information the application needs to create installer shortcuts on the user's computer. Installer shortcuts support [installation-on-demand](#) and [advertisement](#). Note that you cannot edit the properties of installer shortcuts as you can in the case of regular shortcuts.

Setting the **DISABLEADVTSHORTCUTS** property disables the generation of installer shortcuts and specifies that these shortcuts should instead be replaced by regular shortcuts.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installing Multiple Instances of Products and Patches

Windows Installer permits one instance of a product code to be installed per context, and the two possible types of context are the following:

- Machine
- User

If a product code remains unchanged, only one instance can be installed in the machine context and only one instance can be installed in each user context.

For multiple instances to remain isolated, they must have different product codes and cannot share file data or nonfile data. The Windows Installer cannot install multiple instances of products using [concurrent installations](#). However, you can install multiple instances of a product if you have a separate installation package for each instance of a product or patch. Then each package can keep its own set of data and have its own unique product code.

Starting with the installer running Windows Server 2003 and Windows XP with Service Pack 1 (SP1), you can install multiple instances of a product by using product code transforms and one .msi package or one patch. You can also use product code transforms to install multiple instances of a product with Windows 2000 with Service Pack 4 (SP4) and Windows Installer 3.0. The only way to install more than one instance of a product with previous versions of the installer is to have a separate installation package for each instance.

Using instance transforms significantly reduces the effort needed to support multiple instances of a product. You can author one base Windows Installer package for a product and then multiple instance transforms that change the product code and manage data for each instance.

For more information, see [Authoring Multiple Instances with Instance Transforms](#) and [Installing Multiple Instances with Instance Transforms](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring Multiple Instances with Instance Transforms

To install multiple instances of a product from one Windows Installer package, you need to author a base installation package for the product and an instance transform for each instance to be installed in addition to the base instance. Use the following guidelines when authoring your base package and transforms:

- Your setup application can check for the presence of the installer running on a version of Windows Vista, Windows Server 2003, Windows XP with Service Pack 1 (SP1), Windows 2000 with Service Pack 4 (SP4), and the Windows Installer 3.0 redistributable. Any of these installer versions (or later) are required to install multiple instances from a single package using a product code–changing transform.
- Each instance must have a unique product code and instance identifier. You may define a property in the base package, the value of which can be set to the instance identifier.
- To keep the files of each instance isolated, the base package should install files into a directory location that depends on the instance identifier.
- To keep the nonfile data of each instance isolated, the base package should collect nonfile data into sets of components for each instance. The appropriate components should then be installed based on conditional statements that depend on the instance identifier.
- Author an instance transform for each instance being installed in addition to the base instance. The base package may install its own instance.
- The instance transform must change the product code and identifier for each instance.

- It is recommended that the product transform also change the product name so that the instance is readily distinguished in Add/Remove Programs through Control Panel.
- If the instance transform installs files, they should be installed in a directory that depends on the instance identifier.
- All nonfile data, such as registry keys, should include the instance name in their path to prevent collisions. This can be accomplished by using the property whose value is the instance identifier in the path as shown by the following example of a [Registry Table](#).

Registry	Root	Key	Name	Value
Reg1	1	Software\Microsoft\MyProduct\ [InstanceId]	InstanceGuid	[ProductCode]

For more information, see [Installing Multiple Instances of Products and Patches](#) and [Installing Multiple Instances with Instance Transforms](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installing Multiple Instances with Instance Transforms

This topic provides guidelines for installing or reinstalling a multiple instance installation that uses instance transforms.

- When installing a new instance with an instance transform, include the **MSINEWINSTANCE** property and set **MSINEWINSTANCE=1**.
- When installing a new instance with an instance transform, include the TRANSFORMS property and set the first transform in the list of transforms to the instance transform that changes the product code. Any customization transforms should follow the instance transform at the beginning of this list.
- The easiest way to initiate a maintenance installation, and reinstall an instance, is to reference the product code of the instance. If you initiate the maintenance installation by using the package path, you must also specify the product code of the instance. From the command line, use the /n {Product Code} option. From code or script, use the **MSIINSTANCEGUID** property.

The following example shows installing a new instance from a command line where the instance transform is prefixed by a colon which specifies that the transform is embedded in the package.

```
msiexec /I mypackage.msi TRANSFORMS=:instance.mst  
MSINEWINSTANCE=1 /qb
```

The following example shows installing a new instance using **MsiInstallProduct**.

```
UINT uiStat = MsiInstallProduct(_T("path to mypackage.msi"),
```

The following example shows installing the new instance from script.

```
Dim Installer As Object  
Set Installer = CreateObject("WindowsInstaller.Installer")
```

```
Installer.InstallProduct "path to mypackage.msi", "TRANSFORM
```

The following example reinstalls an instance without re-caching the package. The instance is referred to by its product code {00000001-0002-0000-0000-624474736554}.

```
msiexec /I {00000001-0002-0000-0000-624474736554}  
REINSTALL=ALL REINSTALLMODE=omus /qb
```

The following example reinstalls an instance and re-caches the package from the command line. The instance is referred to by the package path. You are only required to include the /n {Product Code} option if the product is originally installed with an instance transform.

```
msiexec /I c:\newpath\mypackage.msi /n {00000001-0002-0000-0000-  
624474736554} REINSTALL=ALL REINSTALLMODE=vomus /qb
```

The following example reinstalls an instance and caches the package using [MsiInstallProduct](#). The instance is referred to by the package path. Use the [MSIINSTANCEGUID](#) property to provide the product code of the instance.

```
UINT uiStat = MsiInstallProduct(_T("path to mypackage.msi"),
```

The following example reinstalls an instance and caches the package using script. Use the [MSIINSTANCEGUID](#) property to provide the product code of the instance.

```
Dim Installer As Object  
Set Installer = CreateObject("WindowsInstaller.Installer")  
Installer.InstallProduct "path to mypackage.msi", "MSIINSTAN
```

The following example shows how to advertise an instance using the command line.

```
msiexec /jm mypackage.msi /t :instance.mst /c /qb
```

The following example shows how to install to advertise an instance using [MsiAdvertiseProductEx](#).

```
UINT uiStat = MsiAdvertiseProductEx(_T("path to mypackage.ms
```

The following example shows how to apply a patch to an instance from a

command line. You are only required to include the /n {Product Code} option if the product was originally installed with an instance transform.

```
msiexec /p mypatch.msp /n {00000001-0002-0000-0000-624474736554} /qb
```

The following example shows how to apply a patch to an instance installation using [MsiApplyPatch](#).

```
UINT uiStat = MsiApplyPatch(_T("path to mypatch.msp"), _T("{
```

For more information, see [Installing Multiple Instances of Products and Patches](#) and [Authoring Multiple Instances with Instance Transforms](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Windows Installer with a Terminal Server

The following may affect Windows Installer installations when using a terminal server. Setup developers should always test that their Windows Installer package installs as expected when users are also using a terminal server.

- On operating systems earlier than Windows Server 2008 and Windows Vista, the [EnableAdminTSRemote](#) system policy must be set to enable administrators to perform installations in the client session. Beginning with Windows Server 2008 and Windows Vista, the [EnableAdminTSRemote](#) policy no longer has any effect. Regardless of its setting, administrators and non-administrators can perform an installation in the client session or the console session. Administrators and non-administrators can always perform Windows Installer installations in the console session.
- The Windows Installer prevents installation in the per-user [installation context](#) if the [DisableUserInstallssystem policy](#) is set to 1. In this case, the installer ignores all applications registered as per-user and searches only for applications registered as per-machine.
- When an administrator performs an installation in a client session of a terminal server that is hosted in Windows 2000, the installation must use a UNC path and not a mapped drive letter.

Developers should adhere to the following guidelines when developing a Windows Installer component that may be used with a terminal server.

- Write all **HKCU** registry information in the **HKCU\Software** part of the registry.
- Storing configuration information in INI files is not recommended.
- Write per-user information to the registry when the application is run

for the first time and not at installation time. If you must write per-user information to the registry at installation time, separate the per-user and per-machine information into different Windows Installer components. Author the package such that the installer does not attempt to validate and repair the components containing per-user information when the application is installed.

- A package used for only per-machine installations should write environment variables to the computer's environment by including \* in the Name column of the [Environment Table](#). If the package can be used for per-user installations or per-machine installations, use two components. Include the per-user component in the [Component Table](#) and enter the user settings in the Environment Table. Include the per-machine component in the Component Table and enter the computer settings in the Environment Table. Control which component gets installed by using conditional statements based upon the [ALLUSERS](#) property in the Condition field of the Component Table.
- When performing per-machine installations from a terminal server, the installer writes per-user environment variables to **HKCU\Default\Environment**. Because the terminal server does not replicate this section of the registry, the installation does not set the per-user environment variables.
- Because a server may be configured to prevent users from repairing applications, your application should handle the case of missing registry keys gracefully.

The following applies when a Windows Installer package that uses DLL, EXE, or Script [custom actions](#) is installed in the per-machine installation context on a terminal server. In this case, the installer sets the [TerminalServer](#) property.

- Deferred custom actions run in the context of the local system unless the action has the **msidbCustomActionTypeTSAware** attribute.

This is true even if the custom action impersonates the user on a system that is not a terminal server. Note that if a custom action having the **msidbCustomActionTypeTSAware** attribute changes the user's registry, the installer does not automatically ensure that those changes are also made in the registry of every user on the computer.

- Any registry operations in a deferred custom action that read from the **HKCU** registry hive sees the system's default registry hive and not the current user's registry hive.
- Any registry operations in a deferred custom action that write to **HKCU\Software** are detected by the installer and copied to every user of the computer at the next logon of the user.
- Any registry operations in a deferred custom action that write to **HKCU**, but are not under the **HKCU\Software** registry key, are not detected by the installer or copied.

For more information, see Terminal Services in the Microsoft Windows Software Development Kit (SDK).

## See Also

[EnableAdminTSRemote  
TerminalServer property  
RemoteAdminTS property](#)  
Terminal Services

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Controlling Feature Selection States

You can control which feature installation options are available for users and applications to select by authoring the [Feature table](#) and [Component table](#).

- To prevent selection of the advertise state for a feature, include `msidbFeatureAttributesDisallowAdvertise` in the feature's `Attributes` field in the [Feature table](#).
- To prevent selection of the run-from-source or run-from-network states for a feature, include `msidbComponentAttributesLocalOnly` in the `Attributes` fields in the [Component table](#) for every component belonging to the feature. If a feature has no components, the feature always has the run-from-source and run-from-my-computer options available.
- To prevent selection of the run-from-my-computer state for a feature, include `msidbComponentAttributesSourceOnly` in the `Attributes` fields in the [Component table](#) for every component belonging to the feature. If a feature has no components, the feature always has the run-from-source and run-from-my-computer options available.
- New child features can be authored by including `msidbFeatureAttributesFollowParent` and `msidbFeatureAttributesUIDisallowAbsent` in the `Attributes` field of the [Feature table](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Guidelines for Authoring Secure Installations

Adherence to the following guidelines when authoring a Windows Installer package helps maintain a secure environment during installation:

- Administrators should install managed applications into a target installation folder for which non-admin users do not have change or modify privileges.
- Make any property set by the user a public property. Private properties cannot be changed by the user interacting with the user interface. For information, see [About Properties](#).
- Do not use properties for passwords or other information that must remain secure. The installer may write the value of a property authored into the [Property table](#), or created at run time, into a log or the system registry. For additional information, see [Preventing Confidential Information from Being Written into the Log File](#).
- When the installation requires the installer to use *elevated* privileges, use [Restricted Public Properties](#) to restrict the public properties a user can change. Some restrictions are commonly necessary to maintain a secure environment when the installation requires the installer to use elevated privileges.
- Avoid installing services that impersonate the privileges of a particular user because this may write security data into a log or the system registry. This creates potential for a security problem, password conflict, or the loss of configuration data when the system is restarted. For details, see [ServiceInstall table](#).
- Use the [LockPermissions table](#) and [MsiLockPermissionsEx table](#) to secure services, files, registry keys, and created folders in a locked-down environment.
- Add a digital signatures to the installation to ensure the integrity of

the files. For details, see [Digital Signatures and Windows Installer](#) and [Authoring a Fully Verified Signed Installation](#).

- Author your Windows Installer package such that if the user is denied access to resources, the setup fails in a manner that maintains a secure environment. Check the user's access privileges and determine whether there is sufficient disk space before installation begins. Commonly, the installer should only display a browse dialog box if the current user is an administrator or if the installation does not require elevated privileges. For details, see [Source Resiliency](#).
- Use [Secured Transforms](#) to store transforms in a secure file system locally on the user's computer. This prevents the user from having write access to the transform.
- For information on how to secure media sources of managed applications, see [Source Resiliency](#).
- Use the [Security Summary](#) Property to indicate whether the package should be opened as read-only. This property should be set to read-only recommended for an installation database and to read-only enforced for a transform or patch.
- The installer runs custom actions with user privileges by default in order to limit the access of custom actions to the system. The installer may run custom actions with elevated privileges if a managed application is being installed or if the system policy has been specified for elevated privileges. For details, see [Custom Action Security](#).
- Use the [DisablePatch](#) policy to provide security in environments where patching must be restricted.
- Use the [AppId table](#) to register common security and configuration settings for DCOM objects.
- For related information, see [Guidelines for Securing Custom Actions](#).
- For related information, see [Guidelines for Securing Packages on](#)

## [Locked-Down Computers.](#)

- Starting with Windows Installer 3.0, [User Account Control \(UAC\) Patching](#) enables non-administrator users to patch applications installed in the per-machine context. UAC patching is enabled by providing a signer certificate in the [MsiPatchCertificate](#) table and signing patches with the same certificate.
- The capability of the Windows Installer 5.0 to set access permissions on services, files, created folders, and registry entries can help make installation applications more secure. For information, see [Securing Resources](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Guidelines for Securing Custom Actions

Adherence to the following guidelines when authoring a Windows Installer package with custom actions helps maintain a secure environment during installation:

- Secure any additional files written by your custom action.
- Check buffer lengths and validity of all data read by your custom action. This includes properties that may supply data to your custom action, particularly those that use public properties provided by a user.
- Do not rely on external DLLs that are not trusted by the system on all platforms on which your installation package is intended to run.
- Carefully consider whether to use custom actions that use *elevated* privileges or impersonation. If your custom action must run with elevated privileges, be sure that the custom action code guards against buffer overruns and inadvertent loading of unsafe code. Note that during the execution phase of the installation, the installer passes information to a process with elevated privileges and runs the script. Any custom actions that run during the execution phase may run with elevated privileges.
- Gather all information provided by the user during the UI sequence. Do not prompt the user for any information that can't be set using a public property.
- If your script custom action expands properties, take precautions that the custom action is secured against the possibility of script injection. Script may be logged in clear text.

See also [Custom Action Security](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Guidelines for Securing Packages on Locked-Down Computers

Adherence to the following guidelines when authoring a Windows Installer package that will be used on locked-down computers helps maintain a secure environment during installation:

- Test your package for compatibility with the Windows Installer machine [System Policy](#).
- Make sure your package runs with all [user interface levels](#), none, basic, limited, and full.
- Test your package on NTFS partitions, both with [elevated](#) and non-elevated privileges.
- Starting with Windows Installer 3.0, [User Account Control \(UAC\) Patching](#) enables non-administrator users to patch applications installed in the per-machine context. Test your patch package on Windows Vista and Windows XP for both installation by users with administrator access and by non-administrator users.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Securing Resources

The capability of the Windows Installer to set access permissions on services, files, created folders, and registry entries can help make installation applications more secure. The use of the [MsiLockPermissionsEx](#) or [LockPermissions](#) tables to secure resources is one of the recommended [Guidelines for Authoring Secure Installations](#). The MsiLockPermissionsEx table can enable a package author to secure a resource without having to write a [custom action](#).

Beginning with packages developed for Windows Installer 5.0, the MsiLockPermissionsEx table should replace the use of the LockPermissions table. The extended functionality provided by the MsiLockPermissionsEx table enables a package to secure Windows Services, files, folders, and registry keys. A package should not contain both the MsiLockPermissionsEx and the LockPermissions tables.

Windows Installer 4.5 and earlier ignores the MsiLockPermissionsEx table. Beginning with Windows Installer 5.0, the installation fails with an error message 1941 if the package contains both a LockPermissions table and MsiLockPermissionsEx table. Existing installation packages that contain only the LockPermissions table can be still be installed using Windows Installer 5.0.

Windows Installer 5.0 processes the information in the MsiLockPermissionsEx table when it runs the [InstallFiles](#), [InstallServices](#), [WriteRegistryValues](#) and [CreateFolders](#) standard actions. A securable object must be installed or reinstalled to be secured and it is not possible to append an Access Control List (ACL) to an existing object without reinstalling that object.

To specify the service, file, directory, or registry key that is to be secured, enter the identifying information in the LockObject and Table fields of the MsiLockPermissionsEx table. An object is identified by it's primary key in the [ServiceInstall Table](#), [File Table](#), [Registry Table](#), or [CreateFolder Table](#).

To request that specified permissions apply to an object, enter a valid security descriptor string in the SDDLText field of the MsiLockPermissionsEx table using valid security descriptor definition language (SDDL.) The MsiLockPermissionsEx table can specify a security descriptor that denies permissions, specifies inheritance of

permissions from a parent resource, or specifies the permissions of a new account. For a list of all the permissions that can be granted, denied, or inherited see ACE Strings. Windows Installer 5.0 extends the set of available security identifiers (SIDs.) For a list of the valid SIDs, see SID Strings.

Beginning with Windows Installer 5.0, the [FormattedSDDLText](#) data type extends the [Formatted](#) data type. The Windows Installer validates that the FormattedSDDLText string entered in the SDDLText column of the MsiLockPermissionsEx table conforms to the Security Descriptor String Format.

**Windows Installer 4.5 or earlier:** Not supported. The MsiLockPermissionsEx table and FormattedSDDLText data type are available beginning with Windows Installer 5.0.

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Isolated Components

By using [isolated components](#), authors of installation packages can specify that the installer copy the shared files of an application directly into the application's folder rather than to a shared location. This private set of files are then used exclusively by the application.

For more information, see [Installing a COM component to a private location](#), [Installing a non-COM component to a private location](#), [Make a COM component in an existing package private](#) , or [Make a non-COM component in an existing package private](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installing a COM Component to a Private Location

To force a COM-client application to always use the same copy of a COM-server, author the application's installation package to specify an [isolated components](#) relationship between the COM server and client. This installs a private copy of the COM-server component to a location used exclusively by the client application. Do the following when authoring the package:

- Put the COM server DLL and the .exe client in separate components.
- Enter a record in the [IsolatedComponent table](#) with the COM-client component in the Component\_Shared column and the client application in the Component\_Application column. Include the [IsolateComponents action](#) in the sequence tables.
- Set the msidbComponentAttributesSharedDllRefCount bit in the [Component table](#) record for Component\_Shared. The installer requires this global refcount on the shared location to protect the shared files and registration in cases where there is sharing with other installation technologies.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installing a non-COM Component to a Private Location

To force a client application to always use the same copy of a non-COM server, author the application's installation package to specify an [isolated components](#) relationship between the server and client. This installs a private copy of the server component to a location used exclusively by the client application. Do the following when authoring the package:

- Put the server DLL and the .exe client in separate components.
- Enter a record in the [IsolatedComponent table](#) with the client component in the Component\_Shared column and the client application in the Component\_Application column. Include the [IsolateComponents action](#) in the sequence tables.
- Set the msidbComponentAttributesSharedDllRefCount bit in the [Component table](#) record for Component\_Shared. The installer requires this global refcount on the shared location to protect the shared files and registration in cases where there is sharing with other installation technologies.
- Avoid authoring a shared registered path across components.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Make a COM Component in an Existing Package Private

An administrator can force a COM-client application to always use the same copy of a COM-server in an existing package—without affecting other applications—by specifying an [isolated components](#) relationship between the COM server and client. This installs a private copy of the COM-server component to a location used exclusively by the client application. The administrator needs to use transforms or a package authoring tool to do the following:

- Put the COM server DLL and the .exe client in separate components.
- Enter a record in the [IsolatedComponent table](#) with the COM-client component in the Component\_Shared column and the client application in the Component\_Application column. Include the [IsolateComponents action](#) in the sequence tables.
- Set the **msidbComponentAttributesSharedDllRefCount** bit in the [Component table](#) record for Component\_Shared. The installer requires this global refcount on the shared location to protect the shared files and registration in cases where there is sharing with other installation technologies.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Make a non-COM Component in an Existing Package Private

An administrator can force a client application to always use the same copy of a non-COM server in an existing package—without affecting other applications—by specifying an [isolated components](#) relationship between the server and client. This installs a private copy of the server component to a location used exclusively by the client application. The administrator needs to use transforms or a package authoring tool to do the following:

- Put the server DLL and the .exe client in separate components.
- Enter a record in the [IsolatedComponent table](#) with the client component in the Component\_Shared column and the client application in the Component\_Application column. Include the [IsolateComponents action](#) in the sequence tables.
- Set the **msidbComponentAttributesSharedDllRefCount** bit in the [Component table](#) record for Component\_Shared. The installer requires this global refcount on the shared location to protect the shared files and registration in cases where there is sharing with other installation technologies.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Determining the Windows Installer Version

You can use the following methods to determine the Windows Installer version:

- Call the **MsiGetFileVersion** function with the *szFilePath* parameter set to the path to the file Msi.dll.  
You can call the **SHGetKnownFolderPath** function with the CSIDL\_SYSTEM constant to get the path to Msi.dll. Beginning with Windows Vista, applications should use the **SHGetFolderPath** function and the **REFKNOWNFOLDERID "System."** Existing applications that use the **SHGetFolderPath** function and the **CSIDL** type will continue to work.
- The value of the **Installer.Version** property of the **Installer Object** is equivalent to the four-field strings listed in the [Released Versions of Windows Installer](#) topic.
- Applications can get the Windows Installer version by using **DllGetVersion**.
- The installer sets the **VersionMsi** property to the version of Windows Installer that is run during the installation.

For more information, see [Released Versions of Windows Installer](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installing a COM+ Application with the Windows Installer

The following actions and table for installing COM+ applications are available in Windows Installer:

- [Complus Table](#)
- [RegisterComPlus Action](#)
- [UnregisterComPlus Action](#)

For information about how to install COM+ applications, see [Deploying COM+ Applications](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Adding or Removing Registry Keys on the Installation or Removal of Components

The installer can add or remove registry values after all selected components and their related files are installed. For more information, see the following:

[Modifying the Registry](#)

[Registry Table](#)

[RemoveRegistry Table](#)

[Adding and Removing an Application and Leaving No Trace in the Registry](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Adding and Removing an Application and Leaving No Trace in the Registry

If an application must be registered, author the installation package as described in the section [Adding and Removing Registry Keys on the Installation or Removal of Components](#). Registration is used by the installer for advertisement and by the Add or Remove Programs feature in Control Panel. If an application is not registered, it cannot be advertised, and is not listed in the Add or Remove Programs feature in Control Panel.

You can omit registering an application by removing the [RegisterProduct Action](#), [RegisterUser Action](#), [PublishProduct Action](#), and [PublishFeatures Action](#) from the [InstallExecuteSequence Table](#) and [AdvtExecuteSequence Table](#). All of these actions must be removed, or some trace of the application may remain in the registry. Removing all of these actions prevents the application from being listed in the Add or Remove Programs feature in Control Panel, and prevents the advertisement of the application. Removing all of these actions also prevents the application from being registered with the Windows Installer configuration data. This means that you cannot remove, repair, or reinstall the application by using the Windows Installer [Command-Line Options](#), or the Windows Installer application programming interface (API).

To hide an application from the Add or Remove Programs feature in Control Panel and still be able to use the Windows Installer to manage an application, leave the registration actions in the sequence tables, and set the [ARPSYSTEMCOMPONENT Property](#) in the [Property Table](#) to 1 (one). The application does not appear in the Add or Remove Programs feature, but you can use the Windows Installer to install-on-demand, uninstall, repair, and reinstall the application.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Reducing the Size of an .msi File

Developers of Windows Installer packages may notice their .msi files getting larger than expected after repeated edits and saves. Windows Installer .msi files are compound files that contain storages and streams, and have some of the same storage limitations as OLE document files. If you edit and save the same .msi file over and over, it creates wasted storage space in the file. This only affects authors of .msi files and has no effect on Windows Installer users. Developers may want to remove this wasted storage space before shipping their final .msi file.

To remove wasted storage space and reduce the final size of .msi files, you have the following options.

- Export all of the tables in the database to .idt files, and then import those into a new database. This produces the most compact storage possible.
- Use a software utility for compacting the storage space of OLE document files.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Changing the Target Location for a Directory

If possible, the best way to specify the target location for a directory is to author the [Directory Table](#) in your installation package to provide the correct location. For more information, see [Using the Directory Table](#).

If you need to change the directory location at the time of the installation, you have the following options:

- Specify the location of a directory by setting the value of a [Public Property](#) on the command line. During the [CostFinalize Action](#), the internal directory paths used by the installer are updated to the value of properties listed as keys in the [Directory Table](#). For more information, see [Using Properties](#) and [Setting Public Property Values on the Command Line](#).
- Specify the location of a directory by using a custom action. If the custom action is to run before the [CostFinalize Action](#), you can use a [Custom Action Type 51](#) to set the value of a property from a formatted text string. If the custom action runs after the [CostFinalize Action](#), you can use a [Custom Action Type 35](#) to set the value of the directory path from a formatted text string. Custom actions that change one of the [System Folder Properties](#) should be included in both the execution sequence tables ([InstallExecuteSequence Table](#) or [AdminExecuteSequence Table](#)), and the user interface sequence tables ([InstallUISequence Table](#) and [AdminUISequence Table](#)) so that the folder is changed during both *full UI* and *basic UI* installations.
- If the installation is running a full UI, you can use [MsiSetTargetPath](#) or the [SetTargetPath ControlEvent](#) to set the directory path. Check the [ProductState](#) Property to determine whether the product that contains this component is already installed before calling

**MsiSetTargetPath** or the SetTargetPath ControlEvent. Do not attempt to change the target directory path if some components that use that path are already installed for the current user or a different user.

The following restrictions apply to all of the above options:

- Do not attempt to change the target directory path if some components that use the path are already installed for the current user or for a different user.
- Do not attempt to change the target directory path during a [Maintenance Installation](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Hiding the Cancel Button During an Installation

You can hide the **Cancel** button that is used to cancel an installation by using a command-line option, the Windows Installer API, or a custom action. The **Cancel** button can be hidden for part or all of the installation depending on which method you use.

## Hiding the Cancel Button from the Command Line

The **Cancel** button can be hidden during installation by using the (!) command-line option. This can only be done for a basic user interface level installation (/qb). The **Cancel** button is hidden for the entire installation. For more information, see [Command-Line Options](#) and [User Interface Levels](#). The following command line hides the **Cancel** button and installs Example.msi.

```
msiexec /I example.msi /qb!
```

## Hiding the Cancel Button from an Application or Script

You can write an application or script to hide the **Cancel** button. This can only be done for a basic UI level installation so that the **Cancel** button is hidden for the entire installation.

To hide the Cancel button from an application, set `INSTALLUILEVEL_HIDECANCEL` when calling [MsiSetInternalUI](#). The following example hides the **Cancel** button and installs Example.msi.

```
#include <windows.h>
#include <stdio.h>
#include <Shellapi.h>
#include <msi.h>
#include <Msiquery.h>
#include <tchar.h>
#pragma comment(lib, "msi.lib")

int main()
```

```
{  
  
INSTALLUILEVEL uiPrevLevel = MsiSetInternalUI( INSTALLUILEVEL  
UINT uiStat = MsiInstallProduct(_T("example.msi"), NULL);  
  
return 0;  
}
```

To hide the **Cancel** button from script, add `msiUILevelHideCancel` to the **UILevel** property of the **Installer Object**. The following VBScript sample hides the **Cancel** button and installs Example.msi.

```
Dim Installer As Object  
Set Installer = CreateObject("WindowsInstaller.Installer")  
Installer.UILevel = msiUILevelBasic + msiUILevelHideCancel  
Installer.InstallProduct "example.msi"
```

## Hiding the Cancel Button for Parts of an Installation Using a Custom Action

Your installation can hide and unhide the **Cancel** button during parts of an installation by sending an `INSTALLMESSAGE_COMMONDATA` message using a DLL custom action or scripts. For more information, see [Dynamic-Link Libraries](#), [Scripts](#), [Custom Actions](#), and [Sending Messages to Windows Installer Using MsiProcessMessage](#).

A call to a custom action must provide a record. Field 1 of this record must contain the value 2 (two) to specify the **Cancel** button. Field 2 must contain either the value 0 or 1. A value of 0 in Field 2 hides the button and a value of 1 in Field 2 unhides the button. Note that allocating a record of size 2 with `MsiCreateRecord` provides fields 0, 1, and 2.

The following sample DLL custom action hides the **Cancel** button.

```
#include <windows.h>  
#include <stdio.h>  
#include <Shellapi.h>
```

```

#include <msi.h>
#include <Msiquery.h>

UINT __stdcall HideCancelButton(MSIHANDLE hInstall)
{
    PMSIHANDLE hRecord = MsiCreateRecord(2);
    if ( !hRecord)
        return ERROR_INSTALL_FAILURE;

    if (ERROR_SUCCESS != MsiRecordSetInteger(hRecord, 1, 2)
        || ERROR_SUCCESS != MsiRecordSetInteger(hRecord, 2, 0))
        return ERROR_INSTALL_FAILURE;

    MsiProcessMessage(hInstall, INSTALLMESSAGE_COMMONDATA);

    return ERROR_SUCCESS;
}

```

The following VBScript Custom Action hides the **Cancel** button.

```

Function HideCancelButton()

    Dim Record
    Set Record = Installer.CreateRecord(2)

    Record.IntegerData(1) = 2
    Record.IntegerData(2) = 0

    Session.Message msiMessageTypeCommonData, Record

    ' return success
    HideCancelButton = 1
    Exit Function

End Function

```

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Windows Installer with UAC

Windows Installer complies with *User Account Control* (UAC) in Windows Vista. With authorization from an administrator, the Windows Installer can install applications or patches on behalf of a user that may not be a member of the Administrators group. This is referred to as an *elevated* installation because the Windows Installer makes changes to the system on behalf of the user that would not normally be allowed if the user were making the changes directly.

- When using Windows Vista in a corporate environment, applications can be designated as managed applications. Using application deployment and Group Policy, administrators can lockdown directories and then assign or publish the managed applications in those directories to *standard users* for install, repair, or removal. Managed applications are registered in the **HKEY\_LOCAL\_MACHINE** registry hive. Once an application has been registered as a managed application, subsequent installation operations always run with elevated privileges. If the user is running as an administrator, no prompting is required to continue the installation. If the user is running as a standard user, and the application has already been assigned or published, the installation of the managed application can continue without prompting.
- When using Windows Vista in a non-corporate environment, UAC handles the elevation of application installation. Windows Installer 4.0 can call to the *Application Information Service* (AIS) to request administrator authorization to elevate an installation. Before an installation identified as requiring administrator privileges can be run, UAC prompts the user for consent to elevate the installation. The consent prompt is displayed by default, even if the user is a member of the local Administrators group, because administrators run as standard users until an application or system component that requires administrative credential requests permission to run. This

user experience is called [Admin Approval Mode](#) (AAM). If a standard user attempts to install the application, the user has to get a person with administrator privilege to provide them their administrator credentials to continue the installation. This user experience is called an [Over the Shoulder](#) (OTS) credential prompt.

- Because UAC restricts privileges during the stages of an installation, developers of Windows Installer packages should not assume that their installation will always have access to all parts of the system. Windows Installer package developers should therefore adhere to the package guidelines described in [Guidelines for Packages](#) to ensure their package works with UAC and Windows Vista. A package that has been authored and tested to comply with UAC should contain the **MSIDEPLOYMENTCOMPLIANT** property set to 1.
- An administrator can also use the methods described in the section: [Installing a Package with Elevated Privileges for a Non-Admin](#) to enable a non-administrator user to install an application with elevated system privileges.
- Privileges are required to install an application in the per-user-managed context, and therefore subsequent Windows Installer reinstallations or repairs of the application are also performed by the installer using elevated privileges. This means that only patches from trusted sources can be applied to an application in the per-user-managed state. Beginning with Windows Installer 3.0, you can apply a patch to a per-user managed application after the patch has been registered as having elevated privileges. For information see [Patching Per-User Managed Applications](#).

**Note** When elevated privileges are not required to install a Windows Installer package, the author of the package can suppress the dialog box that UAC displays to prompt users for administrator authorization. For more information, see [Authoring Packages without the UAC Dialog Box](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Guidelines for Packages

Because *User Account Control* (UAC) in Windows Vista restricts privileges during an installation, developers of Windows Installer packages should not assume that their installation always has access to all parts of the system.

An installer package that can be successfully deployed to standard users via Group Policy should in most cases also work with UAC in Windows Vista. Exceptions to this can occur if the [InstallUISequence table](#) contains the [LaunchConditions action](#) or the [LaunchCondition table](#) contains a condition based on the [Privileged property](#). Windows Installer package developers should therefore adhere to the following guidelines to ensure their package works with UAC and Windows Vista.

- When including an [installation context](#) condition with an action in the [InstallUISequence table](#), use a conditional statement based on the [Privileged property](#). Do not use a condition based on the [AdminUser property](#).
- When including the installation context with the installation launch conditions, use a [Custom Action Type 19](#) in the [InstallExecuteSequence table](#) and make the custom action conditional upon the [Privileged property](#). Do not use an action in the [LaunchCondition table](#) with a condition based on the [AdminUser property](#) or [Privileged property](#).
- To read or modify the system configuration, use a [deferred execution custom action](#) in the [InstallExecuteSequence table](#). Do not use immediate execution custom actions in the [InstallUISequence table](#) to modify the system configuration.
- To modify parts of the system that are not user specific, use a deferred custom action in the [InstallExecuteSequence table](#). You should include the [msidbCustomActionTypeNoImpersonate](#) bit in the custom action type.
- Omit Bit 3 from the value of the [Word Count](#) Summary Property to

indicate that the package can be required to be elevated. Do not include this bit unless elevated privileges are not required to install this package.

- Include a manifest with the application's Requested Execution Level.
- Include a certificate in the [MsiPatchCertificate](#) table of original package and sign all patches with the same certificate.
- If elevated privileges are required to install a Windows Installer package, the author of the package should include the [ElevationShield](#) attribute for the [PushButton control](#) used to start the installation. This will alert the user that clicking on the button will display the UAC dialog box requesting administrator authorization to continue the installation.
- Set the **MSIDEPLOYMENTCOMPLIANT** property to 1 to indicate to the Windows Installer that the package has been authored and tested to comply with UAC in Windows Vista. If this property is not set, the installer determines whether the package complies with UAC.

Outside of Group Policy, the following check for UAC compliance can be used on Windows XP.

#### ▶ **To check for UAC compliance outside of Group Policy**

1. Log on to the computer as an administrator.
2. Advertise the package for a per-machine installation:  
**msiexec /jm package.msi**
3. Log off the computer.
4. Log on to the computer as a standard user.
5. Attempt to install the advertised package:  
**msiexec /i package.msi**
6. In most cases, if the installation is successful, the package is UAC compliant.

7. Set the **MSIDEPLOYMENTCOMPLIANT** property in the package to 1.
8. Test for correct installation of the package using Windows Vista.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring Packages without the UAC Dialog Box

When elevated privileges are not required to install a Windows Installer package, the author of the package can suppress the dialog box that *User Account Control* (UAC) displays to prompt users for administrator authorization.

To suppress the display of the UAC dialog box when installing the application, the author of the package should do the following:

- Install the application using Windows Installer 4.0 or later on Windows Vista.
- Do not depend on using elevated system privileges to install the application on the computer.
- Install the application in the per-user context and make this the default installation context of the package. If the **ALLUSERS** property is not set, the installer installs the package in the per-user context. If you do not include the **ALLUSERS** property in the [Property table](#), the installer does not set this property and so per-user installation becomes the default installation context. You can override this default by setting the **ALLUSERS** property on the command line.
- Set Bit 3 in the [Word Count Summary](#) property to indicate that elevated privileges are not required to install the application.

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installing a Package with Elevated Privileges for a Non-Admin

An administrator can use the following methods to enable a non-administrator user to install an application with elevated system privileges.

- In Windows Vista with Windows Installer, a member of the Administrators group can provide authorization for a non-administrator to elevate the installation through [User Account Control](#) (UAC) as described in [Using Windows Installer with UAC](#).

**Windows Vista:** Required.

The following methods can also be used to install an application with elevated system privileges.

- An administrator can advertise an application on a user's computer by assigning or publishing the Windows Installer package using application deployment and Group Policy. The administrator advertises the package for per-machine installation. If a non-administrator user then installs the application, the installation can run with elevated privileges. Non-administrator users cannot install unadvertised packages that require elevated system privileges.
- An administrator can go to the user's computer and [advertise](#) the application for per-machine installation. Because the Windows Installer always has elevated privileges while doing installs in the per-machine [installation context](#), if a non-administrator user then installs the advertised application, the installation can run with elevated privileges. Non-administrator users still cannot install unadvertised packages that require elevated privileges.
- A non-privileged user can install an advertised application that requires elevated privileges if a local system agent advertises the

application. The application can be advertised for a per-user or per-machine installation. An application installed using this method is considered managed. For more information, see [Advertising a Per-User Application To Be Installed with Elevated Privileges](#).

- An administrator can set the [AlwaysInstallElevated](#) policy for both per-user and per-machine installations. This method can open a computer to a security risk, because when this policy is set, a non-administrator user can run installations with elevated privileges and access secure locations on the computer, such as the SystemFolder or the **HKLM** registry key.

If the application is installed per-machine while the [AlwaysInstallElevated](#) policy is set, the product is treated as managed. In this case, the application can still perform a repair with elevated privileges if the policy is removed. Also, if the application is installed per-user while the [AlwaysInstallElevated](#) policy is set, the application is unable to perform a repair if the policy is removed.

- An administrator can go to a user's computer and do a per-machine installation of the application. Because privileges are required to perform this type of installation, per-machine installations are always managed.

## See Also

[Installation Context](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Advertising a Per-User Application To Be Installed with Elevated Privileges

To advertise an application on a per-user installation basis when the application requires elevated (that is, system) privileges for installation, use the guidelines in the following list:

- Your process must be a service that runs under the LocalSystem system account on Windows 2000 or later.
- Generate an advertise script by calling [MsiAdvertiseProduct](#) or [MsiAdvertiseProductEx](#).
- Your process must impersonate the user that is the target for the advertisement.
- Call [MsiAdvertiseScript](#), and use the flags  
SCRIPTFLAGS\_CACHEINFO |  
SCRIPTFLAGS\_REGDATA\_APPINFO |  
SCRIPTFLAGS\_REGDATA\_CNFGINFO |  
SCRIPTFLAGS\_SHORTCUTS.

When you follow the guidelines, you advertise an application to a specified user, and when the user chooses to install, the application is installed with elevated privileges.

## See Also

[Patching Per-User Managed Applications](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Windows Installer with Restart Manager

Applications that use Windows Installer 4.0 for installation and servicing on Windows Vista automatically use the Restart Manager to reduce system restarts. The default behavior on Windows Vista is to shut down applications rather than shut down and restart the operating system whenever possible. In cases where a system restart is unavoidable, installers can use the Restart Manager API to schedule restarts in such a way that it minimizes the disruption of the user's work flow.

Windows Installer developers can perform the following actions to prepare their package to work with the Restart Manager.

- Add the [MsiRMFilesInUse](#) dialog box to your package. If the MsiRMFilesInUse dialog box is present in the package, the Windows Vista user running an installation at the Full UI [user interface level](#) is given the option to automatically close and restart applications. An installation package can contain information for both the MsiRMFilesInUse dialog box and the [FilesInUse](#) dialog box . The MsiRMFilesInUse dialog box is only displayed if the package is installed with at least Windows Installer 4.0 on Windows Vista, and is otherwise ignored. Existing packages that do not have the MsiRMFilesInUse dialog box continue to function using the FilesInUse dialog box. A customization transform can be used to add an MsiRMFilesInUse dialog box to existing packages. End-users typically run installations at the Full UI user interface level. Basic UI or Reduced UI level installations give the user the option of using the Restart Manager to reduce system restarts even if the MsiRMFilesInUse dialog box is not present. Silent UI level installations always shut down applications and services, and on Windows Vista, always use Restart Manager.
- Register applications for a restart using the

**RegisterApplicationRestart** function. Restart Manager can only restart applications that have been registered for restart. Restart Manager restarts registered applications after the installation. If the installation requires a system restart, Restart Manager restarts the registered application after the system restart.

- Specify `INSTALLLOGMODE_RMFILESINUSE` when enabling an external user-interface handler with the **MsiSetExternalUI** and **MsiSetExternalUIRecord** functions. Windows Installer will send a `INSTALLMESSAGE_RMFILESINUSE` message for external user-interface handlers that support the Restart Manager. If no registered or internal user-interface handles the `INSTALLMESSAGE_RMFILESINUSE` message, the installer sends a `INSTALLMESSAGE_FILESINUSE` message for user-interface handlers that support the FilesInUse dialog box. For more information, see [Using Restart Manager with an External UI](#).
- Custom actions can add resources belonging to a Restart Manager session. The custom action should be sequenced before the **InstallValidate** action. Custom actions can use the **MsiRestartManagerSessionKey** property to obtain the session key, and should call the **RmJoinSession** and **RmEndSession** functions of the Restart Manager API. Custom actions cannot remove resources belonging to a Restart Manager session. Custom actions should not attempt to shutdown or restart applications using the **RmShutdown**, **RmGetList** and **RmRestart** functions.
- Package authors can base a condition in the **LaunchCondition** table on the **MsiSystemRebootPending** property to prevent the installation of their package when a system restart is pending.
- Package authors and administrators can control the interaction of the Windows Installer and Restart Manager by using the **MSIRESTARTMANAGERCONTROL**, **MSIDISABLERMRESTART**, **MSIRMSHUTDOWN** properties and the

[DisableAutomaticApplicationShutdown](#) policy.

- Applications and services should follow the guidelines described in the Using Restart Manager section of the Restart Manager documentation.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Restart Manager with an External UI

Windows Installer developers can prepare their installation package to work with Restart Manager by following the guidelines described in [Using Windows Installer with Restart Manager](#).

Specify the `INSTALLLOGMODE_RMFILESINUSE` message type when calling the [MsiSetExternalUI](#) or [MsiSetExternalUIRecord](#) function to enable the external user-interface handler. Windows Installer then sends an `INSTALLMESSAGE_RMFILESINUSE` message for use by external user-interface handlers that support the Restart Manager.

Your external user-interface handler should handle the information contained in `INSTALLMESSAGE_RMFILESINUSE` messages. If no registered or internal user-interface handles the `INSTALLMESSAGE_RMFILESINUSE` message, Windows Installer sends an `INSTALLMESSAGE_FILESINUSE` message for use by existing external handlers that support `INSTALLMESSAGE_FILESINUSE` messages and the [FilesInUse](#) dialog box.

The external user interface can return the values listed in the following table.

<b>External UI return value</b>	<b>Action taken by Windows Installer</b>
IDOK	The <b>OK</b> button was pressed by the user. The Windows Installer will request that the Restart Manager shut down and restart the applications that hold files currently in use.
IDCANCEL	The <b>CANCEL</b> button was pressed. Cancel the installation.
IDIGNORE	The <b>IGNORE</b> button was pressed. Ignore and continue the installation. A restart will be required at the end of the installation.
IDNO	The <b>NO</b> button was pressed. If the package has an <a href="#">MsiRMFilesInUse</a> dialog box, send a 1610 message. For more information, see <a href="#">Windows Installer Error Messages</a> . If the

	package does not have a MsiRMFilesInUse dialog box, send an INSTALLMESSAGE_FILESinUSE message.
IDRETRY	The <b>RETRY</b> button was pressed. Send the INSTALLMESSAGE_FILESinUSE message.
-1	An error. End the installation.

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Windows Installer with WMI

The Windows Installer provider allows Windows Management Instrumentation (WMI) to access information collected from Windows Installer applications.

The Windows Installer Provider is an optional Windows component. Optional installation of the Windows Installer provider ensures backward compatibility, but does not indicate future availability of the provider. For more information about the availability of the Windows Installer Provider, see Operating System Availability of WMI Components in the Windows Management Instrumentation (WMI) documentation.

For the current description of this WMI provider, see Windows Installer Provider in the main Windows Management Instrumentation (WMI) documentation. For a list of WMI classes that are supported by Windows Installer, see Installed Applications Classes.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Enumerating Components

Windows Installer 5.0 running on Windows Server 2008 R2 or Windows 7 can enumerate all components installed on the computer and obtain the key path for the component. A package authored for Windows Installer 5.0 can use the **MsiEnumComponentsEx**, **MsiEnumClientsEx**, and **MsiGetComponentPathEx** functions to search for components and products across user accounts and installation contexts. The **MsiEnumComponents**, **MsiEnumClients**, and **MsiGetComponentPath** functions only return information for components and products installed for the user account that called the function. Multiple calls to these functions, at least once for each user account, are required to collect information for the entire computer.

The **MsiEnumComponentsEx** function enumerates installed components. The function retrieves one component code each time it is called. The **Component Object** receives information about an installed component by this function.

The **MsiEnumClientsEx** function enumerates products that are clients of a specified installed component. The **Client Object** receives information about a client by this function.

The **MsiGetComponentPathEx** function returns the full path to an installed component. The function returns the registry key if the key path for the component is a registry key. The **ComponentInfo Object** receives information about an installed component by this function.

**Windows Installer 4.5 or earlier:** Not supported. This functionality is available beginning with Windows Installer 5.0 running on Windows 7 or Windows Server 2008 R2.

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Services Configuration

Services configuration enables the Windows Installer to customize the services on a computer. Developers can author a Windows Installer package to install, stop, start and delete services during an installation by using the [ServiceControl](#) and [ServiceInstall](#) tables and the [InstallServices](#), [StopServices](#) and [DeleteServices](#) actions.

Beginning with packages written for Windows Installer 5.0, developers can also use the [MsiConfigureServices](#) standard action and the [MsiServiceConfig](#) table to configure the extended service customization options available with Windows 7 and Windows Server 2008 R2 and Windows Vista and Windows Server 2008. Existing installation packages written for versions of the Windows Installer that did not include the [MsiServiceConfig](#) table can be still be installed using Windows Installer 5.0. The services configuration feature of the Windows Installer cannot configure network service accounts, install shared service host (svchost) processes, or restart services stopped as part of the installation.

**Windows XP and Windows Server 2003 or earlier:** Not supported. The service configuration tables and standard actions are available beginning with Windows Installer 5.0 running on Windows 7 and Windows Server 2008 R2 and Windows Installer 4.5 running on Windows Vista and Windows Server 2008.

You must include the [MsiConfigureServices](#) action in the [InstallExecuteSequence](#) table to request the service configurations that you specify in the [MsiServiceConfig](#) table. The Windows Installer uses the information in the [MsiServiceConfig](#) table only when the [MsiConfigureServices](#) standard action is included in a sequence table. The [MsiConfigureServices](#) standard action also uses information in the [ServiceControl](#) and [ServiceInstall](#) tables.

To request that the system give only required privileges to a particular service, specify the service and the [SERVICE\\_CONFIG\\_REQUIRED\\_PRIVILEGES\\_INFO](#) configuration option in the [MsiServiceConfig](#) table. Remove the unneeded privileges from the service's process token. This option can be used to configure services run in the security context of the LocalSystem, LocalService, or

NetworkService service user accounts.

To request that the system delay the automatic start of a service for a time after the start of all other auto-start services, specify the service and the `SERVICE_CONFIG_DELAYED_AUTO_START` option in the `MsiServiceConfig` table. The service being delayed must be installed by the current package with `SERVICE_AUTO_START` specified in the `ServiceInstall` table or the service must already be installed as an auto-start service.

To request that the system reserve a resource for the exclusive use of a particular service, specify the service, the service SID type, and the `SERVICE_CONFIG_SERVICE_SID_INFO` configuration option in the `MsiServiceConfig` table. Add the service's SID to the resource's Access Control List (ACL) for the resource.

To request that the Service Control Manager (SCM) wait after sending the `SERVICE_CONTROL_PRESHUTDOWN` notification to a service, do the following. Specify the service, the length of time the SCM should wait, and the `SERVICE_CONFIG_PRESHUTDOWN_INFO` configuration option in the `MsiServiceConfig` table.

To configure when the system should run actions after the failure of a service, specify the service and the `SERVICE_CONFIG_FAILURE_ACTIONS_FLAG` option in the `MsiServiceConfig` table. Add the actions to be run to the [MsiServiceConfigFailureActions table](#).

For more about the extended service customization capabilities introduced with the Windows Vista and Windows Server 2008 operating systems, see [Service Changes for Windows Vista](#).

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Single Package Authoring

A dual-purpose package is a Windows Installer 5.0 package that has been authored to be capable of installing an application in either the per-user or per-machine [installation context](#). Setup developers that use a dual-purpose package for their application can provide their users with a choice of installation context at installation time and can remove UAC credential prompts from per-user installations on Windows 7 or Windows Server 2008 R2. The development of a dual-purpose Windows Installer 5.0 package for installation on Windows 7 and Windows Server 2008 R2 is referred to as single package authoring.

You can begin to develop dual-purpose packages for Windows 7 and Windows Server 2008 R2 by using Windows Installer 5.0, the **MSIINSTALLPERUSER** property, the **ALLUSERS** property, and the per-user capable known folders and registrations of the Windows Shell. When Windows Installer 5.0 installs a dual-purpose package in the per-user context on Windows 7 or Windows Server 2008 R2, the installer directs file and registry entries to per-user locations and does not display UAC prompts for credentials. When Windows Installer 5.0 installs a dual-purpose package in the per-machine context, the installer directs files and registry entries to per-machine locations and prompts for UAC credentials to confirm that the user has sufficient privileges to install software for all users of the computer. Once Windows Installer 5.0 installs an application, it uses the same installation context for all subsequent updates, repairs, or removal of the application.

**Windows Installer 4.5 or earlier:** The **MSIINSTALLPERUSER** property and per-user versions of folders referenced by the **ProgramFilesFolder**, **CommonFilesFolder**, **ProgramFiles64Folder**, and **CommonFiles64Folder** properties are not supported. The FOLDERID\_UserProgramFiles and FOLDERID\_UserProgramFilesCommon folders are available beginning with Windows 7 and Windows Server 2008 R2. This means installations developed for Windows Installer 4.5 or earlier direct files and registry entries to FOLDERID\_ProgramFiles, FOLDERID\_ProgramFilesCommon, FOLDERID\_ProgramFilesX64, and FOLDERID\_ProgramFilesCommonX64. Because these are locations accessible to other users of the computer, Windows Vista

and later systems require the display of UAC prompts for credentials.

When a user installs a dual-purpose package authored for Windows Installer 5.0 with Windows Installer 4.5 or earlier, the installer ignores the **MSIINSTALLPERUSER** property. In this case, the installation can direct files and registry entries to locations accessible to other users and require the system to display UAC prompts for credentials. Windows Installer 5.0 can install a package that was developed for Windows Installer 4.5 or earlier, however, the installation directs files and registry entries to locations accessible to other users and requires the system to display UAC prompts for credentials.

## Development Guidelines

Adhere to the following single-package authoring guidelines to ensure that the package can be installed in either the per-user or per-machine context. Follow these guidelines to enable the user to choose at installation time a per-user or per-machine installation and to remove UAC prompts from per-user installations.

- Per-user installation requires Windows Installer 5.0 on Windows 7 or Windows Server 2008 R2. You should inform the user that the package supports per-machine installation of the application on earlier versions of the system.
- Initialize the values for the **ALLUSERS** and **MSIINSTALLPERUSER** properties in the [Property Table](#) of your dual-purpose package. Use **ALLUSERS** value of 2 and a **MSIINSTALLPERUSER** value of 1 as the initial values. This specifies per-user installation as the default for the dual-purpose package.
- Consider authoring a dialog box for the [user interface](#) of your dual-purpose package that enables the user to choose the context at installation time. Author the controls on this custom dialog box to set the values of the **ALLUSERS** and **MSIINSTALLPERUSER** properties. For the **ALLUSERS** value of 2, set **MSIINSTALLPERUSER** to a value of 1 to specify a per-user

installation and set **MSIINSTALLPERUSER** to an empty string ("" ) to specify a per-machine installation. Users can also set the **ALLUSERS** and **MSIINSTALLPERUSER** on the command line if they install the package from the command line.

- Validate the package using [Internal Consistency Evaluators - ICEs](#). The package must be able to pass validation by [ICE105](#) to be a valid dual-purpose package.
- Use the [Registry Table](#) and [RemoveRegistry Table](#) to redirect registry entries to the per-user parts of the registry during per-user installations. In a per-user installation, registry entries that have -1 in the Root column are redirected to HKEY\_CURRENT\_USER, and in a per-machine installation these are directed to HKEY\_LOCAL\_MACHINE. In a per-user installation, registry entries that have msidbRegistryRootClassesRoot (0) in the Root column are redirected to **HKCU\Software\Classes**, and in a per-machine installation, these are directed to **HKLM\Software\Classes**.
- Use the **ProgramFilesFolder** property in the [Directory table](#) of [32-bit Windows Installer Packages](#) to specify the locations of directories containing 32-bit components not shared across applications. When a user installs the dual-purpose package using the per-machine context, these components are saved in the Program Files folder on 32-bit versions of Windows and in the Program Files (x86) folder on 64-bit versions of the system. The components in these directories can be accessed by all users. When a user installs the dual-purpose package on Windows 7 or Windows Server 2008 R2 using the per-user context, these components are saved in the Programs folder of the current user (for example at %LocalAppData%\Programs) and can be accessed only by that user.
- Use the **CommonFilesFolder** property in the [Directory table](#) of [32-bit Windows Installer Packages](#) to specify the locations of directories containing 32-bit components shared across applications. When a

user installs the dual-purpose package using the per-machine context, these components are saved in the Common Files folder and can be accessed by all users. When a user installs the dual-purpose package on Windows 7 or Windows Server 2008 R2 using the per-user context, these components are saved in the Common folder of the current user (for example at %LocalAppData%\Programs\Common) and can be accessed only by that user.

- Use the **ProgramFiles64Folder** property in the Directory table of [64-bit Windows Installer Packages](#) to specify the locations of directories containing 64-bit components not shared across applications. When a user installs the dual-purpose package using the per-machine context, these components are saved in the Program Files folder. The components in these directories can be accessed by all users. When a user installs the dual-purpose package on Windows 7 or Windows Server 2008 R2 using the per-user context, these components are saved in the Programs folder of the current user (for example at %LocalAppData%\Programs) and can be accessed only by that user. For more information about authoring a package to install an application on 64-bit operating systems, see [Windows Installer on 64-bit Operating Systems](#).
- Use the **CommonFiles64Folder** property in the Directory table of 64-bit Windows Installer Packages to specify the locations of directories containing 64-bit components shared across applications. When a user installs the dual-purpose package using the per-machine context, these components are saved in the Common Files folder and can be accessed by all users. When a user installs the dual-purpose package on Windows 7 or Windows Server 2008 R2 using the per-user context, these components are saved in the Common folder of the current user (for example at %LocalAppData%\Programs\Common) and can be accessed only by

that user.

- Use the **ProgramFilesFolder** and **CommonFilesFolder** properties in the Directory table of 64-bit Windows Installer Packages to specify the location of directories containing 32-bit components. Use different names for the 32-bit and 64-bit versions of any components provided with the same name, or alternatively, save the versions in different folders. For example, add information to the Directory table to specify the location of the directory containing the 32-bit version as [ProgramFilesFolder]\SV Name\Application Name\x86 and the location of the directory containing the 64-bit version as [Program64FilesFolder]\SV Name\Application Name\x64. A per-machine installation then saves the 32-bit version in Program Files(x86)\SV Name\Application Name\x86 and saves the 64-bit version in Program Files\SV Name\Application Name\x64. A per-user installation saves the 32-bit version in %LocalAppData%\Programs\SV Name\Application Name\x86 and installs the 64-bit version in %LocalAppData%\Programs\SV Name\Application Name\x64.
- Store per-user configuration data for the application under \Users\username\AppData.
- Store templates and files generated by the application in subfolders under \Users\username.
- If your application uses shell extensions, you should use the per-user-capable **shell extensibility** points that are available beginning in Windows 7 or Windows Server 2008 R2.
- Do not use custom actions in your package that require elevated privileges to run. The **CustomAction** table should contain no custom actions that have been marked to run with elevated privileges. For more information about elevated custom actions, see **Custom Action Security**.
- Do not write in any global system folders. The Directory table should

not contain a reference to any of the following system folder properties.

**[AdminToolsFolder](#)**

**[CommonAppDataFolder](#)**

**[FontsFolder](#)**

**[System16Folder](#)**

**[System64Folder](#)**

**[SystemFolder](#)**

**[TempFolder](#)**

**[WindowsFolder](#)**

**[WindowsVolume](#)**

- Do not install a common language runtime assembly into the global assembly cache (GAC.) For more information about installing assemblies to the global assembly cache, see [Adding Assemblies to a Package](#) and [Installation of Common Language Runtime Assemblies](#).
- Do not install any ODBC data sources. Do not use the [ODBCDataSource](#) table to install a data source.
- Do not install any services. Do not use the [ServiceInstall](#) table to install a service.

## Example

An example of a dual-purpose package is provided in the [Windows SDK Components for Windows Installer Developers](#) as the file PUASample1.msi. If you have the current SDK, you have access to all the tools and data necessary to reproduce the sample installation package. For more information about this example, see [Single Package Authoring Example](#).

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Windows Installer Guide

The Windows Installer Guide contains information of interest to authors of Windows Installer packages and users of the Windows Installer service.

- [Digital Signatures and Windows Installer](#)
- [Assemblies](#)
- [User Interface](#)
- [Standard Actions](#)
- [Custom Actions](#)
- [Properties](#)
- [Summary Information Stream](#)
- [Patching and Upgrades](#)
- [Database Transforms](#)
- [Package Validation](#)
- [Merge Modules](#)
- [Windows Installer Bootstrapping](#)
- [Windows Installer on 64-bit Operating Systems](#)
- [Windows Installer and Logo Requirements](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Digital Signatures and Windows Installer

The Windows Installer can use digital signatures to detect corrupted resources. A signer certificate may be compared to the signer certificate of an external resource to be installed by the package. For more information regarding the use of digital signatures, digital certificates, and **WinVerifyTrust**, see the [Security](#) section of the Microsoft Windows Software Development Kit (SDK).

With Windows Installer, digital signatures can be used with Windows Installer packages, transforms, patches, merge modules, and external cabinet files. Windows Installer is integrated with Software Restriction Policy on Microsoft Windows XP. Policies can be created to allow or prevent installations based upon different criteria, including a particular signer certificate or publisher. The Windows Installer can perform signature validation of external cabinet files on all platforms where CryptoAPI version 2.0 is installed.

Note that the sample Setup.exe bootstrap provided with the Windows Installer SDK performs a signature check on a Windows Installer package prior to initiating the installation.

Performing an [administrative installation](#) removes the digital signature from the package. An administrative installation modifies the installation package in order to add the AdminProperties stream, which would invalidate the original digital signature. An administrator can resign the package.

Applying a patch to an administrative installation also removes the digital signature from the package. The reason is because the changes persist in the patched installation package of the administrative installation. An administrator can resign the package.

Starting with Windows Installer version 3.0, [User Account Control \(UAC\) Patching](#) enables non-administrator users to patch applications installed in the per-machine context. UAC patching is enabled by providing a signer certificate in the [MsiPatchCertificate](#) table and signing patches with the same certificate.

For more information, see [Digital Signatures and External Cabinet Files](#), [Windows Installer and Software Restriction Policy](#), [Authoring a Fully Verified Signed Installation](#), and [A URL based Windows Installer Installation Example](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Digital Signatures and External Cabinet Files

Windows Installer can use digital signatures to detect corrupted resources. When installing an external resource, the signer certificate belonging to the resource may be verified against a reference signer certificate authored in the package. The installer cannot verify signatures for internal cabinets. It can only verify digital signatures by using the [MsiDigitalSignature table](#) and [MsiDigitalCertificate table](#).

Windows Installer does the following when installing a file stored in an external cabinet:

- The installer checks to see whether the media entry for that external cabinet is listed in the [MsiDigitalSignature table](#). A file stored in an external cabinet is identified by having an entry in the Cabinet column of the [Media table](#) that is not prefixed by a '#' character.
- Before opening the external cabinet, the installer calls `WinVerifyTrust` to extract the current certificate and hash information. If there is a mismatch between the current signature information on the cabinet and the signature information authored in the package, the installation fails. The installation fails because the cabinet may have been compromised and cannot be trusted.

For more information regarding the use of digital signatures, digital certificates, and **WinVerifyTrust**, see the [Security](#) section of the Microsoft Windows Software Development Kit (SDK).

For more information, see [MsiGetFileSignatureInformation](#), [MsiDigitalCertificate table](#), and [MsiDigitalSignature table](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Windows Installer and Software Restriction Policy

Windows Installer is integrated with Software Restriction Policy in Microsoft Windows XP. Software Restriction Policy is configurable through group policy. Software Restriction Policy allows an administrator to restrict both administrators and nonadministrators from running files based upon the path, URL zone, hash, or publisher criteria. Software Restriction Policy has two levels: unrestricted and disallowed. The Windows Installer only installs packages allowed to run at the unrestricted level.

Patches or transforms must also be allowed to run at the unrestricted level. If a package, patch, or transform is configured to run at a level different from unrestricted, the Windows Installer displays an error message and logs an entry in the application event log. Software restriction policy is evaluated the first time an application is installed, when a new patch is applied, and when the installation package is re-cached.

If a package, patch, or transform is restricted, the Windows Installer displays an error message and writes an [Event Logging](#) entry in the application event log. Software restriction policy is evaluated the first time an application is installed, when a new patch is applied, and when the installation package is re-cached.

For more information on software restriction policy, consult the product documentation and [search the TechNet Site](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring a Fully Verified Signed Installation

You can use these guidelines to cover an entire Windows Installer installation by a digital signature.

Authors of Windows Installer installations must adhere to the following to ensure that all parts of the installation are covered by a digital signature:

- Use internal cabinet files, or use signed external cabinet files and correctly author the [MsiDigitalSignature table](#) and [MsiDigitalCertificate table](#).
- Use only custom actions stored within the package or installed with the package.
- Sign the installation package.
- Include an [MsiPatchCertificate table](#) in the package. To enable [User Account Control \(UAC\) Patching](#), this table must contain information used to identify the signer certificates used to digitally sign patches. UAC patching enables the author of the installation package to identify digitally-signed patches that can be applied in the future by non-administrator users.

For an example showing how to author a signed installation using automation, see [Authoring a Fully Verified Signed Installation Using Automation](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring a Fully Verified Signed Installation Using Automation

The following sample demonstrates how to populate the [MsiDigitalCertificate table](#) and [MsiDigitalSignature table](#) by using a Visual Basic for Applications (VBA) subroutine. For more information about securing Windows Installer packages see [Guidelines for Authoring Secure Installations](#).

The [FileSignatureInfo method](#) returns a SAFEARRAY of bytes. For more information, see the **SAFEARRAY Data Type**. The data from this array must be converted to Unicode because Visual Basic does not have a way to write bytes straight into a file. The [SetStream method](#) can then use the file of converted data to write stream data into a specified record field of a [Record object](#). Note that conversion of the byte data to Unicode can potentially change the data and that the converted data must match the original data for correct signature verification. The package author must ensure that the original and converted data match.

```
Sub PopulateDigitalSignature()  
  
    Dim Installer As Object  
    Dim Database As Object  
    Dim x() As Byte  
  
    Const szSignedCabinet = "c:\test.cab"  
    Const szCertFile = "c:\temp\test.cer"  
    Const szDatabase = "c:\test.msi"  
  
    Set Installer = CreateObject("WindowsInstaller.Installer")  
  
    x = Installer.FileSignatureInfo(szSignedCabinet, 0, 'msi'  
  
    Dim fs, ts  
    Dim s As String  
    Set fs = CreateObject("Scripting.FileSystemObject")  
    Set ts = fs.CreateTextFile(szCertFile, True) 'Cr  
  
    s = StrConv(x, vbUnicode)
```

```
ts.Write s
ts.Close

Set Database = Installer.OpenDatabase(szDatabase, msiOp
Set ViewCert = Database.OpenView("SELECT * FROM `MsiDig
ViewCert.Execute 0
Set ViewSig = Database.OpenView("SELECT * FROM `MsiDig
ViewSig.Execute 0

Set RecordCert = Installer.CreateRecord(2)
RecordCert.StringData(1) = "Test"
RecordCert.SetStream 2, szCertFile
ViewCert.Modify msiViewModifyInsert, RecordCert

Set RecordSig = Installer.CreateRecord(4)
RecordSig.StringData(1) = "Media"
RecordSig.StringData(2) = "1"
RecordSig.StringData(3) = "Test"
ViewSig.Modify msiViewModifyInsert, RecordSig

Database.Commit
    fs.DeleteFile(szCertFile)
End Sub
```

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Assemblies

Windows Installer can install, remove, and update Win32 assemblies and assemblies used by the common language runtime of the Microsoft .NET Framework. An assembly is handled by the Windows Installer as a single installer component. All of the files that constitute an assembly must be contained by a single installer component that is listed in the [Component](#) table.

Windows Installer running on Windows Vista and Windows XP can install side-by-side assemblies. Side-by-side assemblies can safely share assemblies among multiple applications and can offset the negative effects of assembly sharing, such as DLL conflicts. Instead of a single version of an assembly that assumes backward compatibility with all applications, side-by-side assembly sharing enables multiple versions of a COM or Win32 assembly to run simultaneously on the system. This improved capability to isolate applications is an important part of the Microsoft .NET Framework. For more information, see [Isolated Applications and Side-by-side Assemblies](#).

The following sections describe the use of assemblies with the Windows Installer.

- [Adding Assemblies to a Package](#)
- [Installing and Removing Assemblies](#)
- [Updating Assemblies](#)
- [Reinstallation Modes of Common Language Runtime Assemblies](#)
- [Assembly Registry Keys Written by Windows Installer](#)

For information about installing COM and COM+ 1.0 applications, see [Installing a COM+ Application with the Windows Installer](#), [Installing a COM Component to a Private Location](#), and [Make a COM Component in an Existing Package Private](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Adding Assemblies to a Package

Windows Installer developers can use the guidelines in this topic to author Windows Installer packages that contain assemblies.

The following guidelines apply to Win32 assemblies, and assemblies that the common language runtime of the Microsoft .NET Framework uses.

- A Windows Installer component should contain no more than one assembly.
- All of the files in the assembly should be in a single component.
- Each component that contains an assembly should have an entry in the [MsiAssembly](#) table.
- The strong assembly cache name of each assembly should be authored into the [MsiAssemblyName](#) table.
- Use the [Registry](#) table instead of the [Class](#) table when you register COM Interop for an assembly.
- Assemblies that have the same strong name are the same assembly. When the same assembly is installed by different applications, the components that contain the assembly should use the same value for the ComponentId in their [Component](#) tables.

**Note** Product advertisements identify assemblies that can be installed and used by different applications. Product advertisements do not identify private assemblies.

## Adding Win32 Assemblies

Use the following guidelines when you include Win32 assemblies:

- The KeyPath value in the Component table for a component that contains a Win32 assembly should not be Null.
- The KeyPath value in the Component table for a component that contains a Win32 policy assembly should be the manifest file.

- The KeyPath value in the Component table for a component that contains a Win32 assembly, that is not a policy assembly, should not be the manifest file or catalog file. It should be a different file in the assembly.
- Add a row to the MsiAssemblyName table for each name and value pair that are listed in the **assemblyIdentity** section of the Win32 assembly's manifest.

## Adding Assemblies used with the .NET Framework

Use the following guidelines when you include assemblies that the common language runtime of the .NET Framework uses.

- The KeyPath value in the Component table for a component that contains the assembly should not be Null.
- When you install an assembly used by the common language runtime to the global assembly cache, the value in the File\_Application column of the MsiAssembly table must be Null.
- Add a row to the MsiAssemblyName table for each attribute of the assembly's strong name. All assemblies must have the Name, Version, and Culture attributes that are specified in the MsiAssemblyName table. A publicKeyToken attribute is required for a global assembly. The following table is an example of the MsiAssemblyName table for a global assembly for use by the common language runtime.

MsiAssemblyName Table

Component	Name	Value
ComponentA	Name	simple
ComponentA	version	1.0.0.0
ComponentA	Culture	neutral
ComponentA	publicKeyToken	9d1ec8380f483f5a

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installing and Removing Assemblies

When installing an assembly to an isolated location for a specific application, the application must be specified in the File\_Application column of the [MsiAssembly table](#). This field is a key into the [File table](#). The installer uses the directory structure that is specified in the [Directory table](#) to install the assembly into the same directory as the component that contains this file.

When installing an assembly to the global assembly cache, the value in the File\_Application column of the MsiAssembly Table must be Null.

The following sections describe the installation of Win32 and common language runtime assemblies:

- [Installation of Win32 Assemblies](#)
- [Installation of Common Language Runtime Assemblies](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installation of Win32 Assemblies

Install Win32 assemblies by making them a component of Microsoft Windows Installer package that installs or updates your application. When you author the [MsiAssembly table](#) and [MsiAssemblyName table](#), this identifies the component in the Component\_ column as an assembly. The installer will set the [MsiWin32AssemblySupport](#) property to the file version of Sxs.dll on operating systems that can support Win32 assemblies and not set this property otherwise.

Side-by-side assemblies are not available on systems earlier than Windows XP. For more information, see [Isolated Applications and Side-by-side Assemblies](#).

This table summarizes the ways Win32 assemblies can be installed on different Windows operating systems. For information, see [Shared Assemblies](#), [Private Assemblies](#), and [Side-by-Side Assemblies](#).

<b>Operating System</b>	<b>Windows Installer writes assembly information into the registry.</b>	<b>Shared assemblies can be used.</b>	<b>Private assemblies can be used.</b>	<b>Side-by-side assemblies can be used.</b>
Windows Vista	No	Yes	Yes	Yes
Windows XP	No	Yes	Yes	Yes
Windows 2000	Yes	Yes	Yes	No

The following sections describe how to author a Windows Installer package to install Win32 assemblies as shared, private, or side-by-side on different Windows operating systems.

- [Installing Win32 Assemblies for Side-by-Side Sharing on Windows XP](#)
- [Installing Win32 Assemblies for the Private Use of an Application on Windows XP](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Shared Assemblies

A Win32 assembly can be installed as a shared assembly and be available for use by multiple applications on the computer. Shared assemblies should be installed by a Windows Installer package used to install or update an application.

Shared assemblies are installed as [side-by-side assemblies](#). The Windows Installer installs shared side-by-side assemblies into the Winsxs folder. The assemblies are not registered globally on the system, but they are globally available to any application that specifies a dependency on the assembly in a manifest file. For more information, see [Isolated Applications and Side-by-side Assemblies](#).

On operating systems earlier than Windows XP, shared assemblies are usually installed in the Windows system folder and registered globally. The latest installed version is available to any application that binds to it.

For information on how to author a Windows Installer package to install a shared assembly, see the following:

- [Installing Win32 Assemblies for Side-by-Side Sharing on Windows XP](#)
- [Installing Win32 Assemblies for the Private Use of an Application on Windows XP](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Private Assemblies

A Win32 assembly can be installed as a private assembly and be exclusively available for use by one application. Private assemblies should be installed by a Windows Installer package used to install or update an application.

On Windows XP, private assemblies are installed as [side-by-side assemblies](#). The Windows Installer installs private side-by-side assemblies in a folder private to the application. Commonly the folder that contains the applications executable file. The dependency of the application on the private assembly is specified in an application manifest file. For more information, see [Isolated Applications and Side-by-side Assemblies](#).

On operating systems earlier than Windows XP, a copy of the private assembly and a .local file is installed into a private folder for the exclusive use of the application. A version of the assembly is also globally registered on the system and available for any application that binds to it. The global version of the assembly may be the version installed with the application or an earlier version. The global version is determined by the same rules use by [Isolated Components](#).

Note that the Windows Installer cannot install a private assembly in a location having a path that contains more than 234 characters, including the terminating null character.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Side-by-Side Assemblies

Side-by-Side assembly sharing is an infrastructure that is used to do the following:

- Safely share assemblies among multiple applications
- Offset some negative effects of sharing, for example, DLL conflicts.

Instead of having a single version of an assembly that assumes backward compatibility with all applications, side-by-side assembly sharing enables multiple versions of a COM or Win32 assembly to run simultaneously on a system. For more information, see [Isolated Applications and Side-by-side Assemblies](#).

Side-by-Side Assemblies can be installed as [Shared Assemblies](#) or as [Private Assemblies](#).

Side-by-Side Assemblies are not available on systems earlier than Windows XP.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installing Win32 Assemblies for Side-by-Side Sharing on Windows XP

The following describes how to create a Windows Installer package to install a Win32 assembly. The package installs a side-by-side assembly in the Wsxs folder for the shared use of the application. After installing the package, the shared assembly is globally available to any application that specifies a dependence on the assembly in an assembly manifest file. The installer does not globally register the side-by-side assembly on the system.

Note that you may install shared side-by-side assemblies using [merge modules](#).

Before continuing you should understand how to author a Windows Installer package without assemblies. For an example of how to author a simple installation, see [An Installation Example](#).

## ► To install a shared assembly side-by-side on Windows XP

1. Define a Windows Installer component that includes the Win32 assembly. This component may contain other resources that should always be installed or removed with the assembly. All the other components of the application can be authored just as for an installation without assemblies. Add a row to the [Component table](#) for the component containing the Win32 assembly. Enter a valid Windows Installer [GUID](#) for this component code. Do not use the manifest file as the key path for this component.
2. Add a row to the [FeatureComponents table](#) tying the component to a Windows Installer feature. For information see, [Components and Features](#). A Windows Installer feature should be a piece of the application's functionality that is recognizable to a user. The assembly is activated when this feature is selected by a user or faulted in by an application. If the assembly defines an additional

feature, then add an additional row to the [Feature table](#) for the feature's attributes. This step is not needed when authoring a merge module.

3. For side-by-side assemblies, binding and activation information, such as COM classes, interfaces, and type libraries, is stored in manifest files rather than the registry. Shared assemblies store this information in an assembly manifest. On systems that support side-by-side assemblies, the installer skips processing any information about the component entered in the [Extension table](#), [Verb table](#), [TypeLib table](#), [MIME table](#), [Class table](#), [ProgId table](#), and [AppId table](#). Binding and activation information may be entered into these tables for use by systems that do not support side-by-side assembly sharing.
4. Side-by-side installation does not globally register the assembly, the installer skips self-registering the component if any self-registration information has been entered in the [SelfReg table](#). Self-registration information may be entered into the SelfReg table for self-registration of the component on systems that do not support side-by-side assembly sharing.
5. Add any other registry information, exclusive of binding and activation or self-registration of the component, to the [Registry table](#), [RemoveRegistry table](#), and [Environment table](#).
6. Because this is a shared assembly do not generate a .local file. Do not include information for this component in the [IsolatedComponent table](#). The installer skips the IsolatedComponent table for this component on operating systems that support side-by-side sharing. Add information to the IsolatedComponent table if you would like the assembly to be private on systems that support .local files.
7. To enable side-by-side sharing, the Win32 assembly must be installed into the Winsxs folder. This is accomplished by leaving

the File\_Application column of the [MsiAssembly table](#) null for the assembly. This tells the installer to install the assembly to the WinSxS folder, instead of to the folder of the component. Add a row to the [MsiAssembly table](#) for the component that contains the Win32 assembly. Enter a value of 1 in the Attributes field of the MsiAssembly table to specify that this is a Win32 assembly. For a shared assembly, leave the File\_Application field empty. Add the [MsiPublishAssemblies action](#) to the [InstallExecuteSequence table](#) or [AdvtExecuteSequence table](#). Add the [MsiUnpublishAssemblies action](#) to the InstallExecuteSequence table.

8. Add rows to the [MsiAssemblyName table](#) for the component. Add one row for each name and value pair specified in the assemblyIdentity section of the manifest. For an example, see [MsiAssemblyName table](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installing Win32 Assemblies for the Private Use of an Application on Windows XP

The procedure in this topic identifies how to create a Windows Installer package to install a Win32 assembly. The package installs the assembly and an application manifest file into an authored folder that the application uses. The application manifest specifies the dependence of the application on the private assembly. After installing the package, the private assembly is available for the exclusive use of the application. The assembly dependence that is specified in the application manifest overrides (for this application) any other global assembly dependencies that are specified in assembly manifest files.

Before you continue, it is a good idea to understand how to author a Windows Installer package without assemblies. For more information, see [An Installation Example](#).

## ► To install a private assembly on Windows XP

1. Define a Windows Installer component that includes the Win32 assembly and the application manifest file. This component can contain other resources that should always be installed or removed with the assembly. The remaining steps of this procedure describe how to author the installation database to install this component. All the other components of the application can be authored the same as a Windows 2000 installation without assemblies.
2. Add a row to the [Component table](#) for the component that contains the Win32 assembly and application manifest file. Enter a valid Windows Installer [GUID](#) for this component code. For more information, see [Changing the Component Code](#) and [What happens if the component rules are broken?](#)
3. The installer copies the assembly manifest file into the folder that

contains the file specified in the File\_Application field of the [MsiAssembly table](#).

4. Add a row to the [FeatureComponents table](#) tying the component to a Windows Installer feature. For more information, see [Components and Features](#). A Windows Installer feature should be a piece of the application functionality that a user can recognize. The assembly is activated when this feature is selected by a user or faulted in by an application. If the assembly defines an additional feature, then add an additional row to the [Feature table](#) for the feature attributes. This step is not required if authoring a merge module.
5. For side-by-side assemblies, binding and activation information, for example, COM classes, interfaces, and type libraries, is stored in manifest files rather than the registry. Private assemblies store this information in an assembly manifest. On systems that support side-by-side assemblies, the installer skips processing any information about the component that is entered in the [Extension table](#), [Verb table](#), [TypeLib table](#), [MIME table](#), [Class table](#), [ProgId table](#), and [AppId table](#). Binding and activation information can be entered into the tables for use by systems that do not support side-by-side assembly sharing.
6. Side-by-side installation does not register the assembly globally. The installer skips self-registering the component if any self-registration information is entered in the [SelfReg table](#). Self-registration information can be entered into the SelfReg table for self-registration of the component on systems that do not support side-by-side assembly sharing.
7. Add any other registry information, exclusive of binding and activation or self-registration of the component, to the [Registry table](#), [RemoveRegistry table](#), and [Environment table](#).
8. The installer skips the [IsolatedComponent table](#) for this

component on operating systems that support side-by-side sharing. Enter information into this table if you want the assembly to be private on systems that support local files.

9. Add a row to the [MsiAssembly table](#) for the component that contains the Win32 assembly. Enter a value of 1 in the Attributes field of the MsiAssembly table to specify that this is a Win32 assembly. Enter the file key of the private assembly in the File\_Application field of the [MsiAssembly table](#). Add the [MsiPublishAssemblies action](#) to the [InstallExecuteSequence table](#) or [AdvtExecuteSequence table](#). Add the [MsiUnpublishAssemblies action](#) to the [InstallExecuteSequence table](#). Author a folder for the assembly and manifest file into the [Directory table](#). This folder should be in the application's root directory and contain the file specified in the File\_Application field of the [MsiAssembly table](#). During the installation of the application, the installer resolves the [Directory table](#) for the path to this folder. For more information, see [Using the Directory Table](#).
10. Add rows to the [MsiAssemblyName table](#) for the component. Add one row for each name and value pair specified in the assemblyIdentity section of the manifest. For more information, see [MsiAssemblyName table](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installation of Common Language Runtime Assemblies

The following sections describe the installation of assemblies to the global assembly cache:

- [Installation of Assemblies to the Global Assembly Cache](#)
- [Rollback of Assemblies in the Global Assembly Cache](#)
- [Removal of Assemblies from the Global Assembly Cache](#)
- [Reinstallation Modes of Common Language Runtime Assemblies](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installation of Assemblies to the Global Assembly Cache

The Windows Installer installs common language runtime assemblies into the global assembly cache using the Microsoft .NET Framework. When installing assemblies to the global assembly cache, the installer cannot use the same directory structure and file version rules it uses when installing regular Windows Installer components. Regular Windows Installer components may be installed into multiple directory locations by different products. Assemblies can exist only once in the assembly cache. Each assembly is added and removed from the assembly cache as an indivisible whole; therefore, all the files comprising an assembly are always installed or removed together.

The disk cost of regular Windows Installer components and common language runtime assemblies are calculated differently. The total disk cost of a regular Windows Installer component includes local costs, source costs, and removal costs. For details, see [File Costing](#). This method cannot be used to cost common language runtime assemblies because these may have clients other than the Windows Installer. The cost of common language runtime assemblies must be determined by querying the Microsoft .NET Framework common language runtime.

The Windows Installer uses a two-step transactional process to install products containing common language runtime assemblies. This enables the rollback of assembly installation and removal. For more information, see [Rollback of Assemblies in the Global Assembly Cache](#).

Note that assemblies installed to the global assembly cache by an installation in the per-user [installation context](#) are not protected by Windows File Protection. Assemblies that are installed to the global assembly cache by an installation in the per-machine installation context are protected by Windows Resource Protection.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Rollback of Assemblies in the Global Assembly Cache

A two-step process extends the Windows Installer's transaction model to products containing common language runtime assemblies. This enables the installer to rollback unsuccessful installations and removals of assemblies.

During the first step, the Windows Installer uses the Microsoft .NET Framework to create one interface for each assembly. The Windows Installer uses as many interfaces as there are assemblies being installed. Committing an assembly using one of these interfaces only means that the assembly is ready to replace any existing assembly with the same name, it does not yet replace it. If the user cancels the installation, or if there is a fatal installation error, the Windows Installer can still rollback the assembly to its previous state by releasing these interfaces.

After the Windows Installer completes installing all assemblies and Windows Installer components, the installer may initiate the second step of the installation. The second step uses a separate function to do the final commit of all the new common language runtime assemblies. This replaces any existing assemblies with the same name.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Removal of Assemblies from the Global Assembly Cache

The Windows Installer determines whether to allow the removal of a common language runtime assembly based upon a client list it keeps independently of the assembly. The Windows Installer keeps one pin bit to represent Windows Installer clients of the assembly. The installer pins the assembly for the first Windows Installer client and unpins it when the last Windows Installer client is removed. The assembly maintains a pin bit for every client of an assembly.

The Windows Installer is not directly responsible for the physical removal of common language runtime assemblies from the computer. The installer unpins the assembly when the last Windows Installer client is removed. If the Windows Installer is the last client of the assembly, the common language runtime provides the option to force a synchronous cleanup of the assembly. The cleanup process is atomic and there is no provision for a "rollback" at this point. All unpinning of common language runtime assemblies must occur after the user has had a chance to cancel the installation or removal.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Updating Assemblies

The information in this topic identifies the recommended guidelines for updating assemblies using Windows Installer patches.

Authors of assembly updates can use the following guidelines, which apply to all types of assemblies:

- The recommended method for updating an assembly is to change the strong name of the assembly in the [MsiAssemblyName](#) table. The new assembly version can be provided by a new component or by the same component that provides the old version.
- If the new assembly version is provided by the same component, do not change the assembly type from a .NET Framework assembly to a Win32 assembly or vice versa.
- If all applications on the system must use the updated assembly, you should deploy a policy assembly. A policy assembly can redirect applications on the system to use the new assembly version. Policy assemblies should be provided by creating a new component.
- Windows Installer 3.0 or later is required to uninstall assembly updates. For more information, see [Removing Patches](#).
- The newer version of the assembly should contain the same or higher versions of files from previously released assemblies.
- Windows Installer 3.0 can service .NET Framework and Win32 assemblies by a complete file replacement or by a smaller delta update. For more information, see [Reducing Patch Size](#).
- If your update changes the strong name of the assembly, the [MsiPatchOldAssemblyFile](#) table and [MsiPatchOldAssemblyName](#) table are required if the patch package does not have a [MsiPatchSequence](#) table. The [MsiPatchOldAssemblyFile](#) table and [MsiPatchOldAssemblyName](#) table are not required if the patch package has patch sequencing information in a [MsiPatchSequence](#)

table.

- Before releasing a new patch, test the patch by applying it with all previously released patches.

## Updating Win32 Assemblies

Use the following guidelines when you update Win32 assemblies:

- Change the strong name of the new assembly that is specified in the MsiAssemblyName table.
- If you want your application to always use the new version of the assembly without affecting the assembly version used by other applications on the system, use the same component to contain the new assembly version that you used for the old assembly version. Keep the same ComponentId in the [Component](#) table. After your application is patched, it only holds a reference to the new version of the assembly. The old version of the assembly can remain with the new version in the global assembly cache. This does not affect other applications on the system that use the old version of the assembly. When the same component is used for both the new and old versions of the assembly, your update can be a smaller delta patch.
- If the new version of the assembly is not compatible with all the systems that you want to install your application on, you can install the new and old versions of the assembly as side-by-side assemblies. To install both assembly versions side-by-side, create a new component to contain the new assembly version. The ComponentId in the Component table for the new component should be different from the ComponentId of the component with the old version. After your application is patched, it holds references to both assembly versions. Then the application can be directed to the compatible version of the assembly by a manifest. When different components are used for the new and old versions of the assembly,

your update uses complete file replacement.

## Updating .NET Framework Assemblies

Use the following guidelines when you update .NET Framework assemblies.

- Change the strong name of the new assembly that is specified in the MsiAssemblyName table.
- If you want your application to always use the new version of the assembly without affecting the assembly version used by other applications on the system, change the strong name of the new assembly that is specified in the MsiAssemblyName table, and use the same component to contain the new assembly version that you used for the old assembly version. Keep the same ComponentId in the Component table. After your application is patched, it only holds a reference to the new version of the assembly. The old version of the assembly can remain with the new version in the global cache. This does not affect other applications on the system that use the old version of the assembly. When the same component is used for both the new and old versions of the assembly, your update can be a smaller delta patch.
- If the new version of the assembly is not compatible with all systems that you want to install your application on, you can install the new and old versions of the assembly as side-by-side assemblies. To install both assembly versions side-by-side, change the strong name of the new assembly that is specified in the MsiAssemblyName table, and create a new component to contain the new assembly version. The ComponentId in the Component table for the new component should be different than the ComponentId of the component with the old version. After your application is patched, it holds references to both assembly versions. The application can then be directed to the compatible version of the assembly by a

manifest. When different components are used for the new and old versions of the assembly, your update uses complete file replacement.

- An in-place update overwrites the copy of a .NET Framework Assembly in the global assembly cache. This type of assembly update does not change the strong name of the assembly. Only the value in the FileVersion field of the MsiAssemblyName table is changed. The in-place update of a .NET Framework Assembly requires .NET Framework 1.1 SP1 or greater.

**Note** The in-place type of update overwrites the copy of the assembly in the global cache, and can break other applications if the new version of the assembly is not completely backward compatible. The recommended method for updating an assembly is to change the strong name of the assembly in the MsiAssemblyName table.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Reinstallation Modes of Common Language Runtime Assemblies

Common language runtime assemblies that are installed to the global assembly cache must have identical files in all occurrences of the assembly. This means that the reinstallation modes used by [MsiReinstallFeature](#) and [MsiReinstallProduct](#) must have different meanings for regular components and common language runtime assemblies. The following definitions of reinstall modes apply to common language runtime assemblies.

<b>Reinstall mode</b>	<b>Meaning for Microsoft .NET Components</b>
REINSTALLMODE_FILEMISSING	Install or reinstall the common language runtime component if it is missing or incomplete.
REINSTALLMODE_FILEOLDERVERSION	Install or reinstall the common language runtime component if it is missing or incomplete.
REINSTALLMODE_FILEEQUALVERSION	Install or reinstall the common language runtime component if it is missing or incomplete.
REINSTALLMODE_FILEEXACT	Install or reinstall the common language runtime component if it is missing or incomplete.
REINSTALLMODE_FILEVERIFY	Install or reinstall the common language runtime component if it is missing or if the existing component is incomplete or corrupt.

REINSTALLMODE\_FILEREPLACE

Always install or reinstall the common language runtime component.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Assembly Registry Keys Written by Windows Installer

If a Windows Installer package installs or advertises assemblies, the installer stores information about those assemblies in the local system registry. Please note that these registry keys are only intended to be used internally by Windows Installer and they should not be relied upon by your application. The content, location, and structure of information stored in these keys is subject to change. Applications should rely upon [MsiProvideAssembly](#) to manage assemblies.

Assemblies are registered by their assembly names. The names of the values stored in the following locations are the assembly names. The actual values are of the type REG\_MULTI\_SZ and contain data used by [MsiProvideAssembly](#) to install or repair assemblies.

## Information About Private Assemblies

Windows Installer stores information about private assemblies carried by Windows Installer packages that have been installed as managed per-user applications under the following registry key:

**HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Installer\Managed\<SID>\Installer\Assemblies\<path to config file>**

Windows Installer stores information about private assemblies carried by Windows Installer packages that have been installed per-user under the following registry key:

**HKCU\Software\Microsoft\Installer\Assemblies\<path to config file>**

Windows Installer stores information about private assemblies carried by Windows Installer packages and installed per-machine under the following registry key:

**HKLM\SOFTWARE\Classes\Installer\Assemblies\<path to config file>**

## Information About Global or Shared Assemblies

Windows Installer stores information about shared assemblies carried by

Windows Installer packages that have been installed as managed per-user applications under the following registry key:

**HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Installer\Managed\{SID}\Installer\Assemblies\Global**

Windows Installer stores information about shared assemblies carried by Windows Installer packages that have been installed per-user under the following registry key:

**HKCU\Software\Microsoft\Installer\Assemblies\Global**

Windows Installer stores information about shared assemblies carried by Windows Installer packages and installed per-machine under the following registry key:

**HKLM\SOFTWARE\Classes\Installer\Assemblies\Global**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# User Interface

Windows Installer provides a complete user interface (UI) for installing an application or product. The user interface presents the user with the options available to configure the installation and obtains information from the user about the pending installation process.

[About the User Interface](#) describes the functionality of the installer user interface. [Using the User Interface](#) describes the use of the installer internal UI.

Reference information on the installer internal dialog box and control styles and options is presented in [User Interface Reference](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# About the User Interface

Windows Installer contains functionality that enables installation package developers to author a graphical user interface (GUI) that is displayed to the end-user during installation. This user interface can exhibit [user interface wizard behavior](#), display dialog boxes and billboards, and present interactive controls to users during the installation.

The installer internal UI is managed and controlled through a set of database tables within Windows Installer itself. The installer provides only a small set of default dialog boxes that are intended to handle error and information messages. All custom dialog boxes must be created by the package author.

There is no specific Windows Installer API to allow a package author to create a UI programmatically. It is possible to use the Microsoft Windows API to create a UI programmatically; however, it is recommended that package authors use the internal UI provided.

Installer package authors create custom dialog boxes by entering the name of their custom dialog into the "\_Dialog" column of the dialog table and specifying the size, position, and other attributes using the remaining columns.

Windows Installer also implements a number of standard controls that a package author can place onto dialog boxes. Not all standard Microsoft Windows controls are available, and custom controls cannot be created for use with the installer UI.

Controls are created on a specific dialog box by entering the name of the dialog box, the primary key to the dialog box's entry in the dialog table, into the second field of the control table and specifying the control's size, position, and other attributes using the remaining columns.

Active controls must be linked to a ControlEvent in the [ControlEvent table](#) to allow user interaction with the installation. Passive controls that receive and display information must be subscribed to an appropriate ControlEvent in the [EventMapping table](#).

For more information about ControlEvents, see [ControlEvent Overview](#). Note that a control publishes a ControlEvent if listed in the ControlEvent table and subscribes to an event if listed in the EventMapping table.

The installer UI display during installation is managed through the UI sequence tables: [InstallUISequence Table](#), and [AdminUISequence Table](#). One of these sequence tables is executed depending on the top-level action that initiated the installation: [INSTALL](#), [ADMIN](#), or [ADVERTISE](#).

For more information about implementing a UI in Windows Installer, please see [Using the User Interface](#), [User Interface Schema](#), as well as the individual topics for dialog boxes and controls.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.



[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Dialog Box Overview

The process of creating a dialog box in Windows Installer is similar to the process of creating a dialog box programmatically using a Microsoft Windows API dialog box template. While a Windows dialog box template is stored in a null terminated-character string, Windows Installer stores the dialog box parameters in the Dialog table. The Dialog table contains an attributes column that is analogous to Window styles in the Microsoft Windows user interface API. However, the number of dialog style bits in Windows Installer is a reduced and specialized set.

To physically create the dialog box using the Windows user interface API, **DialogBox** is called and passes a pointer to the template. In Windows Installer, the dialog box is created during execution of one of the three UI sequence tables.

Windows Installer does not contain a default UI that package authors can utilize for their packages. It does contain a limited set of default dialog boxes that display error and information messages, but these are displayed only if Windows Installer queries the Dialog and UI Sequence tables and finds there are no custom dialog boxes available.

The installer SDK includes a minimal database named Uisample.msi that contains populated UI and execution sequence tables and a complete UI. This database is easily customized and merged on to any package.

Some standard actions and internal Windows Installer error conditions return a specific set of UI data and therefore require a dialog box with a specific set of controls to accommodate that data. Windows Installer checks two separate locations for the identifiers for these dialog boxes.

- Windows Installer contains a set of embedded dialog box names. The custom action queries the Dialog table for these reserved names.
- In each of the three UI sequence tables, dialog names with a sequence number of -1,-2, or -3 are displayed in response to exit, user requested exit, and fatal error messages from Windows Installer.

A complete list of reserved primary key dialog box names is available in [dialog boxes](#). Note that the primary key dialog box name is only reserved by the installer and there is no specific implementation of the dialog box within Windows Installer. Package authors must still populate all the fields in the database that specify the attributes and controls of these reserved dialog boxes.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Controls Overview

Windows Installer implements a number of standard controls that package authors can place onto dialog boxes. However, not all standard Microsoft Windows controls are available, and custom controls cannot be created for use with the installer UI.

Controls are created on dialog boxes in the installer in a manner similar to how dialog boxes are created programmatically using the Microsoft Windows user interface API. A control is created from a template recorded in the Control table. This template is slightly different in that it contains the unique name of the dialog box on which the control appears.

In the Microsoft Windows user interface API, user interaction is accomplished by creating a callback function to handle messages posted by the control. Also, most Microsoft Windows controls receive messages, such as those sent by the **SendMessage** function.

Communication between the user and controls is accomplished in the installer through the use of [ControlEvents](#). However, there is a limited set of ControlEvents that are specific to each control in the limited set of controls in Windows Installer. Controls may post more than one ControlEvent and may receive more than one ControlEvent.

Controls can publish specific ControlEvents through the use of the ControlEvent table. Controls can receive ControlEvents through the use of the [EventMapping table](#).

Windows Installer publishes ControlEvents during the execution of some actions as well, and controls subscribed to these ControlEvents in the EventMapping table receive them.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ControlEvent Overview

ControlEvents are analogous to Microsoft Windows messages in Win32-based applications. However, rather than creating a callback function to receive Windows messages and sending Windows messages with the **SendMessage** function, the user interface (UI) installer and controls publish [ControlEvents](#). Other controls and the installer can be specified to subscribe to particular ControlEvents that will then change attributes of the subscribing control. To add working controls to dialog boxes, the author of the UI specifies the publication of ControlEvents in the [ControlEvent table](#) and subscribes controls to ControlEvents in the [EventMapping table](#).

The installer will publish the following events to subscribing controls listed in the [EventMapping table](#). A [ProgressBar control](#) or [Billboard control](#) typically subscribes to SetProgress, the rest are subscribed to by [Text controls](#).

[ActionData ControlEvent](#)

[ActionText ControlEvent](#)

[SetProgress ControlEvent](#)

[TimeRemaining ControlEvent](#)

[ScriptInProgress ControlEvent](#)

The following events are published by the control when the item selection is moved in a [SelectionTree control](#) or [DirectoryList Control](#). Subscribing controls must be located on the same dialog box and listed in the [EventMapping table](#).

[IgnoreChange ControlEvent](#)

[SelectionDescription ControlEvent](#)

[SelectionSize ControlEvent](#)

[SelectionPath ControlEvent](#)

[SelectionAction ControlEvent](#)

[SelectionNoItems ControlEvent](#)

The following ControlEvents can be published at the discretion of a user

by interacting with a [PushButton control](#) or [CheckBox control](#) on a dialog box. The Checkbox control can only publish the AddLocal, AddSource, Remove, DoAction, and SetProperty events. With Windows Installer versions that shipped with Windows Server 2003 and later, the [SelectionTree control](#) can publish the DoAction, ControlEvent and SetProperty ControlEvents. The author of the UI should list the ControlEvent in the [ControlEvent table](#). The UI handler of the installer is the subscriber to these events.

[AddLocal ControlEvent](#)

[AddSource ControlEvent](#)

[CheckExistingTargetPath ControlEvent](#)

[CheckTargetPath ControlEvent](#)

[DoAction ControlEvent](#)

[EnableRollback ControlEvent](#)

[EndDialog ControlEvent](#)

[NewDialog ControlEvent](#)

[Reinstall ControlEvent](#)

[ReinstallMode ControlEvent](#)

[Remove ControlEvent](#)

[Reset ControlEvent](#)

[SetInstallLevel ControlEvent](#)

[SetProperty ControlEvent](#)

[SetTargetPath ControlEvent](#)

[SpawnDialog ControlEvent](#)

[SpawnWaitDialog ControlEvent](#)

[ValidateProductID ControlEvent](#)

A [PushButton control](#) can publish the following events to a subscribing [SelectionTree control](#) or [DirectoryList control](#) located in the same dialog box. The PushButton Control should be listed in the ControlEvent table and the subscribing controls should be listed in the EventMapping table.

[SelectionBrowse ControlEvent](#)

[DirectoryListUp ControlEvent](#)

[DirectoryListNew ControlEvent](#)

[DirectoryListOpen ControlEvent](#)

Control events generally require that the UI be run at the *full UI* level. Most ControlEvents will not work with a *reduced UI* or *basic UI* because these levels only display modeless dialog boxes. The ActionText, AddSource, SetProgress, TimeRemaining, and ScriptInProgress events are exceptions and will work in reduced or basic UI. For more information about UI levels, see [User Interface Levels](#).

You can run [custom actions](#) by publishing a ControlEvent from a [PushButton control](#) or [Checkbox control](#). Add a record to the [ControlEvent table](#) with the names of the dialog and the control publishing the ControlEvent. This control should publish a [DoAction ControlEvent](#) notifying the installer to run the custom action. On Windows XP or earlier systems, you cannot run a custom action by publishing a ControlEvent from a [SelectionTree control](#).

For a more information about particular ControlEvents, see the list of standard [ControlEvents](#) in [User Interface Reference](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Accessibility

Authors should be aware of the tables and fields in the following list when designing their UI to be in compliance with Active Accessibility guidelines. The user interface of an installer package should facilitate accessibility of the application or product to all users.

- Tooltip text is contained in the Help column of the [Control table](#). This text is shown by screen readers for controls containing a picture.
- The Text field of the [Control table](#) for the [VolumeCostList](#), [ListView](#), [DirectoryList](#) and [SelectionTree](#) controls is never displayed. Instead it can be read by screen review utilities as the description of the control. People who cannot use the visual information on the screen can interpret the information with the aid of a screen review utility. Screen review utilities (also referred to as a screen reader programs or speech access utilities) take the displayed information on the screen and direct it through alternative media, such as synthesized speech or a refreshable Braille display.
- Controls in dialog boxes should be linked using the Control\_Next field of the [Control table](#). The controls need to be authored such that they can all be reached by using the TAB key.
- Shortcut keys should be provided for gaining access to controls directly.
- Text color displayed in the user interface is set in the [TextStyle table](#). If the chosen text color is too close to the background's then the text's color choice is ignored.
- Text size and font is set in the [TextStyle table](#). Larger font sizes should be used for packages intended for the Asian market. For example, a font size of 10 points that is legible for English text may not necessarily be true for Chinese.
- For [Edit](#), [PathEdit](#), [ListView](#), [ComboBox](#) or [VolumeSelectCombo controls](#), screen readers take accName and accKeyboardShortcut

from a [Text Control](#) that must precede the control in the Control\_Next sequence of the dialog box. The screen reader takes accName from the Text field of the Text control and accKeyboardShortcut from the keyboard shortcut in the Text field, if a shortcut exists.

- Because static text cannot take the focus, a [Text control](#) that describes an [Edit](#), [PathEdit](#), [ListView](#), [ComboBox](#) or [VolumeSelectCombo control](#) must be made the first control in the dialog box to ensure compatibility with screen readers.
- For a [PushButton control](#) that displays an icon or bitmap image, accName and accKeyboardShortcut are specified in the Help field of the [Control table](#) record, to the left of the | separator.
- Avoid the use of text controls on top of white bitmaps because under High Contrast Black the text may become invisible.
- Do not put a black text control on a background that is an all white bitmap image. This text is not visible to a user who changes the Windows display to High Contrast Black.
- Do not put a white text control on a background that is an all black bitmap image. This text is not visible to a user who changes the Windows display to High Contrast White.
- Do not place transparent [Text controls](#) on top of colored bitmaps. The text may not be visible if the user changes the display color scheme. For example, text may become invisible if the user sets the high contrast parameter for accessibility.
- Note that the focus on a dialog box does not tab to a [RadioButtonGroup control](#) until one of the buttons in the group has been selected. To make the focus tab to this button group, specify one of the buttons as a default setting for the control.
- To provide screen reader programs with extra descriptive text about a [RadioButtonGroup control](#). Follow the example provided in [Adding Extra Text to Radio Buttons](#).
- The relative size of dialogs, controls and fonts can change

depending upon the font size chosen. For more information see [Installer Units](#). To ensure the correct display of text and controls in the user interface, setup developers should always test their application using all font sizes that might be used.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Adding Extra Text to Radio Buttons

Screen reader programs can only read the text of a [RadioButtonGroup control](#) that has been authored into the Text column of the [RadioButton table](#). If this text is an insufficient description of the radio buttons, overlapping [Text controls](#) can be added to provide extra descriptive text. These Text controls should overlap each other in the dialog box and have conditions set in the [ControlCondition table](#) so that only one Text control is shown at a time. The Text controls must not overlap the [RadioButtonGroup control](#) or other controls in the dialog because this makes the controls invisible to screen readers. When the user hovers the cursor over the Text control, the screen reader program reads the extra text.

In the following example the MySample dialog box has a [RadioButtonGroup control](#) named Colors with two choices for the value of the TheColor property. For each choice there is a Text control with a condition to hide or show, depending on the current choice selected for TheColor. An initial TheColor value is defined in the Property table. The Text controls have the extra descriptive text authored in the Text field of the [RadioButton table](#). When a user hovers the cursor over the Text control in the dialog box, the screen reader can read the extra description of the current choice.

## Dialog table

Dialog	HCentering	VCentering	Width	Height	Attributes	Title
MySample	50	50	200	180	3	Accessible radio buttons

## Control table

Dialog_	Control	Type	X	Y	Width	Height	Attribut
MySample	Colors	RadioButtonGroup	2	20	100	50	3
MySample	HowIsBlue	Text	20	80	150	15	2

MySample	HowIsGreen	Text			20	80	150	15	2

### RadioButton table

Property	Order	Value	X	Y	Width	Height	Text	Help
TheColor	1	Blue	10	10	80	15	&Blue	
TheColor	2	Green	10	30	80	15	&Green	

### Property table

Property	Value
TheColor	Blue

### ControlCondition table

Dialog_	Control_	Action	Condition
MySample	HowIsBlue	Hide	TheColor <> "Blue"
MySample	HowIsBlue	Show	TheColor = "Blue"
MySample	HowIsGreen	Hide	TheColor <> "Green"
MySample	HowIsGreen	Show	TheColor = "Green"

For more information, see [Accessibility](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# User Interface Wizard Behavior

To author a simple user interface, developers of installation packages can utilize a user interface design that adheres to interface wizard behavior.

The term user interface wizard behavior means that each dialog box in a sequence contains a **Next>>** button, which the user clicks to move to the next dialog box after entering data or configuring information in the current dialog box. If the user decides to go back and change any information entered in a previous dialog box, each dialog box contains a **<<Previous** button that the user clicks to go back. At the end of the wizard sequence, the user clicks a **Finish** button to begin the installation process.

The UI described in the UI example adheres to user interface wizard behavior. See [An Installation Example](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# User Interface Levels

Windows Installer provides package developers the capability to author an internal user interface that has multiple levels of functionality. Because the internal UI must be created by the author of the package, the behavior of the full UI, reduced UI, basic UI, and None levels depends on the installation package. The following table describes the functionality commonly ascribed to UI levels.

UI Level	Description
Full UI	Displays modal and modeless dialog boxes that have been authored into the internal UI. Displays authored <a href="#">Error Dialog</a> boxes. <b>Note</b> Modal dialog boxes require user input before the installation can continue and are specified by setting the <a href="#">Modal Dialog Style Bit</a> in the Attributes column of the <a href="#">Dialog</a> table. A modeless dialog box does not require user input for the installation to continue. A full UI commonly exhibits <a href="#">User Interface Wizard Behavior</a> .
Reduced UI	Displays any modeless dialog boxes that have been authored into the UI. Does not display any authored modal dialog boxes. Displays authored Error Dialog boxes. Displays <a href="#">Disk Prompt</a> messages. Displays <a href="#">FilesInUse Dialog</a> boxes.
Basic UI	Displays the built-in modeless dialog boxes that show progress messages. Displays built-in error dialog boxes. Does not display any authored dialog boxes. Prompts users to insert a disk by displaying a dialog box containing the <a href="#">DiskPrompt</a> property value.
None	None means a silent installation that displays no UI.

The level of the internal UI can be set using [MsiSetInternalUI](#). The installer sets the [UILevel](#) property to the current level of the UI.

If the [LIMITUI](#) property is set, the user interface (UI) level used when installing the package is restricted to Basic.

For an example of UI authoring, see [An Installation Example](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using the User Interface

This section is concerned primarily with how developers of installation packages author an installation user interface (UI) using the installer's database and internal UI. For more information about the difference between an internal and external UI see [About the User Interface](#).

To display a dialog box sequence or billboard during the installation, the name of the dialog box must be entered into the Action column of the appropriate action sequence table. The name of the dialog box must appear in the [InstallUISequence](#) or [AdminUISequence](#) table depending on whether the UI is scheduled to run under the [INSTALL](#), [ADVERTISE](#), or [ADMIN](#) action.

Although the installer supports the authoring of custom dialog boxes and billboards, there are also a number of reserved names for certain dialog box sequences. Because the installer uses these names when executing certain actions, these names must only be used with the types of dialog boxes for which they are reserved. A list of these reserved names, and a description of each of the special dialog box sequences, is given in [Dialog Boxes](#).

The properties of each dialog box or billboard in the UI must be specified in the [Dialog](#) and [BillBoard](#) tables, respectively. The style of each dialog box must also be specified in the Dialog table by setting the [dialog's style bit](#) flag.

Controls and text must be added to the dialog box, and these must be tied to [ControlEvents](#), to enable the user to interact with the installation process. See [Adding Controls and Text](#) for more information on how to add controls to a dialog box.

Windows Installer internal UI handler can selectively show or hide dialog boxes to control the level of end-user interactivity during the installation. These levels of end-user interactivity are referred to as full, reduced, basic, and none. See [User Interface Levels](#). for a full description of these UIlevels.

There are two methods to set the UI level. The UI level can be set programmatically with a call to [MsiSetInternalUI](#), and the first parameter of [MsiSetInternalUI](#) specifies the UI level. Package developers can also

set the UI level using the [command line option "/q"](#).

The behavior of each of the UI levels is determined by the authoring of the .msi file by the package developer. The author of an internal UI has flexibility in how these levels behave for a package. The availability of these levels depends on the authoring of the installation package. The author must specify every dialog box and control in the user interface in the Dialog and Control tables.

- A full UI typically exhibits [user interface wizard behavior](#), such as each dialog box in a sequence containing a **Next>>** button. This form of UI is familiar to many users and is the most common type of UI for an author to create. The installer presents a logical sequence of dialog boxes and prompts the user to interact with controls located in each dialog box.
- A reduced UI typically suppresses the display of wizard behavior.
- A basic UI typically only displays progress messages to the user.
- A UI level of None means a silent installation.

Windows Installer provides a unique progress bar indicator in the [ProgressBar control](#) that displays to the user an estimate of the total time remaining until the installation is complete. For more information about the progress bar, see [Authoring a ProgressBar Control](#).

UI authors should facilitate the accessibility of their application or product for all users. To learn more about Active Accessibility and Windows Installer, see [Accessibility](#).

For more information about authoring a user interface, see [Adding Controls and Text](#), [Authoring a ProgressBar Control](#), [Authoring Disk Prompt Messages](#), [Authoring a Conditional "Please Wait . . ." Message Box](#), and [Previewing the User Interface](#). For more information about author billboards see [Displaying Billboards on a Modeless Dialog](#)

Beginning with Windows Installer 4.5 a custom user interface can be embedded within the Windows Installer package. For an example of an embedded custom UI see [Using an Embedded UI](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Adding Controls and Text

Controls and text placed on dialog boxes and billboards enable the user to interact with the installation process. Add a dialog box to the user interface by including it in the [Dialog table](#) as described in [Using the User Interface](#). Fill dialog boxes and billboards with controls by populating the [Control table](#) and [BBControl table](#), respectively.

The initial attributes of control can be specified in the Attributes column of the [Control table](#). See [Control Attributes](#).

To make control attributes dependent upon a condition, use the [ControlCondition table](#) to disable, enable, hide, or show a control according to the value of a property or conditional statement. You can also use this table to override the specification of the default control entered into the [Dialog table](#).

To have an event change a control attribute, subscribe the control to a ControlEvent in the [EventMapping table](#). A ControlEvent specifies an action to be taken by the installer or a change in the attributes of one or more controls in the dialog box. See [ControlEvent Overview](#). Enter the attribute's identifier in the Attribute column and the ControlEvent's identifier in the Event column of the [EventMapping table](#).

Some controls facilitate the collection of information from the user. For example, a check box enables the user to set the value of a property. See the [CheckBox table](#), the [ComboBox table](#), the [ListBox table](#), the [RadioButton table](#), and the [ListView table](#).

Note that for security reasons, private properties cannot be changed by the user interacting with the user interface. If a property is to be set by the user interface it needs to be a public property and have a name in all upper case. See [About Properties](#).

You can either make your dialog box present information to the user or write it to a log in response to installation actions by filling in the [ActionText table](#).

Controls can have a predefined font style. To set the font and font style of a text string, prefix the string of displayed characters with `{style}` or `&style`. Where style is an identifier listed in the TextStyle column of the [TextStyle table](#). If neither of these are present, but the [DefaultUIFont](#)

property is defined as a valid text style, that font will be used.

It is recommended that the **DefaultUIFont** property of every installation package with a UI be set in the [Property table](#) to one of the predefined styles listed in the [TextStyle table](#). If this property is not specified, the installer uses the System font. This can cause the installer to improperly display text strings if the package's code page is different from the user's default UI code page.

For most controls, text is displayed using the character set that is specified by the code page of the database. This ensures that the correct character set is used with the information contained in the database. The exceptions to this are the [Edit](#), [DirectoryList](#), [PathEdit](#), and [DirectoryCombo](#) controls, which always display text using the user's default UI character set. The [Text](#), [ListBox](#), and [ComboBox](#) controls use the user's default UI character set if the [UsersLanguage Control Attribute](#) is set.

In some cases a control may be redrawn incorrectly when canceling out of a dialog box. This has to do with the order in which the controls receive WM\_PAINT messages after the **Cancel** dialog box is removed. To fix this, try changing the order of the controls in the Control table.

Controls should be made large enough to accommodate the entire text viewed at all font size settings. Controls should be made large enough to accommodate the entire localized text, if the text in the UI may be localized. Larger font sizes or localized text can require more space than the original text and can become truncated by a control that is made too small. For more information about localizing UI text see the section: [A Localization Example](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring a ProgressBar Control

Windows Installer contains functionality to display a progress indicator in an action display dialog. The [ProgressBar control](#) graphically represents the installation of individual components and reports either the total time elapsed relative to time remaining or the approximate total time remaining until the installation is complete.

To determine the total time anticipated for the installation, the installer tracks the total progress ticks anticipated by each action during generation of the execution script. When script generation is complete, the progress tick total is stored and the installation begins.

Progress messages detailing the elapsed number of progress ticks are sent to the active message handler as each action in the script is executed. On each progress message, the installer broadcasts a [SetProgress ControlEvent](#) to the currently active dialog box. The UI sequence should be authored to create the action display dialog box during script execution to receive the SetProgress ControlEvent messages from the installer.

When the action display dialog box receives a SetProgress ControlEvent, it checks the [EventMapping table](#) for any controls subscribing to the ControlEvent. The ProgressBar control on the action display dialog box is subscribed to with the Progress control attribute specified in the Attributes column. The Progress Control attribute specifies that the ProgressBar control will be passed "ticksSoFar" and "totalTicks" values along with the SetProgress ControlEvent. The progress bar control uses this information to advance the graphical bar from left to right for an installation and from right to left for a [rollback](#) operation.

In addition, the installer broadcasts a [TimeRemaining ControlEvent](#) on each progress message. The total time remaining for the installation is determined by first calculating the execution rate, which is the total number of ticks elapsed divided by the total time since the installation began. The total ticks remaining divided by the execution rate gives the approximate time remaining.

When the action display dialog box receives the TimeRemaining ControlEvent, it again looks in the EventMapping table for any controls that are subscribed. In order to display the time remaining, a [Text control](#)

must be subscribed to the TimeRemaining ControlEvent with the [TimeRemaining control attribute](#) specified in the Attributes column.

The subscribed Text control queries the [UIText table](#) for a parameterized template string named "TimeRemaining". This string has two parameters, [1] for minutes, and [2] for seconds. The Text control converts each value to minutes and seconds, evaluates the TimeRemaining template string, and updates the text control with the new information.

If the UI display level is set to basic or lower, the installer displays a default dialog box containing a progress bar and TimeRemaining text field. For more information, see [User Interface Levels](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring Disk Prompt Messages

Use the following guidelines to author a Windows Installer installation that displays a message box prompting the user to insert a disk.

## ► To display a message box prompting the user to insert a disk

1. Use the authoring capabilities of the installer to set the **DiskPrompt** property string in the **Property** table. This should include the name of the product being installed and a placeholder parameter within the string for the title printed on the disk. For example, for Microsoft Publisher the **DiskPrompt** property could be "Microsoft Publisher: Disk [1]", where [1] is the placeholder for the disk title.
2. Enter the titles of each of the disks being prompted for into separate rows of the DiskPrompt column of the **Media** table. For example, the first and only entry into the Media table could be "1–Install."
3. During the **InstallFiles** action the value from the DiskPrompt column of the Media table is substituted for the placeholder in the string of the **DiskPrompt** property.
4. The message displayed by the message box is created from a built-in template string in the **Error table**. This is Error 1302 and the template string is: "Please insert the disk: [2]", and the [2] represents a placeholder for the **DiskPrompt** property.

The example displays the following message to the user: "Please insert the disk: Microsoft Publisher: Disk 1 – Install."

Note that disk prompt messages get displayed by all **User Interface Levels** except None.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring a Conditional "Please Wait . . ." Message Box

The following example illustrates how to author a conditional message box that pops up and warns the user that a background task is still running whenever the user activates a displayed control prematurely.

The example also illustrates how the [SpawnWaitDialog ControlEvent](#) can generally be used to protect a control that triggers an action dependent upon the completion of a background task.

In this example, a [Selection Dialog](#) containing three push button controls labeled **Install Now**, **Next**, and **Disk Cost** is displayed to the user during the installation process. However, the installer also carries out a disk space costing task in the background while displaying this dialog box. The author wishes to protect these buttons from activation and wants a "Please wait" message box to pop up if the user clicks any of the buttons before the costing has completed. The author also wants this message box to contain a **Cancel** button and to disappear as soon as the background task is finished.

## ► To display a dialog box asking the user to wait while background disk costing is completed

1. Use the authoring capabilities of the installer to add a new modal dialog box, named **WaitForCosting**, into the [Dialog table](#). The dialog box should display a text string that says "Please wait while disk space costing is completed."
2. Add a single push button control to this dialog box, labeled **Cancel**, by authoring it into the [Control table](#).
3. Link the **Cancel** push button to a ControlEvent that closes the **WaitForCosting** dialog box by authoring an [EndDialog ControlEvent](#) into the [ControlEvent table](#). Set the argument of the EndDialog Control event to be Exit.
4. Link a [SpawnWaitDialog ControlEvent](#) to the existing **Install Now**, **Next**, and **Disk Cost** push button controls displayed in the

[Selection Dialog](#) box. Set the argument of this ControlEvent in the ControlEvent table to be the **WaitForCosting** dialog box and set the expression in the Condition column of the table to be:

**CostingComplete** =1.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Adding Text Stored in a Property

The example described in the section titled [Authoring a Conditional "Please Wait . . ." Message Box](#) displays a dialog box with text that reads: "Please wait while disk space costing is completed." This could be done simply by putting a [Text Control](#) on the dialog box and entering the text string into the Text column of the [Control table](#). In this case, the information about the font style must be embedded in the string. The author must set the font and font style by prefixing the character string with `{style}`. Where *style* is a font style identifier listed in the TextStyle column of the [TextStyle table](#). This method of adding text is illustrated several times in [An Installation Example](#).

An author of a user interface can also store text in a property. The following example illustrates this and shows how ControlEvents can be used to display alternative text strings.

The objective in this example is again to put up a **WaitForCosting** dialog box while a background task is running. The difference with the new scenario is that if the user cancels the **WaitForCosting** dialog box and then tries to activate the control before the background task has finished a second time, the **WaitForCosting** box reappears displaying an alternative message: "Disk space costing is still running. You can continue to wait or return to the main selection box to exit this sequence."

## ► To display a "Please Wait" dialog box that displays alternative messages

1. Begin by adding a conditional **WaitForCosting** dialog box to a Selection dialog box as described in [Authoring a Conditional "Please Wait . . ." Message Box](#).
2. Put a [Text Control](#) on the **WaitForCosting** dialog box by authoring a record in the [Control table](#). Enter the identifier of the **WaitForCosting** dialog box into the Dialog\_ column. Enter the identifier of the Text control into the Control column. Specify the type of control as Text in the Type column.
3. Specify the [Position control attribute](#) for the text control by

entering the horizontal and vertical coordinates of the control's upper left corner in the X and Y columns of the [Control table](#). Use pixels as distance units.

4. Specify the width and height of the text control by entering these dimensions into the Width and Height columns of the [Control table](#). Use pixels as length units.
5. The Property and Control\_Next columns of the Control table do not affect Text controls and can be left blank in this case.
6. Specify the control attributes for the Text control that are associated with bit flags. Add the individual bit values together and enter the total into the Attributes column of the Control table. These are the [Visible](#), [Sunken](#), [Enabled](#), [Transparent](#), [NoWrap](#), and [NoPrefix](#) control attributes. The combination of bits that display a text control on an opaque background, with wrapping text is 0, therefore enter 0 or leave the Attributes column blank.
7. The Text column of the Control table can be left blank. The Text control displays the text string that is the value of the [Text](#) control attribute. The method for setting this attribute is described in subsequent steps of this procedure.
8. Add a record in the [Property table](#) to define the FirstMessage message property. This property is a string containing the font style and text for the first message. Enter the name FirstMessage in the Property column. In the Value column, enter the string: "{WaitStyle}Please wait while disk space costing is completed." Where WaitStyle is an identifier for one of the font styles listed in the TextStyle column of the [TextStyle table](#).
9. Add a record in the [Property table](#) to define the SecondMessage message property. This property is a string containing the font style and text for the second message. Enter the name SecondMessage in the Property column. In the Value column enter the string: "{WaitStyle}Disk space costing is still running.

You can continue to wait or return to the main selection box to exit this sequence."

10. Add a record in the [Property table](#) to define the WaitMessage message property. This property is a string containing the font style and text for the message displayed in the **WaitForCosting** dialog box if the user tries to activate a push button before costing is completed. Enter the name WaitMessage in the Property column. In the Value column of the Property table enter: FirstMessage.
11. Add a [SetProperty ControlEvent](#) to the [ControlEvent table](#) that initializes WaitMessage to FirstMessage each time a **New Selection** dialog box opens. Enter the identifier for the dialog box that comes just before the Selection dialog in the dialog box sequence into the Dialog\_ column. Enter the identifier for the control on this dialog box used to open the Selection dialog box into the Control\_ column. Enter [WaitMessage] into the Event column. Enter [FirstMessage] into the Argument column. Enter 1 into the Condition column and leave the Ordering column blank.
12. Add a [SetProperty ControlEvent](#) to the [ControlEvent table](#) that sets Waitmessage to SecondMessage if the user closes the **WaitForCosting** dialog box before disk space costing has completed. Enter the identifier for the **WaitForCosting** dialog box into the Dialog\_ column. Enter the identifier for the Text control into the Control\_ column. Enter [WaitMessage] into the Event column. Enter [SecondMessage] into the Argument column. Enter NOT [CostingComplete](#) into the Condition column and leave the Ordering column blank.
13. The following step links the Text control attribute to the ControlEvent that spawns the **WaitForCosting** dialog box. This causes the installer to pass the value of the WaitMessage property to the Text control attribute each time the user opens a

**WaitForCosting** dialog box.

14. Subscribe the Text control attribute of the Text control to the [SpawnWaitDialog ControlEvent](#) that opens the **WaitForCosting** dialog box by adding a record to the [EventMapping table](#). Enter the identifier for the **WaitForCosting** dialog box in the Dialog\_ column. Enter the identifier for the Text control into the Control\_ column. Enter SpawnWaitDialog into the Event column. Enter Text, the identifier for the Text control attribute, into the Attribute column of the EventMapping table.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Previewing the User Interface

The installer stores all information about the [user interface](#) in the tables of the installation database. To facilitate the design of the UI the installer also provides a preview mode for viewing this information. The following procedure describes how to enable the UI preview mode and display the current appearance of dialog boxes and billboards.

## ▶ To view the user interface in the preview mode

1. Enable the preview mode by calling the [MsiEnableUIPreview](#) function.
2. Display the particular dialog boxes by calling the [MsiPreviewDialog](#) function.
3. Display particular billboards by calling the [MsiPreviewBillboard](#) function.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Monitoring an Installation Using MsiSetExternalUIRecord

Package authors can monitor internal Windows Installer messages through the creation of an executable application that contains both a record-based callback handler to receive the messages and functionality to initiate an installation.

The record-based handler in the following example conforms to the [INSTALLUI\\_HANDLER\\_RECORD](#) prototype, and a pointer to this callback handler is passed to [MsiSetExternalUIRecord](#) function.

```
//
// ExternalUIRecord.cpp : Defines the entry point for the c
//
#include <tchar.h>
#include <windows.h>
#include <stdio.h>
#include <msi.h>
#include <msiquery.h>
#pragma comment(lib, "msi.lib")
#pragma comment(lib, "user32.lib")
#ifdef _WIN32_MSI
#define _WIN32_MSI 310
#endif

void Usage(LPCTSTR szProgram)
{
    _tprintf(TEXT("Usage: %s <path to .MSI package>\n"), szProg
    return;
}

// In the following snippet, the hRecord parameter is used
// to format the displayed message. However, because hReco
// a handle to a MSI record, the MSI API could be called to
// obtain information. The external UI handler should not c
// hRecord because the handle belongs to the Windows Instal

INT CALLBACK UIHandler(LPVOID pvContext, UINT uiMessageType
{
```

```

INSTALLMESSAGE iMessage = (INSTALLMESSAGE)(0xFF000000 & uiM
BOOL bMessage = (INSTALLMESSAGE_ERROR == iMessage);
BOOL bFatalExit = (INSTALLMESSAGE_FATALEXIT == iMessage);
BOOL bWarning = (INSTALLMESSAGE_WARNING == iMessage);

if ((bMessage || bFatalExit || bWarning) && hRecord && pvCo
{
TCHAR rgchMessage[1024];
DWORD dwMessage = sizeof(rgchMessage)/sizeof(TCHAR);
LPTSTR pszMessage = rgchMessage;

UINT uiRet = MsiFormatRecord(*(MSIHANDLE*)pvContext,
                             hRecord, pszMessage, &dwMe
if (ERROR_MORE_DATA == uiRet)
{
// Allocate a buffer to hold the string and terminating nul
pszMessage = new TCHAR[++dwMessage];
if (!pszMessage)
{
// no memory, return an error
return -1;
}
uiRet = MsiFormatRecord(*(MSIHANDLE*)pvContext,
                         hRecord, pszMessage, &dwMe
if (ERROR_SUCCESS != uiRet)
{
// Return an error if an unexpected error occurs.
if (rgchMessage != pszMessage)
    delete [] pszMessage;
return -1;
}
}
TCHAR rgchFatalError[] = TEXT("Fatal Error");
TCHAR rgchError[] = TEXT("Error");
TCHAR rgchWarning[] = TEXT("Warning");
TCHAR* pszCaption = NULL;

if (INSTALLMESSAGE_ERROR == iMessage)
    pszCaption = rgchError;
else if (INSTALLMESSAGE_FATALEXIT == iMessage)
    pszCaption = rgchFatalError;
else if (INSTALLMESSAGE_WARNING == iMessage)

```

```

        pszCaption = rgchWarning;

int iResponse = MessageBox(NULL, pszMessage,
                           pszCaption, uiMessageType & 0x00FF);

if (rgchMessage != pszMessage)
    delete [] pszMessage;
return iResponse;
}

// Signal that the handler handled this message.
return IDOK;
}

// A console application that takes the path to a package,
// installs the package silently, and uses a record based
// external UI handler to display any error.

int _tmain(int argc, _TCHAR* argv[])
{
    UINT uiRet = ERROR_SUCCESS;
    MSIHANDLE hProduct = NULL;

    if (2 != argc)
    {
        // Incorrect command line used.
        Usage(argv[0]);
        uiRet = 1;
        goto End;
    }

    // Because this example won't restore the UI level,
    // ignore return value.
    MsiSetInternalUI(INSTALLUILEVEL_NONE, /* pWnd */ NULL);

    uiRet = MsiOpenPackage(argv[1], &hProduct);
    if (ERROR_SUCCESS != uiRet)
    {
        _tprintf(TEXT("ERROR: Failed to open %s database.\n"), argv[1]);
        uiRet = 1;
        goto End;
    }
}

```

```

// set the external UI handler
DWORD dwMessageFilter = INSTALLLOGMODE_FATALEXIT
    | INSTALLLOGMODE_ERROR | INSTALLLOGMODE_WARNING;

//Pass the pointer to hProduct as the pvContext argument,
// so that the record based external UI handler
// can use it to format the records we send to it.

uiRet = MsiSetExternalUIRecord(UIHandler,
    dwMessageFilter, (LPVOID)&hProduct, /* ppuiPr

if (ERROR_SUCCESS != uiRet)
{
    _tprintf(TEXT("ERROR: Failed to set the external UI
    uiRet = 1;
    goto End;
}

// Do the default action and install the product.
uiRet = MsiDoAction(hProduct, TEXT(""));

End:
// Close opened resources.
if (hProduct)
    MsiCloseHandle(hProduct);

return ERROR_SUCCESS == uiRet ? 0 : 1;
}
//

```

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Monitoring an Installation Using MsiSetExternalUI

Package authors can monitor internal Windows Installer messages through the creation of an executable application that contains both a callback handler to receive the messages and functionality to initiate an installation.

The callback handler conforms to the `INSTALLUI_HANDLER` prototype, and a pointer to this callback handler is passed to [MsiSetExternalUI](#). Once the installation has been initiated by a call to [MsiInstallProduct](#), installation messages are trapped by the callback handler and the package author can selectively display or ignore any or all of these messages.

For more information, see [Handling Progress Messages Using MsiSetExternalUI](#), [Returning Values from an External User Interface Handler](#), and [Parsing Windows Installer Messages](#).

For more information about using a record-based external handler, see [Monitoring an Installation Using MsiSetExternalUIRecord](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Handling Progress Messages Using MsiSetExternalUI

The following sample demonstrates how to code a simple callback handler to receive Windows Installer progress messages during an installation.

**Note** When using [MsiSetExternalUI](#) with a message type of `INSTALLMESSAGE_FILESINUSE`, the message sent to the external UI handler function does not contain any information about files in use or window titles used by the [FilesInUse](#) dialog box. You should use [MsiSetExternalUIRecord](#) to obtain information.

```
#include <windows.h>

// Globals
//
//   common data information fields
int g_rgiField[3]; //array of fields to handle INSTALLOGMO
WORD g_wLANGID = LANG_NEUTRAL; // initialize to neutral lan
//
//   progress information fields
int iField[4]; //array of record fields to handle INSTALLO
int g_iProgressTotal = 0; // total ticks on progress bar
int g_iProgress = 0; // amount of progress
int iCurPos = 0;
BOOL bFirstTime = TRUE;
BOOL g_bForwardProgress = TRUE; //TRUE if the progress bar
BOOL g_bScriptInProgress = FALSE;
BOOL g_bEnableActionData; //TRUE if INSTALLOGMODE_ACTIONDAT
BOOL g_bCancelInstall = FALSE; //Should be set to TRUE if t

// In the following snippet, note that the internal user
// interface level is set to INSTALLLEVEL_NONE. If the inte
// user interface level is set to anything other than
// INSTALLUILEVEL_NONE, the user interface level is
// INSTALLUILEVEL_BASIC by default and the installer only
// displays an initial dialog. If the authored wizard
// sequence of the package is to be displayed, the user
// interface level should be set to INSTALLUILEVEL_FULL.
```

```

// If the external user interface handler is to have full
// control of the installation user interface, the user
// interface level must be set to INSTALLUILEVEL_NONE.

// Because an external UI handler cannot handle the
// INSTALLMESSAGE_RESOLVESOURCE message,
// Windows Installer allows a UI level,INSTALLUILEVEL_SOURCE
// that will allow an external UI handler to have full control
// permitting an install to resolve the source

MsiSetInternalUI(INSTALLUILEVEL(INSTALLUILEVEL_NONE|INSTALLUILEVEL_SOURCE));

MsiSetExternalUI (TestMyBasicUIHandler,
    INSTALLLOGMODE_PROGRESS|INSTALLLOGMODE_FATALEXIT|INSTALLLOGMODE_WARNING|INSTALLLOGMODE_ERROR|INSTALLLOGMODE_RESOLVESOURCE|INSTALLLOGMODE_ACTIONSTART|INSTALLLOGMODE_COMMONDATA|INSTALLLOGMODE_TERMINATE|INSTALLLOGMODE_VERBOSE|INSTALLLOGMODE_PROGRESS_UI|INSTALLLOGMODE_WARNING_UI|INSTALLLOGMODE_ERROR_UI|INSTALLLOGMODE_RESOLVESOURCE_UI|INSTALLLOGMODE_ACTIONSTART_UI|INSTALLLOGMODE_COMMONDATA_UI|INSTALLLOGMODE_TERMINATE_UI|INSTALLLOGMODE_VERBOSE_UI|INSTALLLOGMODE_PROGRESS_UI_CUSTOM|INSTALLLOGMODE_WARNING_UI_CUSTOM|INSTALLLOGMODE_ERROR_UI_CUSTOM|INSTALLLOGMODE_RESOLVESOURCE_UI_CUSTOM|INSTALLLOGMODE_ACTIONSTART_UI_CUSTOM|INSTALLLOGMODE_COMMONDATA_UI_CUSTOM|INSTALLLOGMODE_TERMINATE_UI_CUSTOM|INSTALLLOGMODE_VERBOSE_UI_CUSTOM|INSTALLLOGMODE_PROGRESS_UI_CUSTOM_TEXT("TEST"));

MsiInstallProduct(/*full path to package*/,NULL);

//
// FUNCTION: TestMyBasicUIHandler()
//
// PURPOSE: Demonstrates usage of an external user interface handler.
//
// COMMENTS:
//
int _stdcall TestMyBasicUIHandler(LPVOID pvContext, UINT iMsg, WPARAM wParam, LPARAM lParam)
{
// File costing is skipped when applying Patch(es) and INSTALLMESSAGE_RESOLVESOURCE
// Workaround: Set INSTALLUILEVEL to anything but NONE only
    if (bFirstTime == TRUE)
    {
        UINT r1 = MsiSetInternalUI(INSTALLUILEVEL_SOURCE);
        bFirstTime = FALSE;
    }

    if (!szMessage)

```

```

    return 0;

INSTALLMESSAGE mt;
UINT uiFlags;

mt = (INSTALLMESSAGE)(0xFF000000 & (UINT)iMessageType);
uiFlags = 0x00FFFFFF & iMessageType;

switch (mt)
{
    //Premature termination
case INSTALLMESSAGE_FATALEXIT:
    /* Get fatal error message here and display it*/
    return MessageBox(0, szMessage, TEXT("FatalError"));

case INSTALLMESSAGE_ERROR:
    {
        /* Get error message here and display it*/
        // language and caption can be obtained from context menu
        MessageBeep(uiFlags & MB_ICONMASK);
        return MessageBoxEx(0, szMessage, TEXT("Error"), MB_OK);
    }

case INSTALLMESSAGE_WARNING:
    /* Get warning message here and display it */
    return MessageBox(0, szMessage, TEXT("Warning"), MB_OK);

case INSTALLMESSAGE_USER:
    /* Get user message here */
    // parse uiFlags to get Message Box Styles Flag and return IDOK;

case INSTALLMESSAGE_INFO:
    return IDOK;

case INSTALLMESSAGE_FILESINUSE:
    /* Display FilesInUse dialog */
    // parse the message text to provide the names of the
    // applications that the user can close so that the
    // files are no longer in use.
    return 0;

case INSTALLMESSAGE_RESOLVESOURCE:

```

```

    /* ALWAYS return 0 for ResolveSource */
    return 0;

case INSTALLMESSAGE_OUTOFDISKSPACE:
    /* Get user message here */
    return IDOK;

case INSTALLMESSAGE_ACTIONSTART:
    /* New action started, any action data is sent by t
    g_bEnableActionData = FALSE;
    return IDOK;

case INSTALLMESSAGE_ACTIONDATA:
    // only act if progress total has been initialized
    if (0 == g_iProgressTotal)
        return IDOK;
    SetDlgItemText(/*handle to your dialog*/, /*identifi
    if(g_bEnableActionData)
    {
        SendMessage(/*handle to your progress control*/,
    }
    return IDOK;

case INSTALLMESSAGE_PROGRESS:
    {
        if(ParseProgressString(const_cast<LPSTR>(szMess
        {
            // all fields off by 1 due to c array notat
            switch(iField[0])
            {
            case 0: // Reset progress bar
                {
                    //field 1 = 0, field 2 = total numk

                    /* get total number of ticks in pro
                    g_iProgressTotal = iField[1];

                    /* determine direction */
                    if (iField[2] == 0)
                        g_bForwardProgress = TRUE;
                    else // iField[2] == 1
                        g_bForwardProgress = FALSE;

```

```

        /* get current position of progress
        // if Forward direction, current p
        // if Backward direction, current p
        g_iProgress = g_bForwardProgress ?
        SendMessage(/*handle to your progr

        // if g_bScriptInProgress, finish p
        SendMessage(/*handle to your progr

        iCurPos = 0;

        /* determine new state */
        // if new state = 1 (script in prog
        // new state = 1 means the total #
        g_bScriptInProgress = (iField[3] ==

        break;
    }
case 1: // ActionInfo
    {
        //field 1 = 1, field 2 will contain
        //ignore if field 3 is zero
        if(iField[2])
        {
            // movement direction determine
            SendMessage(/*handle to your pr
            g_bEnableActionData = TRUE;
        }
        else
        {
            g_bEnableActionData = FALSE;
        }

        break;
    }
case 2: //ProgressReport
    {
        // only act if progress total has k
        if (0 == g_iProgressTotal)
            break;
    }

```

```

        iCurPos += iField[1];

        //field 1 = 2, field 2 will contain
        // movement direction determined by
        SendMessage(/*handle to your prog...

        break;
    }
    case 3: // ProgressAddition - fall through
    default:
        {
            break;
        }
    }
}

if(g_bCancelInstall == TRUE)
{
    return IDCANCEL;
}
else
    return IDOK;
}

case INSTALLMESSAGE_COMMONDATA:
{
    if (ParseCommonDataString(const_cast<LPSTR>(szM
    {
        // all fields off by 1 due to c array notat
        switch (g_rgiField[0])
        {
            case 0:
                // field 1 = 0, field 2 = LANGID, field
                g_wLANGID = g_rgiField[1];
                break;
            case 1:
                // field 1 = 1, field 2 = CAPTION
                /* you could use this as the caption fo
                break;
            case 2:
                // field 1 = 2, field 2 = 0 (hide cancell

```

```

        ShowWindow(/*handle to cancel button co
        break;
    default:
        break;
    }
}
return IDOK;
}

// this message is received prior to internal UI initializ
case INSTALLMESSAGE_INITIALIZE:
    return IDOK;

// Sent after UI termination, no string data
case INSTALLMESSAGE_TERMINATE:
    return IDOK;

//Sent prior to display of authored dialog or wizard
case INSTALLMESSAGE_SHOWDIALOG:
    return IDOK;

default:
    return 0;
}
}

//
// FUNCTION: ParseCommonDataString(LPSTR sz)
//
// PURPOSE:  Parses the common data message sent to the IM
//
// COMMENTS: Ignores the 3rd field and the caption common
//
BOOL ParseCommonDataString(LPSTR sz)
{
    char *pch = sz;
    if (0 == *pch)
        return FALSE; // no msg

    while (*pch != 0)
    {
        char chField = *pch++;

```

```

    pch++; // for ':'
    pch++; // for sp
    switch (chField)
    {
    case '1': // field 1
        {
            // common data message type
            g_rgiField[0] = *pch++ - '0';
            if (g_rgiField[0] == 1)
                return FALSE; // we are ignoring caption
            break;
        }
    case '2': // field 2
        {
            // because we are ignoring caption msg, the
            g_rgiField[1] = FGetInteger(pch);
            return TRUE; // done processing
        }
    default: // unknown field
        {
            return FALSE;
        }
    }
    pch++; // for space (' ') between fields
}

return TRUE;
}

//
// FUNCTION: FGetInteger(char*& pch)
//
// PURPOSE: Converts the string (from current pos. to next)
//           to an integer.
//
// COMMENTS: Assumes correct syntax. Ptr is updated to next
//           or null terminator.
//
int FGetInteger(char*& rpch)
{
    char* pchPrev = rpch;
    while (*rpch && *rpch != ' ')

```

```

        rpch++;
        *rpch = '\\0';
        int i = atoi(pchPrev);
        return i;
    }

//
// FUNCTION: ParseProgressString(LPSTR sz)
//
// PURPOSE:  Parses the progress data message sent to the
//
// COMMENTS: Assumes correct syntax.
//
BOOL ParseProgressString(LPSTR sz)
{
    char *pch = sz;
    if (0 == *pch)
        return FALSE; // no msg

    while (*pch != 0)
    {
        char chField = *pch++;
        pch++; // for ':'
        pch++; // for sp
        switch (chField)
        {
            case '1': // field 1
                {
                    // progress message type
                    if (0 == isdigit(*pch))
                        return FALSE; // blank record
                    iField[0] = *pch++ - '0';
                    break;
                }
            case '2': // field 2
                {
                    iField[1] = FGetInteger(pch);
                    if (iField[0] == 2 || iField[0] == 3)
                        return TRUE; // done processing
                    break;
                }
            case '3': // field 3

```

```
        {
            iField[2] = FGetInteger(pch);
            if (iField[0] == 1)
                return TRUE; // done processing
            break;
        }
    case '4': // field 4
        {
            iField[3] = FGetInteger(pch);
            return TRUE; // done processing
        }
    default: // unknown field
        {
            return FALSE;
        }
    }
    pch++; // for space (' ') between fields
}

return TRUE;
}
```

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Returning Values from an External User Interface Handler

An external user interface (UI) handler can return any number of values to Windows Installer depending on the button type provided in the message type parameter the installer passes to the handler.

The external UI handler can return the values  $-1$  and  $0$  at any time because these are not related to the button types. A return value of  $-1$  indicates that an internal error occurred in the external UI handler. A return value of  $0$  indicates that the external UI handler has not handled the installer message and the installer must handle the message instead.

For messages that do not include a button type, such as `INSTALLMESSAGE_ACTIONDATA` and `INSTALLMESSAGE_PROGRESS`, returning `IDCANCEL` cancels the installation. Returning `IDOK` notifies the installer that the message was handled by the external UI handler.

The remaining return values, as described below, are directly related to the button types that are included with the message type.

External UI return value	Meaning
IDOK	The <b>OK</b> button was pressed by the user. The message information was understood.
IDCANCEL	The <b>CANCEL</b> button was pressed. Cancel the installation.
IDABORT	The <b>ABORT</b> button was pressed. Abort the installation.
IDRETRY	The <b>RETRY</b> button was pressed. Try the action again.
IDIGNORE	The <b>IGNORE</b> button was pressed. Ignore the error and continue.
IDYES	The <b>YES</b> button was pressed. The affirmative response, continue with current sequence of events..
IDNO	The <b>NO</b> button was pressed. The negative response, do not continue with current sequence of events.

For example, if the external UI handler is sent a message with the MB\_ABORTRETRYIGNORE message box styles flag, the external UI handler can return one of the following values:

- -1 (error in external UI handler)
- 0 (no action taken in external UI handler, let Windows Installer handle it)
- IDABORT (**ABORT** button pressed)
- IDRETRY (**RETRY** button pressed)
- IDIGNORE (**IGNORE** button pressed)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Parsing Windows Installer Messages

An external UI handler can process the list of installer messages specified by the *dwMessagedFilter* parameter of the [MsiSetExternalUI](#) function. Some of these messages contain strings that can be used directly, and other messages may need to be parsed and processed by the external UI handler to be useful. Your external UI handler may only need to monitor Windows Installer messages without performing any operation that affects the installation.

The following Windows Installer messages contain strings that can be displayed by a dialog box and need no additional processing. These messages contain a list of buttons and icons that are to be displayed by a dialog box. You can use the `MB_ICONMASK`, `MB_DEFMASK`, and `MB_TYPEMASK` values to specify icons and buttons.

`INSTALLMESSAGE_FATALEXIT`

Premature termination of installation has occurred.

`INSTALLMESSAGE_ERROR`

Formatted error message.

`INSTALLMESSAGE_WARNING`

Formatted warning message.

`INSTALLMESSAGE_INFO`

Formatted log message.

`INSTALLMESSAGE_USER`

Formatted user message.

`INSTALLMESSAGE_OUTOFDISKSPACE`

Formatted message indicating an out of disk space condition

The external user handler can use the following Windows Installer messages to monitor a sequence of the Windows Installer UI. The installer sends these messages at the start of a Windows Installer UI sequence, as each dialog box is displayed, and at the end of the UI sequence. No processing is required to use these messages.

`INSTALLMESSAGE_TERMINATE`

This message indicates the end of the UI sequence. The string is a

null string.

#### INSTALLMESSAGE\_INITIALIZE

This message indicates that the UI sequence has started. The string is a null string.

#### INSTALLMESSAGE\_SHOWDIALOG

The string contains the name of the current dialog box.

The following Windows Installer messages require additional processing by the external UI handler.

#### INSTALLMESSAGE\_RESOLVESOURCE

The external user interface handler must return 0 and allow Windows Installer to handle the message. The external user interface handler can monitor for this message, but it should not perform any action that affects the installation .

#### INSTALLMESSAGE\_FILESINUSE

The external UI should display a [FilesInUse dialog](#) in response to this message.

#### INSTALLMESSAGE\_RMFILESINUSE

The external UI should display a [MsiRMFilesInUse dialog](#) in response to this message. Available beginning with Windows Installer version 4.0. For more information about this message see [Using Restart Manager with an External UI](#).

#### INSTALLMESSAGE\_ACTIONSTART

This message gives information about the current action. The format is Action [1]: [2]. [3], where a colon is used to separate Field 1 and Field 2 and a period is used to separate Field 2 and Field 3. Field [1] contains the time the action was started using the [Time](#) property format. Field [2] contains the action's name from the sequence table. Field [3] gives the action's description from the [ActionText table](#) or from the [MsiProcessMessage](#) function.

#### INSTALLMESSAGE\_ACTIONDATA

The format of this string is specified by the [Template](#) value provided in the [ActionText table](#) or by the [MsiProcessMessage](#) function.

There can be an unlimited number of

INSTALLMESSAGE\_ACTIONDATA messages after the

INSTALLMESSAGE\_ACTIONSTART message.

## INSTALLMESSAGE\_COMMONDATA

This message has three subtypes: Language, Caption, and CancelShow. The string can have three fields delimited by a number followed by a colon. Not all fields are required. The message can be a NULL or empty ("" ) string.

### Language

Field 1 contains the value 0 to indicate this string contains language information. Field 2 contains a [Language](#) value that is a numeric language identifier (LANGID.) Field 3 is a value that represents an ANSI code page.

### Caption

Field 1 contains the value 1 to indicate this string contains the text of a caption or title. Field 2 contains text that an external UI handler can use as a caption of title for a dialog box. Field 3 is NULL or an empty ("" ) string. Field 3 can be absent from a the Caption message.

### CancelShow

Field 1 contains the value 2 to indicate this string contains information about whether to display the cancel button. If the cancel button should be hidden, Field 2 contains the value 0. If the cancel button should be visible, Field 2 contains the value 1.

## INSTALLMESSAGE\_PROGRESS

This message has four subtypes: Reset, ActionInfo, ProgressReport, and ProgressAddition. The external handler should not act upon any of these messages until the first a Reset progress message is received. This provides an estimate of the total number of ticks for the progress bar.

### Reset

Field 1 contains the value 0 to indicate a reset of the progress bar. Field 2 contains the total number of ticks in the progress bar. Field 3 contains the value 0 for forward progress bar motion. Field 3 contains the value 1 for backward progress bar motion. The value 0 in Field 4 means the installation is in progress and the time remaining may be calculated. The value

1 in Field 4 means script is being run and a "Please wait ..." message can be displayed. The estimate of the total number of ticks is an approximation and may be inaccurate.

#### ActionInfo

Field 1 contains the value 1 to indicate this string contains action information. Field 2 contains the number of ticks the progress bar moves for each ActionData message sent by the current action. If Field 3 contains the value 0, ignore Field 2. If Field 3 contains the value 1, increment the progress bar by the number of ticks in Field 2 for each ActionData message sent by the current action. Field 4 is unused.

#### ProgressReport

Field 1 contains the value 2 to indicate this string contains progress information. Field 2 contains the number of ticks the progress bar has moved. Field 3 is unused. Field 4 is unused.

#### ProgressAddition

Field 1 contains the value 3 to indicate that an action can add ticks the progress bar. Field 2 contains the number of ticks to add to total expected count of progress ticks. Field 3 is unused. Field 4 is unused.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Sending Messages to Windows Installer Using MsiProcessMessage

The messages sent using [MsiProcessMessage](#) are the same messages that are received by the [INSTALLUI\\_HANDLER](#) callback function if [MsiSetExternalUI](#) was called. Otherwise, Windows Installer handles the messages. For details, see [Parsing Windows Installer Messages](#).

For example, to send an `INSTALLMESSAGE_ERROR` message with the `MB_ICONWARNING` icon and the `MB_ABORTRETRYCANCEL` buttons:

```
PMSIHANDLE hInstall;  
PMSIHANDLE hRec;  
MsiProcessMessage(hInstall, INSTALLMESSAGE_ERROR, MB_ICONWARNING, MB_ABORTRETRYCANCEL, hRec);
```

Where *hInstall* is the handle to the installation, provided to a custom action or the *hProduct* handle from [MsiOpenProduct](#) or [MsiOpenPackage](#), and *hRec* is the record containing the error information to format. For information on how formatting is performed, see [MsiFormatRecord](#).

By default, if an `INSTALLMESSAGE_ERROR` or `INSTALLMESSAGE_FATALEXIT` message is sent without specifying button type or icon types, `MB_OK`, no icon, and `MB_DEFBUTTON1` are used.

Windows Installer does not label the **ABORT** button with the "Abort" string when displaying a `MessageBox` with the `MB_ABORTRETRYIGNORE` button specification, instead it labels the button with the "Cancel" string. All error messages refrain from using the word "Abort" and instead use the word "Cancel".

The *hRecord* parameter of the [MsiProcessMessage](#) function depends upon the message type sent to the [MsiProcessMessage](#). The following list details the requirements of the record in relation to the message type:

```
INSTALLMESSAGE_FATALEXIT  
INSTALLMESSAGE_INFO
```

## INSTALLMESSAGE\_OUTOFDISKSPACE

Field	Description
0	Template for the formatting of the resultant string. See <a href="#">MsiFormatRecord</a> for more information. The fields of the record are referenced using [1] for field 1, [2] for field 2, etc.
1 through <i>n</i>	All subsequent fields are directly related to the fields referenced by the template in field 0.

If field 0 is null, the string received by the UI handler is formatted as: 1: [data from field 1] 2: [data from field 2] meaning that for each field of the record, the string contains the field number followed by the data stored in the field.

Information messages from [MsiProcessMessage](#) are logged when [MsiEnableLog](#), the `/!` [command line option](#), or the [logging policy](#) specify `!` or `INSTALLLOGMODE_INFO`.

INSTALLMESSAGE\_ERROR  
INSTALLMESSAGE\_WARNING  
INSTALLMESSAGE\_USER

To use a message from the Error table.

Field	Description
0	Must be null.
1	Message number in the <a href="#">Error table</a> .
2 through <i>n</i>	Related to the specified message in the Error table.

For example.

Field	Type	Data
0	string	null
1	int	1304

2	string	Myfile.txt
---	--------	------------

The resulting message received from the UI handler is:

Error 1304. Error writing to file: Myfile.txt. Verify that you have access to that directory.

If field 0 is not null, the message from the error table is overridden. Instead, the field 0 template determines the format of the message.

The message may also specify the buttons, including the default button, and the icon for use with the message as mentioned above. The button and icon types are listed in [INSTALLUI\\_HANDLER](#).

#### INSTALLMESSAGE\_COMMONDATA

This message is sent to enable or disable the **Cancel** button in a progress dialog box.

Field	Description
0	Unused.
1	2 refers to the <b>Cancel</b> button.
2	A value of 1 indicates the <b>Cancel</b> button should be visible. A value of 0 indicates the <b>Cancel</b> button should be invisible.

For example, to disable or hide the **Cancel** button, the record would appear as follows.

Field	Type	Data
0	string	null
1	int	2
2	int	0

#### INSTALLMESSAGE\_ACTIONSTART

## INSTALLMESSAGE\_ACTIONDATA

The INSTALLMESSAGE\_ACTIONSTART record determines the format of the ActionData record.

Field	Description
0	null
1	Action name.
2	Action description.
3	Action template. This is used for the ActionData whose message is being formatted according to this template.

Do not reference field 0 in the Action template message.

The INSTALLMESSAGE\_ACTIONDATA record is formatted as follows.

Field	Description
0	null
1 through <i>n</i>	Dependent upon field 3 of the corresponding INSTALLMESSAGE_ACTIONSTART message or template specified in <a href="#">ActionText table</a> .

For example, the INSTALLMESSAGE\_ACTIONSTART record.

Field	Type	Data
0	string	null
1	string	MyAction
2	string	This is the description of "MyAction"
3	string	MyAction template: field1 data is [1]. field 2 data is [2].

The template for INSTALLMESSAGE\_ACTIONSTART (field 3) references fields 1 and 2, the INSTALLMESSAGE\_ACTIONDATA record should have 2 fields containing the warranted data. The fields could be

either string or integer fields.

INSTALLMESSAGE\_ACTIONDATA record.

Field	Type	Data
0	string	null
1	int	2
2	string	ActionData for MyAction

INSTALLMESSAGE\_FILESINUSE

The FILESINUSE record is a variable length record.

Field	Description
0	This field can be null. For an installation using a basic UI, this field may specify static text for display in the <a href="#">ListBox control</a> of the <a href="#">FilesInUse dialog</a> . For an installation using a full UI, this field has no effect because the text is specified by the authoring of the custom FilesInUse dialog box.
1	Name of the file in use.
2	This field identifies the process holding the file in use.  <b>Windows Installer version 4.0:</b> The process id (PID) of the process, or the title of the window for the process.  <b>Windows Installer version 3.1 and earlier:</b> This field must be the process id (PID) of the process.

For example, to send a FilesInUse message showing two files in use, red.exe and blue.exe, the record has four fields plus the 0 field. The format of the record would be as shown in the following table. This example requires Windows Installer version 4.0.

**Windows Installer version 3.1 and earlier:** Fields 2 and 4 in the following example must contain the PIDs of the processes holding

red.exe and blue.exe in use.

Field	Description
0	null
1	Red.exe
2	Red Window Title
3	Blue.exe
4	Blue Window Title

**Note** On Windows Installer version 4.0, if the PID passed from the service does not have a window title, such as a system tray application, the file is not be displayed and the verbose log contains the following messages.

```
File In Use: -<FileName>- Window could not be found. Process  
No window with title could be found for FilesInUse
```

## INSTALLMESSAGE\_RESOLVESOURCE

The INSTALLMESSAGE\_RESOLVESOURCE record has seven fields. For INSTALLMESSAGE\_RESOLVESOURCE to work correctly, an external UI handler may not handle the INSTALLMESSAGE\_RESOLVESOURCE message. Windows Installer must handle the INSTALLMESSAGE\_RESOLVESOURCE message. That is, the external UI handler returns 0 to indicate "no action taken" when filtering the INSTALLMESSAGE\_RESOLVESOURCE message. The best practice is to avoid sending a RESOLVESOURCE message.

Field	Description
0	null
1	null
2	Package name.
3	Product code.
4	Relative path if known, can be null.

5	0
6	Whether to validate the package code. A value of '1' indicates the package code should be validated. A value of '0' indicates the package should not be validated.
7	Required disk from media table. A value of '0' indicates that any disk is acceptable.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Displaying Billboards on a Modeless Dialog

Billboards can display a sequence of images and text in a dialog during an installation. Typically, billboards are used to create the visual effect of a slide show or animation that informs the user of the progress of an installation.

## ► To display billboards on a modeless dialog

1. Include a record in the [Dialog Table](#) for the modeless dialog box that contains the billboard. After a billboard is displayed, a modeless dialog box returns control to the Installer. This enables the Installer to process messages and update the dialog box and billboard. To create a modeless dialog box, do not set the Modal Dialog Style Bit in the Attributes field of the [Dialog Table](#). The following [Dialog Table](#) record specifies the ActionDialog dialog box.

[Dialog Table](#) (partial)

Dialog_	HCentering	VCentering	Width	Height	Attributes
ActionDialog	50	50	480	240	5

2. Add a record to the [Control Table](#) to specify that the dialog box displays a billboard. The record defines the size and position of the region on the dialog box where the billboard controls listed in the [BBControl Table](#) are to be displayed. The following [Control Table](#) record defines the position and size of the billboard on the ActionDialog dialog box.

[Control Table](#) (partial)

Dialog_	Control	Type	X	Y	Width	Height	Attributes
ActionDialog	Billboard	Billboard	0	110	480	130	1

- 
3. The [Billboard Table](#) lists the billboard controls and specifies when a specific billboard control is displayed. Add a record to the [Billboard Table](#) for each billboard control. The [Billboard Table](#) watches for progress messages sent during an installation. A billboard is displayed only when a progress message is sent by the actions listed in the Action column of the [Billboard Table](#), and only if the feature in the Feature\_ field is selected for installation. After a billboard is displayed, it remains visible until covered by another billboard, or until the dialog box is closed. If multiple billboards are specified for an action, they are displayed one at a time in the order specified by the Ordering field. For example, the following [Billboard Table](#) entries first display the BB1 and then the BB2 [Billboard Controls](#) when the [InstallFiles](#) action is run and the QuickTest feature has been selected to be installed.

[Billboard Table](#) (partial)

<b>Billboard</b>	<b>Feature</b>	<b>Action</b>	<b>Ordering</b>
BB1	QuickTest	InstallFiles	1
BB2	QuickTest	InstallFiles	2

4. The [BBControl Table](#) specifies the controls that belong to the Billboard Controls that are listed in the [Billboard Table](#). The [Text Control](#), [Bitmap Control](#), and [Icon Control](#) are the only types of controls that can go on a billboard. Multiple controls can be placed on each billboard. Enter the name of the billboard into the Billboard\_ field of the [BBControl Table](#) exactly as it appears in the [Billboard Table](#). Each control position is specified as the coordinates of the upper left corner of the control. The coordinate system origin is located

at the upper left corner of the billboard control rather than at a corner of the dialog box. The coordinates are in Installer units, not dialog units. An Installer unit is equal to one-twelfth the height of the 10-point MS Sans Serif font size. The following [BBControl Table](#) records ties controls to billboards.

[BBControl Table](#) (partial)

Billboard	BBControl	Type	X	Y	Width	Height	Attributes	T
BB1	Text	Text	100	30	280	280	3	F E
BB1	Bitmap1	Bitmap	0	0	100	100	3	S
BB1	Bitmap2	Bitmap	380	0	100	100	3	M
BB2	Text	Text	100	30	280	20	3	S E
BB2	Bitmap1	Bitmap	0	0	100	100	3	M
BB2	Bitmap2	Bitmap	380	0	100	100	3	S

- To display a billboard on the ActionDialog dialog box, you must subscribe the billboard control to [SetProgress ControlEvent](#) by adding a record to the [EventMapping Table](#). When the Installer publishes the SetProgress ControlEvent that is specified in the Event column, the Installer sets the control attribute that is specified in the Attribute field. The Event field contains the string identifier (without quotes) of the SetProgress ControlEvent. The Attribute field contains the string identifier (without quotes) of the attribute to be set. The Dialog\_ and Control\_ fields identify the Billboard Control and should match those fields in the [Control Table](#). For example, the following [EventMapping Table](#) subscribes a control to an event.

[EventMapping Table](#) (partial)

---

<b>Dialog_</b>	<b>Control_</b>	<b>Event</b>	<b>Attribute</b>
ActionDialog	Billboard	SetProgress	Progress

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using an Embedded UI

A custom user interface can be embedded within the Windows Installer package.

The DLL file containing the custom UI, and any resource files used by the custom UI, should be listed in the [MsiEmbeddedUI](#) table. For example, this MsiEmbeddedUI table contains a row for the DLL file containing the embedded UI and a row for a bitmap file used by the UI.

MsiEmbeddedUI	FileName	Attributes	MessageFilter	Data
EmbeddedUI	embedui.dll	3	201359327	[Binary Data]
CustomBitmap	custom.bmp	0		[Binary Data]

The custom UI DLL, in this example embedui.dll, should export the user-defined [InitializeEmbeddedUI](#), [EmbeddedUIHandler](#), and [ShutdownEmbeddedUI](#) functions. The following sample code illustrates these functions.

```
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>
#include <msi.h>
#include <msiquery.h>
#include <Aclapi.h>
#include <strsafe.h>
#pragma comment(lib, "msi.lib")

#define cchGUID 38

int __stdcall InitializeEmbeddedUI(MSIHANDLE hInstall,
    LPCWSTR szResourcePath, LPDWORD pdwInternal
{
    // The hInstall handle is only valid within this fu
    // can be used to get or set properties. The handle
    // does not need to be explicitly closed.

    WCHAR szProductCode[cchGUID+1];
```

```

DWORD cchProductCode = ARRAYSIZE(szProductCode);
UINT uiStat = MsiGetProperty(hInstall, L"ProductCode",
                             szProductCode, &cchProductCode);
if (ERROR_SUCCESS != uiStat)
{
    // The installation should fail.
    return ERROR_INSTALL_FAILURE;
}

WCHAR* szReinstall = NULL;
DWORD cchReinstall = 0;
uiStat = MsiGetProperty(hInstall, TEXT("REINSTALL"),
                       szReinstall, &cchReinstall);
if (ERROR_MORE_DATA == uiStat)
{
    ++cchProductCode; // Add 1 for terminating
    szReinstall = new WCHAR[cchReinstall];
    if (szReinstall)
    {
        uiStat = MsiGetProperty(hInstall, L"REINSTALL",
                                szReinstall, &cchReinstall);
    }
}
if (ERROR_SUCCESS != uiStat)
{
    if (szReinstall != NULL)
        delete [] szReinstall;
    // This installation should fail.
    return ERROR_INSTALL_FAILURE;
}

if (INSTALLSTATE_DEFAULT != MsiQueryProductState(szProductCode))
{
    if (INSTALLUILEVEL_BASIC == *pdwInternalUILevel)
    {
        // Insert the custom UI used by basic
    }
    else
    {
        // Insert the custom UI used by full
    }
}

```

```

else if (szReinstall && szReinstall[0])
{
    // Reinstall the UI sequence.
}
else
{
    // This is a maintenance installation. Remove the UI.
    MsiSetProperty(hInstall, TEXT("REMOVE"), TEXT("REMOVE"));
}

if (szProductCode)
    delete [] szReinstall;

// Setting the internal UI level to none specifies
// no authored UI should run.
*pdwInternalUILevel = INSTALLUILEVEL_NONE;
return 0;
}

int __stdcall ShutdownEmbeddedUI()
{
    // ShutdownEmbeddedUI is optional. It can allow the installer
    // to perform any cleanup. After this call, the embedded UI
    // should not receive any additional callbacks.
    return 0;
}

INT __stdcall EmbeddedUIHandler(UINT iMessageType, MSIHANDLE hInstall)
{
    // This function is similar to the MsiSetExternalUIHandler function.
    INSTALLMESSAGE mt;
    UINT uiFlags;

    mt = (INSTALLMESSAGE) (0xFF000000 & (UINT) iMessageType);
    uiFlags = 0x00FFFFFF & iMessageType;

    switch (mt)
    {
    case INSTALLMESSAGE_FATALEXIT:
        {

```

```
        return IDOK;
    }
case INSTALLMESSAGE_ERROR:
    {
        return IDOK;
    }
case INSTALLMESSAGE_WARNING:
    {
        return IDOK;
    }
case INSTALLMESSAGE_FILESIINUSE:
    {
        return IDOK;
    }
case INSTALLMESSAGE_RESOLVESOURCE:
    {
        return IDOK;
    }
case INSTALLMESSAGE_USER:
    {
        return IDOK;
    }
case INSTALLMESSAGE_INFO:
    {
        return IDOK;
    }
case INSTALLMESSAGE_OUTOFDISKSPACE:
    {
        return IDOK;
    }
case INSTALLMESSAGE_ACTIONSTART:
    {
        return IDOK;
    }
case INSTALLMESSAGE_ACTIONDATA:
    {
        return IDOK;
    }
case INSTALLMESSAGE_PROGRESS:
    {
        return IDOK;
    }
}
```

```
case INSTALLMESSAGE_SHOWDIALOG:
    {
        return IDOK;
    }
case INSTALLMESSAGE_COMMONDATA:
    {
        return IDOK;
    }
case INSTALLMESSAGE_INSTALLSTART:
    {
        // This message is sent when the InstallShield
        // Record contains the ProductName
        return IDOK;
    }
case INSTALLMESSAGE_INSTALLEND:
    {
        // This message is sent when the InstallShield
        // Record contains the ProductName
        // and return value of the installation
        return IDOK;
    }

default:
    {
        return 0;
    }
}
}
```

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# User Interface Reference

This section is intended for developers who are writing their own setup programs using the internal user interface of the Windows Installer database. For general information about the installer, see [About Windows Installer](#).

For more information about how to author a user interface, see [Using the User Interface](#).

[Dialog Boxes](#)[Dialog Style Bits](#)

[Controls](#)

[Control Attributes](#)

[ControlEvents](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer Units

A Windows Installer user interface unit is approximately equal to one-twelfth (1/12) the height of the 10-point MS Sans Serif font size. The relative size of dialogs, controls and fonts can change depending upon the font size used in the user interface. To ensure the correct display of the user interface, setup developers should always test their application using the actual font sizes.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Dialog Boxes

Dialog boxes are specified in the Dialog column of the [Dialog table](#). For more information on adding a dialog box or billboard to a user interface, see [Using the User Interface](#).

## Reserved Dialog Box Names

The following dialog box names are reserved by Windows Installer and should not be used for any user-authored custom dialog boxes. The installer requires that these dialog boxes be listed in the [Dialog table](#) using the following reserved names. Each dialog box and name can only be listed once. Developers must author these dialog boxes into the user interface. For information about how to preview dialog boxes, see [Importing the User Interface](#).

Dialog box name	Brief description of dialog box
<a href="#">FilesInUse Dialog</a>	Alerts user to processes overwriting or deleting files.
<a href="#">FirstRun Dialog</a>	Collects user name, company name, and product ID.
<a href="#">MsiRMFilesInUse Dialog</a>	Alerts the user to processes overwriting or deleting files and gives the user the option to use the Restart Manager to close and restart applications.

## Required Dialog Boxes

During the installation, certain events cause Windows Installer to check the [user interface sequence tables](#) in the package and display the specified dialog box. For example, in the case of a fatal error, Windows Installer displays the dialog box that is listed with a sequence number of -3 in the user interface sequence table regardless of what that dialog box is named in the [Dialog table](#). The following table lists the specific events and their corresponding sequence number in the user interface sequence table:

Type of	User interface sequence table
---------	-------------------------------

Event	sequence number	Description of dialog box
Fatal error	-3	The installation was terminated by a fatal error.
User exit	-2	The installation was terminated at the user's request.
Exit	-1	The installation completed successfully.

In addition, the package author must create a generic dialog box to display Windows Installer [error](#) messages. This dialog box can be named anything, but this name must be specified in the **ErrorDialog** property.

## Typical Dialog Boxes

The following dialog boxes are optional and are commonly included in the authored user interface of an installation package. These dialogs are typical of most [user interface wizards](#) for installing files. These dialog boxes can have any name in the Dialog table. The names shown are only recommended for clarity and can be modified as necessary. For example, two different custom **LicenseAgreement** dialog boxes can be used in the package and distinguished in the Dialog table by the names ProfessionalLicenseAgreement and LimitedLicenseAgreement.

Dialog box type	Brief description of dialog box
DiskCost dialog box	Indicates insufficient disk space for the installation.
Browse dialog box	Enables user to select a directory.
Cancel dialog box	Confirms a request to terminate the installation.
License agreement dialog box	Modal box displaying the license agreement.
Selection dialog box	Modal box enabling the user to select items.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Browse Dialog

A Browse dialog box enables the user to select a directory. The directory does not have to exist and may be created by using this control.

This type of dialog box commonly contains the following three controls. These controls are connected to the same property. That property is the path being selected.

- A [PathEdit control](#) to select the tail section of the path. This control cannot lose focus if the entered tail is not valid on the present volume.
- A [DirectoryCombo control](#) to show the presently selected path that is displayed by the PathEdit control. This control does not show the last segment of the path.
- A [DirectoryList control](#) to show the folders below the directory currently displayed by the DirectoryCombo. This can also show a folder that is not yet created.

A Browse dialog box also usually contains a [DirectoryCombo control](#) that specifies the volume types to display. It is common for all volume types to be displayed on a Browse dialog box.

Browse dialog boxes commonly contain three [PushButton controls](#). These buttons are linked to their respective ControlEvents in [ControlEvent table](#). These buttons are used for activating the following control options.

<b>Control option</b>	<b>ControlEvent</b>
Up One Level	<a href="#">DirectoryListUp</a>
New Folder	<a href="#">DirectoryListNew</a>
Open	<a href="#">DirectoryListOpen</a>

For the New Folder option to work with a non-default folder name, the new folder's path must be specified in the [UIText table](#). The path string

should use the "short file name|long file name" form for the file name. For example, use a file name such as "MyProd~1|My Fabulous Product". See the [Filename](#) column data type for more information about the file name format. If the path is not present in the UIText table, or if it is set to an invalid value, then it is set to a value of "Fldr|New Folder" by default. The **New Folder** button can be omitted if the dialog box only need to search for existing folders.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Cancel Dialog

A **Cancel** dialog box confirms that the user wants to terminate the installation. This is a modal dialog box.

This type of dialog box commonly contains a [Text control](#) and two [PushButtons](#). The two buttons give the user the choice of either returning to the last dialog box or confirming the termination of the installation.

The [EndDialog](#) ControlEvent is linked to these two buttons in the [ControlEvent table](#). The *Return* parameter of the EndDialog ControlEvent is linked to one of the buttons and causes the **Cancel** dialog box to be terminated and the focus to return to the previous dialog box. The *Exit* parameter is linked to the other button and causes the user interface to return control to the installer with the appropriate code indicating that the user wants to exit. The installer then shuts down and displays the [UserExit Dialog](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DiskCost Dialog

A DiskCost dialog box commonly appears when the [OutOfDiskSpace](#) property is set. This indicates that there is not enough disk space for the selected installation.

This dialog box may also be activated from the [Selection Dialog](#) to give the user an overview of the disk space needed for the installation.

A DiskCost dialog box box is a modal dialog box box. It commonly contains a [PushButton](#) that returns the user to the previous dialog box box and a [VolumeCostList control](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Error Dialog

An Error dialog box is a modal dialog box that displays an error message. More than one Error dialog box can exist in each installation.

An ErrorDialog property needs to be set that specifies which dialog box is to be used. If this property is not set or does not point to a valid Error dialog box, then the error messages will not be displayed. In this case, the error is only logged with a warning about the missing dialog box.

An Error dialog box must have the [Error Dialog style bit](#) set. The dialog box must have a [Text control](#) named ErrorText. The record for the Error dialog box in the [Dialog table](#) must have the ErrorText control entered into the Control\_First field.

The dialog box must contain seven [PushButtons](#). All of these buttons specify the [EndDialog](#) ControlEvent in the [ControlEvent table](#). Each button specifies one of the following attributes: **ErrorAbort**, **ErrorCancel**, **ErrorIgnore**, **ErrorNo**, **ErrorOk**, **ErrorRetry**, **ErrorYes**.

**Note** The focus of these controls should not be linked through the use of the Control\_Next column in the [Control table](#).

These buttons should be placed in approximately the same position in the dialog box because when it is created, only a subset of these seven buttons is created, depending on the message. The X coordinate of the buttons is modified so the buttons that are displayed are evenly spaced. The Y coordinate, height, and width of the buttons are unchanged. Because the buttons are arranged horizontally, no other control can be placed in the same horizontal region of the dialog box.

For an Error dialog box, the Control\_Default and Control\_Cancel fields in the Dialog table are ignored. The Control\_First field for an Error dialog box must specify the ErrorText control.

If an [Icon control](#) named ErrorIcon is included on this dialog, the following standard Windows icons are displayed:

- IDI\_ERROR in response to imtFatalExit messages.
- IDI\_WARNING in response to imtError and imtWarning messages.
- IDI\_INFORMATION in response to imtOutOfDiskSpace messages.

The ErrorIcon control should be created with the [FixedSize control attribute](#) set to avoid improper sizing of the standard Windows icons.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Exit Dialog

The Exit dialog box is displayed at the end of a successful installation. This dialog box typically contains only some text and an **OK** button. The name of this dialog box must appear in the [AdminUISequence table](#) and [InstallUISequence table](#) with -1 as the sequence number. This is a modal dialog box.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# FatalError Dialog

This modal dialog box displays at the end of an installation, if the installation is terminated because of a fatal error. This dialog box typically contains only some text and an **OK** button. The name of this dialog box must appear in the [AdminUISequence table](#) and [InstallUISequence table](#) with -3 as the sequence number.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# FilesInUse Dialog

A FilesInUse dialog box alerts the user of processes currently running files that must be overwritten or deleted by the installation. This gives the user the opportunity to shut down these processes and avoid having to reboot the computer to complete the replacement or deletion of these files. See [System Reboots](#).

This dialog box will be created as required by the [InstallValidate action](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# FirstRun Dialog

A FirstRun dialog box sequence collects user name, company name, and product ID information. The installer verifies the product ID during this dialog.

A FirstRun dialog box sequence is usually not a part of the action sequence and is instead called by the [MsiCollectUserInfo](#) function on the first run of the product.

An author of an installer package may use the template dialog sequence or create a different sequence. The dialog sequence however needs to have the user set the following properties:

- [USERNAME](#) property
- [COMPANYNAME](#) property
- [PIDKEY](#) property

The product ID will be validated during the dialog using the [ValidateProductID action](#) or the [ValidateProductID ControlEvent](#).

If the product ID is set as a property on the command line, or by a transform, then the necessity of having the user reenter the product ID during the first-run dialog can be circumvented by controlling the display using the [ProductID](#) property. Following the successful validation of the product ID the [ProductID](#) property is set to the full, valid product ID.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# LicenseAgreement Dialog

This modal dialog box displays the license agreement to the user. Typically the dialog box displays the text of the agreement using a [ScrollableText control](#) and a pair of option buttons that let the user accept or reject the agreement. For legal reasons it is advisable that neither button be presented to the user as already selected by default. This forces the user to make an active choice. Authors commonly use a [RadioButtonGroup control](#) for this purpose. Note however that the focus on a dialog box does not move to a RadioButtonGroup control until one of the buttons in the group has been selected.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiRMFilesInUse Dialog

The MsiRMFilesInUse Dialog box can be authored to display a list of processes that are currently running files that need to be overwritten or deleted by the installation. The user can select between options to "Automatically close applications and restart them" or "Do not close applications. (A reboot will be required.)" If the user selects the "Automatically close applications and restart them" option, a push button control on this dialog box can be authored to publish the RmShutdownAndRestart control event and the Restart Manager can close the applications and restart them at the end of the installation. This can eliminate or reduce the need to restart the computer. For more information, see [System Reboots](#).

The **MsiRMFilesInUse** dialog box is displayed during an installation only if all the following are true. When any of these are false, the Windows Installer ignores the **MsiRMFilesInUse** dialog box.

- A Windows Installer version that is not earlier than version 4.0 and an operating system that is not earlier than Windows Vista or Windows Server 2008 is running the installation.
- The Full UI [user interface level](#) is used.
- Interactions with the Restart Manager have not been disabled by the **MSIRESTARTMANAGERCONTROL** property or the [DisableAutomaticApplicationShutdown](#) policy.
- The **MsiRMFilesInUse** dialog box is present in the installation package.
- All calls from the Windows Installer to the Restart Manager are successful.

This dialog box will be created as required by the [InstallValidate action](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Selection Dialog

This modal dialog box enables users to select particular items.

A Selection Dialog box contains a [SelectionTree control](#) that publishes several ControlEvents. Commonly these ControlEvents are subscribed to by [Text](#), [Icon](#), or [Bitmap](#) controls that display a description, the size, the path, and the icon of the highlighted item.

There is a [PushButton control](#) on the dialog box that publishes the [SelectionBrowse ControlEvent](#) and spawns a [Browse Dialog](#). The Browse control enables the user to select a directory.

The selection tree is populated only after the [CostInitialize action](#) and [CostFinalize action](#) are called.

A Selection dialog box is commonly used to select features. The features are listed as items in a SelectionTree control and labeled with the short string of text appearing in the Title column of the [Feature table](#). The text string in the Description column of the Feature table is published as a [SelectionDescription ControlEvent](#) and displayed by a [Text control](#) in the Selection Dialog box. The Selection tree control also publishes the handle to the icon of the highlighted item.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UserExit Dialog

This modal dialog box displays at the end of an installation that was ended by the user. This dialog box typically contains only some text and an **OK** button. The name of this dialog box must appear in the [AdminUISequence table](#) and [InstallUISequence table](#) with -2 as the sequence number.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Dialog Style Bits

The style of a dialog box is specified in the Attributes column of the [Dialog table](#) by a 32-bit word composed of style bit flags.

The following list provides links to descriptions of reserved dialog box style bits.

Name	Decimal	Hexadecimal	Constant
<a href="#">Visible</a>	1	0x00000001	msidbDialogAttributesVisible
<a href="#">Modal</a>	2	0x00000002	msidbDialogAttributesModal
<a href="#">Minimize</a>	4	0x00000004	msidbDialogAttributesMinimize
<a href="#">SysModal</a>	8	0x00000008	msidbDialogAttributesSysModal
<a href="#">KeepModeless</a>	16	0x00000010	msidbDialogAttributesKeepModel
<a href="#">TrackDiskSpace</a>	32	0x00000020	msidbDialogAttributesTrackDiskS
<a href="#">UseCustomPalette</a>	64	0x00000040	msidbDialogAttributesUseCustomi
<a href="#">RTLRO</a>	128	0x00000080	msidbDialogAttributesRTLRO
<a href="#">RightAligned</a>	256	0x00000100	msidbDialogAttributesRightAligne
<a href="#">LeftScroll</a>	512	0x00000200	msidbDialogAttributesLeftScroll
<a href="#">BiDi</a>	896	0x00000380	msidbDialogAttributesBiDi = msidbDialogAttributesRTLRO   msidbDialogAttributesRightAligne msidbDialogAttributesLeftScroll
<a href="#">Error</a>	65536	0x00010000	msidbDialogAttributesError

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# BiDi Dialog Style Bit

This is a combination of the right to left reading order [RTLRO](#), the [RightAligned](#), and the [LeftScroll](#) dialog style bits.

## Value

Decimal	Hexadecimal	Constant
896	0x00000380	msidbDialogAttributesBiDi = msidbDialogAttributesRTLRO   msidbDialogAttributesRightAligned   msidbDialogAttributesLeftScroll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Error Dialog Style Bit

If this bit is set, the dialog box is an error dialog.

There can be more than one dialog with this style. The [ErrorDialog](#) property determines which dialog is used as an error dialog. The selected dialog can be only one of those that have this style bit set. The error dialog must have a static text control named `ErrorText`. This control receives the text of the error message. The dialog should also have the seven push buttons corresponding to the possible return values. The error message determines which of these buttons are actually displayed. The displayed buttons are rearranged so they are evenly distributed on the dialog. This rearrangement changes the X coordinate of the buttons, but not the other three coordinates. Therefore it is advisable that no other control should be authored in the same horizontal region of the dialog as the buttons. If the error message specifies no button, the **OK** button is displayed. The **Default** button, **First active control** and **Cancel** button values are ignored for an error dialog. The **Default** button defined in the error message will be assigned to all three values. The effect of pushing these buttons have to be defined in the [ControlEvent table](#) just like for all other buttons. The title of the dialog is authored in a way similar to other dialogs. It can get overwritten by the error message if it specifies a header text after the button list.

## Value

Decimal	Hexadecimal	Constant
65536	0x00010000	<code>msidbDialogAttributesError</code>

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# KeepModeless Dialog Style Bit

Normally, when this bit is not set and a dialog box is created through DoAction, all other (typically modeless) dialogs are destroyed. If this bit is set, the other dialogs stay alive when this dialog box is created.

## Value

Decimal	Hexadecimal	Constant
16	0x00000010	msidbDialogAttributesKeepModeless

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# LeftScroll Dialog Style Bit

If this style bit is set, the scroll bar is located on the left side of the dialog box.

## Value

Decimal	Hexadecimal	Constant
512	0x00000200	msidbDialogAttributesLeftScroll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Minimize Dialog Style Bit

If this bit is set, the dialog box can be minimized. This bit is ignored for modal dialog boxes, which cannot be minimized.

## Value

Decimal	Hexadecimal	Constant
4	0x00000004	msidbDialogAttributesMinimize

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Modal Dialog Style Bit

If this bit is set, the dialog box is modal, other dialogs of the same application cannot be put on top of it, and the dialog keeps the control while it is running. If this bit is not set, the dialog is modeless, other dialogs of the same application may be moved on top of it. After a modeless dialog is created and displayed, the user interface returns control to the installer. The installer then calls the user interface periodically to update the dialog and to give it a chance to process the messages. As soon as this is done, the control is returned to the installer.

**Note** There should be no modeless dialogs in a wizard sequence, since this would return control to the installer, ending the wizard sequence prematurely.

## Value

Decimal	Hexadecimal	Constant
2	0x00000002	msidbDialogAttributesSysModal

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RightAligned Dialog Style Bit

If this style bit is set, the text is aligned on the right side of the dialog box.

## Value

Decimal	Hexadecimal	Constant
256	0x00000100	msidbDialogAttributesRightAligned

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RTLRO Dialog Style Bit

If this style bit is set the text in the dialog box is displayed in right-to-left-reading order.

## Value

Decimal	Hexadecimal	Constant
128	0x00000080	msidbDialogAttributesRTLRO

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SysModal Dialog Style Bit

If this style bit is set, the dialog box will stop all other applications and no other applications can take the focus. This state remains until the SysModal dialog is dismissed.

## Value

Decimal	Hexadecimal	Constant
8	0x00000008	msidbDialogAttributesSysModal

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# TrackDiskSpace Dialog Style Bit

If this bit is set, the dialog box periodically calls the installer. If the property changes, it notifies the controls on the dialog. This style can be used if there is a control on the dialog indicating disk space. If the user switches to another application, adds or removes files, or otherwise modifies available disk space, you can quickly implement the change using this style.

Any dialog box relying on the [OutOfDiskSpace](#) property to determine whether to bring up a dialog must set the TrackDiskSpace Dialog Style Bit for the dialog to dynamically update space on the target volumes.

## Value

Decimal	Hexadecimal	Constant
32	0x00000020	msidbDialogAttributesTrackDiskSpace

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UseCustomPalette Dialog Style Bit

If this bit is set, the pictures on the dialog box are created with the custom palette (one per dialog received from the first control created). If the bit is not set, the pictures are rendered using a default palette.

Typically one would set this bit if the dialog contains a picture with a special palette, or several pictures sharing a custom palette. The bit should not be set if the dialog contains several pictures with different palettes. In this case, the default palette is most likely to give a satisfactory result for all pictures.

## Value

Decimal	Hexadecimal	Constant
64	0x00000040	msidbDialogAttributesUseCustomPalette

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Visible Dialog Style Bit

If this bit is set the dialog is originally created as visible, otherwise it is hidden.

## Value

Decimal	Hexadecimal	Constant
1	0x00000001	msidbDialogAttributesVisible

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Controls

Developers of installation packages can author a user interface containing the controls discussed in this topic. For information on how to add a particular control to a dialog box, see the topic for that control and read the section [Adding Controls and Text](#).

Some controls, such as CheckBox and ComboBox, are associated with a property specified in the Property column of the [Control table](#). A user changes the value of this property by interacting with the control. Passive controls, such as Billboard and bitmap, are not associated with such a property.

For security, private properties cannot be changed by a user interacting with the user interface. For a property to be set by the user interface, it needs to be a public property and in uppercase. See also [About Properties](#).

In some cases a control may be redrawn incorrectly when canceling out of a dialog. This has to do with the order in which the controls receive WM\_PAINT messages after the Cancel dialog is removed. To fix this, try changing the order of the controls in the Control table.

<b>Control name</b>	<b>Associated property</b>	<b>Brief description of control</b>
<a href="#">Billboard</a>	No	Displays billboards based on progress messages.
<a href="#">Bitmap</a>	No	Displays a static picture of a bitmap.
<a href="#">CheckBox</a>	Yes	A two-state check box.
<a href="#">ComboBox</a>	Yes	A drop-down list with an edit field.
<a href="#">DirectoryCombo</a>	Yes	Select all except the last segment of the path.
<a href="#">DirectoryList</a>	Yes	Displays folders below the main part of path.
<a href="#">Edit</a>	Yes	A regular edit field for any string or integer.

GroupBox	No	Displays a rectangle that groups other controls together.
Hyperlink	No	Displays a HTML link to an address, which opens in the default browser.  <b>Windows Installer 4.5 and earlier:</b> Not supported.
Icon	No	Displays a static picture of an icon.
Line	No	Displays a horizontal line.
ListBox	Yes	A drop-down list without an edit field.
ListView	Yes	Displays a column of values with icons for selection.
MaskedEdit	Yes	An edit field with a mask in the text field.
PathEdit	Yes	Displays folder name or entire path in an edit field.
ProgressBar control	No	Bar graph that changes length as it receives progress messages.
PushButton	No	Displays a basic push button.
RadioButtonGroup	Yes	A group of radio buttons.
ScrollableText	No	Displays a long string of text.
SelectionTree	Yes	Displays information from the Feature table and enables the user to change their selection state.
Text	No	Displays static text.
VolumeCostList	No	Displays costing information on different volumes.
VolumeSelectCombo	Yes	Selects volume from an alphabetical list.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Billboard Control

The Billboard control displays commonly used controls that are added and removed from the dialog box by ControlEvents. Only controls that are not associated with a property, such as [Text](#), [Bitmap](#), or [Icon](#), or custom controls can be placed on a billboard. Billboard controls most typically display progress messages.

## Control Attributes

You can use the following attributes with the Billboard control. To change the value of an attribute using an event, subscribe the control to a ControlEvent in the [EventMapping table](#) and list the attribute's identifier in the Attribute column. Enter the identifier of the ControlEvent in the Event column.

Attribute identifier	Hexadecimal bit	Description
<a href="#">BillboardName</a>		Name of the current billboard. Enter the billboard's identifier in the Billboard column of the <a href="#">BBControl table</a> .
<a href="#">Position</a>		Position of control in the dialog box. Enter the control's width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the <a href="#">BBControl table</a> . Use <a href="#">installer units</a> for length and distance.
<a href="#">Visible</a>	0x00000000 0x00000001	Hidden control. Visible control.  Include this bit in the bit word of the Attributes column in the <a href="#">BBControl table</a> to make the control visible or hidden upon its creation.  Hide or show a control by using the <a href="#">ControlCondition table</a> .

<a href="#">Sunken</a>	0x00000000 0x00000004	Displays the default visual style. Displays the control with a sunken, 3-D, look.  Include these bits in the bit word in the Attributes column of the <a href="#">Control table</a> .
------------------------	--------------------------	--

## Remarks

This control has no window of its own.

Billboard controls that appear in the full user interface are listed in the [Billboard table](#).

Controls that are located on a billboard must be listed in the [BBControl table](#) rather than the Control table.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Bitmap Control

The Bitmap control displays a bitmap or JPEG static picture file. The Windows Installer will automatically determine the format of the binary data and display the picture.

## Control Attributes

You can use the following attributes with this control. To change the value of an attribute using an event, subscribe the control to a ControlEvent in the [EventMapping table](#) and list the attribute's identifier in the Attribute column. Enter the identifier of the ControlEvent in the Event column.

Attribute identifier	Hexadecimal bit	Description
<a href="#">Position</a>		Position of the control in the dialog box. Enter the control's width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the <a href="#">Control table</a> or <a href="#">BBControl table</a> . Use <a href="#">installer units</a> for length and distance.
<a href="#">Text</a>		Contains the name of a bitmap stored in the <a href="#">Binary table</a> . To display a bitmap stored in the Binary table, do the following. Enter the name of the bitmap image appearing in the Name column of the Binary table into the Text column of the <a href="#">Control table</a> record for this control.
<a href="#">Visible</a>	0x00000000 0x00000001	Hidden control. Visible control.  Include this bit in the bit word of the Attributes column in the <a href="#">Control table</a> or <a href="#">BBControl table</a> to make the control visible or hidden upon its creation.  You can also hide or show a control by using the

		<a href="#">ControlCondition table</a> .
<a href="#">Sunken</a>	0x00000000 0x00000004	Displays the default visual style. Displays the control with a sunken, 3-D look.  Include these bits in the bit word in the Attributes column of the <a href="#">Control table</a> .
<a href="#">FixedSize control</a>	0x00000000 0x00100000	Stretches the bitmap image to fit the control. Crops or centers the bitmap image in the control.  Include this bit in the bit word of the Attributes column of the <a href="#">BBControl table</a> or the <a href="#">Control table</a> .

## Remarks

This control can be created from the `STATIC` class by using the **CreateWindowEx** function. It has the `SS_BITMAP`, `SS_CENTERIMAGE`, `WS_CHILD`, and `WS_GROUP` styles.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# CheckBox Control

This `CheckBox_control` is a two-state check box. To associate an integer or string property with this control, enter the property name into the Property column of the [Control table](#). The selected state of the box sets the property either to the value specified in the Value column of the [CheckBox table](#) or to the initial value of the property specified in the [Property table](#). If the property has no initial value, the checked state sets it to 1. The unselected state sets the property to null.

CheckBox controls can only be used to publish [AddLocal ControlEvent](#), [AddSource ControlEvent](#), [Remove ControlEvent](#), [DoAction ControlEvent](#), or [SetProperty ControlEvent](#) controls.

## Control Attributes

You can use the following attributes with this control. To change the value of an attribute using an event, subscribe the control to a `ControlEvent` in the [EventMapping table](#) and list the attribute's identifier in the Attribute column. Enter the identifier of the `ControlEvent` in the Event column.

Attribute identifier	Hexadecimal bit	Description
<a href="#">IndirectPropertyName</a>		This is the name of an indirect property associated with the control. If the Indirect attribute bit is set, the control displays or changes the value of the property having this name. If the Indirect attribute bit is set, this name is also the value of the property listed in the Property column of the <a href="#">Control table</a> .
<a href="#">Position</a>		Position of control in the dialog box. Enter the control's width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the <a href="#">Control table</a> . Use <a href="#">installer units</a> for length and distance.

<p>PropertyName</p>		<p>This is the name of the property associated with this control. If the Indirect attribute bit is not set, the control displays or changes the value of the property having this name. This attribute is specified in the Property column of the <a href="#">Control table</a>.</p>
<p>PropertyValue</p>		<p>Current value of the property displayed or changed by this control. If the Indirect attribute bit is not set, this is the value of PropertyName. If the Indirect attribute bit is set, this is the value of IndirectPropertyName. If the attribute changes, the control reflects the new value.</p>
<p>Text</p>		<p>This control attribute can specify the text displayed by the control, an image stored in the Binary table, or an image set at run time.</p> <p>To specify text, enter the text string into the Text column of the <a href="#">Control table</a>. To set the font and font style of a text string, prefix the string of displayed characters with <code>{\style}</code> or <code>{&amp;style}</code>. Where style is an identifier listed in the TextStyle column of the <a href="#">TextStyle table</a>. If neither of these are present, but the <b>DefaultUIFont</b> property is defined as a valid text style, that font will be used.</p> <p>To specify an icon or bitmap image stored in the <a href="#">Binary table</a>, enter the primary key of the image's record from the Name column of the Binary table into the Text column of the <a href="#">Control table</a> record for the control.</p>

<p><b>Visible</b></p>	<p>0x00000000 0x00000001</p>	<p>Hidden control. Visible control.</p> <p>Include this bit in the bit word of the Attributes column in the <a href="#">Control table</a> to make the control visible or hidden upon its creation.</p> <p>You can also hide or show a control by using the <a href="#">ControlCondition table</a>.</p>
<p><b>Enabled</b></p>	<p>0x00000000 0x00000002</p>	<p>Control in a disabled state. Control in an enabled state.</p> <p>Include this bit in the Attributes column of the <a href="#">Control table</a> to enable the control on creation.</p> <p>You can also enable or disable a control by using the <a href="#">ControlCondition table</a>.</p>
<p><b>Sunken</b></p>	<p>0x00000000 0x00000004</p>	<p>Displays the default visual style. Displays the control with a sunken, 3-D, look.</p> <p>Include these bits in the bit word in the Attributes column of the <a href="#">Control table</a>.</p>
<p><b>Indirect</b></p>	<p>0x00000000 0x00000008</p>	<p>The control displays or changes the value of the property in the Property column of the <a href="#">Control table</a>. The control displays or changes the value of the property that has the Identifier listed in the Property column of the <a href="#">Control table</a>.</p> <p>Determines if the property associated with this control is referenced indirectly.</p>

<a href="#">Integer</a>	0x00000000 0x00000010	Property associated with the control is a string value. Property associated with the control is an integer value.  Include this bit in the bit word of the Attributes column of the <a href="#">Control table</a> to set this attribute on creation of the control.
<a href="#">RTLRO</a>	0x00000000 0x00000020	Text in the control is displayed in left-to-right reading order. Text in the control is displayed in right-to-left reading order.
<a href="#">RightAligned</a>	0x00000000 0x00000040	Text in the control is aligned to the left. Text in the control is aligned to the right.
<a href="#">PushLike</a>	0x00000000 0x00020000	Control is drawn with its usual appearance. Control has the BS_PUSHLIKE style, and is drawn to appear as a push button.  Include this bit in the bit word of the Attributes column of the <a href="#">Control table</a> to set this attribute on creation of the control.

## Remarks

This control can be created from the `BUTTON` class by using the `CreateWindowEx` function. It has the `BS_CHECKBOX`, `WS_TABSTOP`, `WS_GROUP`, `WS_CHILD`, and `BS_MULTILINE` styles.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ComboBox Control

The ComboBox control displays a drop-down list of predefined values and an edit field into which the user can enter a value. To associate this control with a string or integer property, enter the property's name in the Property column of the [Control table](#).

## Control Attributes

You can use the following attributes with this control. To change the value of an attribute using an event, subscribe the control to a ControlEvent in the [EventMapping table](#) and list the attribute's identifier in the Attribute column. Enter the identifier of the ControlEvent in the Event column.

<b>Attribute identifier</b>	<b>Hexadecimal bit</b>	<b>Description</b>
<a href="#">IndirectPropertyName</a>		This is the name of an indirect property associated with the control. If the Indirect attribute bit is set, the control displays or changes the value of the property having this name. If the Indirect attribute bit is set, this name is also the value of the property listed in the Property column of the <a href="#">Control table</a> .
<a href="#">Position</a>		Position of the control in the dialog box. Enter the control's width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the <a href="#">Control table</a> . Use <a href="#">installer units</a> for length and distance.
<a href="#">PropertyName</a>		This is the name of the property associated with this control. If the Indirect attribute bit is not set, the control displays or changes the value

		of the property having this name. This attribute is specified in the Property column of the <a href="#">Control table</a> .
<a href="#">PropertyValue</a>		The current value of the property displayed or changed by this control. If the Indirect attribute bit is not set, this is the value of PropertyName. If the Indirect attribute bit is set, this is the value of IndirectPropertyName. If the attribute changes, the control reflects the new value.
<a href="#">Text</a>		To set the font and font style of a text string, prefix the string of displayed characters with {\style} or {&style}. Where style is an identifier listed in the TextStyle column of the <a href="#">TextStyle table</a> . If neither of these are present, but the <b>DefaultUIFont</b> property is defined as a valid text style, that font will be used. To specify the number of characters the user can enter, append {n} after any font specifications, where n is a positive integer.
<a href="#">Visible</a>	0x00000000 0x00000001	Hidden control. Visible control.  Include this bit in the bit word of the Attributes column in the <a href="#">Control table</a> to make the control visible or hidden upon its creation.  You can also hide or show a control by using the <a href="#">ControlCondition table</a> .
<a href="#">Enabled</a>	0x00000000 0x00000002	Control in a disabled state. Control in an enabled state.  Include this bit in the bit word in the

		<p>Attributes column of the <a href="#">Control</a> to enable the control on creation.</p> <p>You can also enable or disable a control by using the <a href="#">ControlCondition table</a>.</p>
<a href="#">Sunken</a>	<p>0x00000000 0x00000004</p>	<p>Displays the default visual style.</p> <p>Displays the control with a sunken, 3-D, look.</p> <p>Include these bits in the bit word in the Attributes column of the <a href="#">Control table</a>.</p>
<a href="#">Indirect</a>	<p>0x00000000 0x00000008</p>	<p>The control displays or changes the value of the property in the Property column of the <a href="#">Control table</a>.</p> <p>The control displays or changes the value of the property that has the identifier listed in the Property column of the Control table.</p> <p>Determines if the property associated with this control is referenced indirectly.</p>
<a href="#">Integer</a>	<p>0x00000000 0x00000010</p>	<p>Property associated with the control is a string value.</p> <p>Property associated with the control is an integer value.</p> <p>Include this bit in the bit word of the Attributes column of the <a href="#">Control table</a> to set this attribute on creation of the control.</p>
<a href="#">RTLRO</a>	<p>0x00000000 0x00000020</p>	<p>Text in the control is displayed in left-to-right reading order.</p> <p>Text in the control is displayed in right-to-left reading order.</p>

<a href="#">RightAligned</a>	0x00000000 0x00000040	Text in the control is aligned to the left. Text in the control is aligned to the right.
<a href="#">LeftScroll</a>	0x00000000 0x00000080	The scroll bar is located on the right side of the control. The scroll bar is located on the left side of the control.
<a href="#">BiDi</a>	0x000000E0	Set this value for a combination of the <a href="#">RTLRO</a> , <a href="#">RightAligned</a> , and <a href="#">LeftScroll</a> attributes.
<a href="#">Sorted</a>	not set 0x00010000	Items displayed in alphabetical order. Items displayed in order specified in the <a href="#">ListView table</a> .  The control queries the <a href="#">ComboBox table</a> , and if the Sorted style bit is set, the <b>ComboBox</b> has the CBS_SORT style and displays items as specified by Ordering. If this style bit is not set, items are displayed in alphabetical order.
<a href="#">ComboList</a>	not set 0x00020000	Combo box with edit field. Combo box with edit field is replaced by a static text field.
<a href="#">UsersLanguage</a>	0x00000000 0x00100000	Fonts created in the database code page. Fonts created in the user's default UI code page.

## Remarks

This control can be created from the COMBOBOX class by using the **CreateWindowEx** function. It has the CBS\_AUTOHSCROLL, WS\_TABSTOP, WS\_GROUP, and WS\_CHILD styles. If the ComboList bit is on, it also has the CBS\_DROPDOWNLIST style, otherwise it has the CBS\_DROPDOWN style.

The length of the text that can be entered can be limited by putting a number from 0 to 2147483646 in curly braces at the beginning of the Text field in the [Control table](#). For example if the text field starts with {80}, the length of the string is limited at 80 characters. If no such limit is supplied in the table, or if 0 is specified, the length is set to the maximum possible (2147483646 characters). A negative or non-numeric value will generate an error.

For compatibility with screen readers, when authoring a dialog box with a ComboBox control as the first active control, you must make the text field belonging to the edit field the first active control in the [Dialog table](#). Since the static text cannot take focus, when the dialog box is created, the edit field will have the focus initially as intended. Doing this ensures that screen readers show the correct information.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DirectoryCombo Control

A DirectoryCombo\_control displays a part of the path that is currently displayed in the [PathEdit control](#). This control does not show the last segment of the path, that segment is displayed by the [DirectoryList control](#).

The DirectoryCombo\_control displays all available volumes in alphabetical order and hierarchical steps of the current path. If the selected path contains any folders that do not exist, those files are displayed with a different icon. The types of volumes displayed are specified using the bits associated with [RemovableVolume](#), [FixedVolume](#), [RemoteVolume](#), [CDROMVolume](#), [RAMDiskVolume](#), and [FloppyVolume](#) controls.

The PathEdit, DirectoryCombo, and DirectoryList controls are associated with the same string-valued property. That property is the path selected by the user. Enter the property's name into the Property column of the [Control table](#). This property must have an initial value containing at least a one volume and one sublevel. Specify the initial value for the property in the Value column of the [Property table](#).

This control is intended to be used on a [Browse Dialog](#) together with the [PathEdit](#) and [DirectoryList](#) controls.

## Control Attributes

You can use the following attributes with this control. To change the value of an attribute using an event, subscribe the control to a ControlEvent in the [EventMapping table](#) and list the attribute's identifier in the Attribute column. Enter the identifier of the ControlEvent in the Event column.

<b>Attribute identifier</b>	<b>Hexadecimal bit</b>	<b>Description</b>
<a href="#">IndirectPropertyName</a>		This is the name of an indirect property associated with the control. If the Indirect attribute bit is set, the control displays or changes the value of the property having this name. If

		<p>the Indirect attribute bit is set, this name is also the value of the property listed in the Property column of the <a href="#">Control table</a>.</p>
<a href="#">Position</a>		<p>Position of the control in the dialog box.</p> <p>Enter the control's width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the <a href="#">Control table</a>. Use <a href="#">installer units</a> for length and distance.</p>
<a href="#">PropertyName</a>		<p>This is the name of the property associated with this control. If the Indirect attribute bit is not set, the control displays or changes the value of the property having this name. This attribute is specified in the Property column of the <a href="#">Control table</a>.</p>
<a href="#">PropertyValue</a>		<p>Current value of the property displayed or changed by this control. If the Indirect attribute bit is not set, this is the value of PropertyName. If the Indirect attribute bit is set, this is the value of IndirectPropertyName. If the attribute changes, the control reflects the new value.</p>
<a href="#">Text</a>		<p>To set the font and font style of a text string, prefix the string of displayed characters with <code>{\style}</code> or <code>{&amp;style}</code>. Where style is an identifier listed in the TextStyle column of the <a href="#">TextStyle table</a>. If neither of these are present, but the <b>DefaultUIFont</b> property is defined as a valid text style, that font will be used.</p>

Visible	0x00000000 0x00000001	<p>Hidden control. Visible control.</p> <p>Include this bit in the bit word of the Attributes column in the <a href="#">Control table</a> to make the control visible or hidden upon its creation.</p> <p>You can also hide or show a control by using the <a href="#">ControlCondition table</a>.</p>
Enabled	0x00000000 0x00000002	<p>Control in a disabled state. Control in an enabled state.</p> <p>Include this bit in the bit word in the Attributes column of the <a href="#">Control table</a> to enable the control on creation.</p> <p>You can also enable or disable a control by using the <a href="#">ControlCondition table</a>.</p>
Sunken	0x00000000 0x00000004	<p>Displays the default visual style. Displays the control with a sunken, 3D look.</p> <p>Include these bits in the bit word in the Attributes column of the <a href="#">Control table</a>.</p>
Indirect	0x00000000 0x00000008	<p>The control displays or changes the value of the property in the Property column of the <a href="#">Control table</a>.</p> <p>The control displays or changes the value of the property that has the Identifier listed in the Property column of the Control table.</p> <p>Determines if the property associated with this control is referenced indirectly.</p>

<a href="#">RTLRO</a>	0x00000000 0x00000020	Text in the control is displayed in left-to-right reading order. Text in the control is displayed in right-to-left reading order.
<a href="#">RightAligned</a>	0x00000000 0x00000040	Text in the control is aligned to the left. Text in the control is aligned to the right.
<a href="#">LeftScroll</a>	0x00000000 0x00000080	The scroll bar is located on the right side of the control. The scroll bar is located on the left side of the control.
<a href="#">BiDi</a>	0x000000E0	Set this value for a combination of the <a href="#">RTLRO</a> , <a href="#">RightAligned</a> , and <a href="#">LeftScroll</a> attributes.
<a href="#">RemovableVolume</a>	0x00010000	Control lists removable drives. Include in the bit word in the Attributes column of the <a href="#">Control table</a> .
<a href="#">FixedVolume</a>	0x00020000	Control lists fixed internal hard drives. Include in the bit word in the Attributes column of the <a href="#">Control table</a> .
<a href="#">RemoteVolume</a>	0x00040000	Control lists remote volumes. Include in the bit word in the Attributes column of the <a href="#">Control table</a> .
<a href="#">CDROMVolume</a>	0x00080000	Control lists CD-ROM volumes. Include in the bit word in the Attributes column of the <a href="#">Control table</a> .

<a href="#">RAMDiskVolume</a>	0x00100000	Control lists RAM disks. Include in the bit word in the Attributes column of the <a href="#">Control table</a> .
<a href="#">FloppyVolume</a>	0x00200000	Control lists floppy drives. Include in the bit word in the Attributes column of the <a href="#">Control table</a> .

## Remarks

This control can be created from the COMBOBOX class by using the **CreateWindowEx** function. It has the CBS\_DROPDOWNLIST, CBS\_OWNERDRAWFIXED, CBS\_HASSTRINGS, WS\_CHILD, WS\_GROUP, WS\_TABSTOP, and WS\_VSCROLL styles.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DirectoryList Control

A DirectoryList control displays a part of the path that is currently displayed in the [PathEdit control](#). The DirectoryList control displays the folders below the directory currently displayed by the [DirectoryCombo control](#).

The PathEdit, DirectoryCombo, and DirectoryList controls are associated with the same string valued property. That property is the path selected by the user. Enter the property's name into the Property column of the [Control table](#). This property must have an initial value containing at least one volume and one sublevel. Specify the initial value for the property in the Value column of the [Property table](#).

This control is intended to be used on a [Browse Dialog](#) together with the [PathEdit](#) and [DirectoryList](#) control.

The DirectoryList control publishes the following ControlEvents.

ControlEvent	Description
<a href="#">DirectoryListNew</a>	Creates a new folder and selects the name field for editing.
<a href="#">IgnoreChange</a>	Highlights, but does not open, a folder in the current directory.
<a href="#">DirectoryListUp</a>	Selects the parent of the present directory.
<a href="#">DirectoryListOpen</a>	Selects and highlights a directory.

The contents of the Text field of the [Control table](#) is never displayed by the DirectoryList control. Instead this field specifies the style of text to be displayed by the control and contains a description of the control used by screen review utilities. To set the font and font style of a text string, prefix the string of displayed characters with `{style}` or `&{style}`. Where style is an identifier listed in the TextStyle column of the [TextStyle table](#). If neither of these are present, but the [DefaultUIFont](#) property is defined as a valid text style, that font will be used. The information following this is read by screen review utilities as the description of the control. See [Accessibility](#).

## Control Attributes

You can use the following attributes with this control. To change the value of an attribute using an event, subscribe the control to a ControlEvent in the [EventMapping table](#) and list the attribute's identifier in the Attribute column. Enter the identifier of the ControlEvent in the Event column.

<b>Attribute identifier</b>	<b>Hexadecimal bit</b>	<b>Description</b>
<a href="#">IndirectPropertyName</a>		This is the name of an indirect property associated with the control. If the Indirect attribute bit is set, the control displays or changes the value of the property having this name. If the Indirect attribute bit is set, this name is also the value of the property listed in the Property column of the <a href="#">Control table</a> .
<a href="#">Position</a>		Position of the control in the dialog box. Enter the control's width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the <a href="#">Control table</a> . Use <a href="#">installer units</a> for length and distance.
<a href="#">PropertyName</a>		This is the name of the property associated with this control. If the Indirect attribute bit is not set, the control displays or changes the value of the property having this name. This attribute is specified in the Property column of the <a href="#">Control table</a> .
<a href="#">PropertyValue</a>		Current value of the property displayed or changed by this control. If the Indirect attribute bit is not set, this is the value of PropertyName. If

		<p>the Indirect attribute bit is set, this is the value of IndirectPropertyName. If the attribute changes, the control reflects the new value.</p>
<a href="#">Text</a>		<p>To display text in screen readers, enter the text into the Text column of the <a href="#">Control table</a>. See <a href="#">Accessibility</a>.</p>
<a href="#">Visible</a>	<p>0x00000000 0x00000001</p>	<p>Hidden control. Visible control.</p> <p>Include this bit in the bit word of the Attributes column in the <a href="#">Control table</a>.to make the control visible or hidden upon its creation.</p> <p>You can also hide or show a control by using the <a href="#">ControlCondition table</a>.</p>
<a href="#">Enabled</a>	<p>0x00000000 0x00000002</p>	<p>Control in a disabled state. Control in an enabled state.</p> <p>Include this bit in the bit word in the Attributes column of the <a href="#">Control</a> to enable the control on creation.</p> <p>You can also enable or disable a control by using the <a href="#">ControlCondition table</a>.</p>
<a href="#">Sunken</a>	<p>0x00000000 0x00000004</p>	<p>Displays the default visual style. Displays the control with a sunken, 3D, look.</p> <p>Include these bits in the bit word in the Attributes column of the <a href="#">Control table</a>.</p>
<a href="#">Indirect</a>	<p>0x00000000 0x00000008</p>	<p>The control displays or changes the value of the property in the Property column of the <a href="#">Control table</a>. The control displays or changes the</p>

		<p>value of the property that has the Identifier listed in the Property column of the Control table.</p> <p>Determines if the property associated with this control is referenced indirectly.</p>
<a href="#">RTLRO</a>	0x00000000 0x00000020	<p>Text in the control is displayed in left-to-right reading order.</p> <p>Text in the control is displayed in right-to-left reading order.</p>
<a href="#">RightAligned</a>	0x00000000 0x00000040	<p>Text in the control is aligned to the left.</p> <p>Text in the control is aligned to the right.</p>
<a href="#">LeftScroll</a>	0x00000000 0x00000080	<p>The scroll bar is located on the right side of the control.</p> <p>The scroll bar is located on the left side of the control.</p>
<a href="#">BiDi Control</a>	0x000000E0	<p>Set this value for a combination of the <a href="#">RTLRO</a>, <a href="#">RightAligned</a>, and <a href="#">LeftScroll</a> attributes.</p>

## Remarks

This control can be created from the `WC_LISTVIEW` class by using the **CreateWindowEx** function. It has the `LVS_LIST`, `LVS_EDITLABELS`, `WS_VSCROLL`, `LVS_SHAREIMAGELISTS`, `LVS_AUTOARRANGE`, `LVS_SINGLESEL`, `WS_BORDER`, `LVS_SORTASCENDING`, `WS_CHILD`, `WS_GROUP`, and `WS_TABSTOP` styles.

This control lets the user select a subfolder of the current selection. With additional buttons it also lets the user select a new folder in the current selection or step up one level in the path. If the user chooses the **Create New Folder** button in a folder where a new folder already exists, a

second new folder is not created and the existing new folder's name is selected for editing.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Edit Control

The Edit control is an edit field that is associated with a string or integer value property. Enter the property's name into the Property column of the [Control table](#).

## Control Attributes

You can use the following attributes with this control. To change the value of an attribute using an event, subscribe the control to a ControlEvent in the [EventMapping table](#) and list the attribute's identifier in the Attribute column. Enter the identifier of the ControlEvent in the Event column.

<b>Attribute identifier</b>	<b>Hexadecimal bit</b>	<b>Description</b>
<a href="#">IndirectPropertyName</a>		This is the name of an indirect property associated with the control. If the Indirect attribute bit is set, the control displays or changes the value of the property having this name. If the Indirect attribute bit is set, this name is also the value of the property listed in the Property column of the <a href="#">Control table</a> .
<a href="#">Position</a>		Position of the control in the dialog box. Enter the control's width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the <a href="#">Control table</a> . Use <a href="#">installer units</a> for length and distance.
<a href="#">PropertyName</a>		This is the name of the property associated with this control. If the Indirect attribute bit is not set, the control displays or changes the value of the property having this name. This

		attribute is specified in the Property column of the <a href="#">Control table</a> .
<a href="#">PropertyValue</a>		Current value of the property displayed or changed by this control. If the Indirect attribute bit is not set, this is the value of <code>PropertyName</code> . If the Indirect attribute bit is set, this is the value of <code>IndirectPropertyName</code> . If the attribute changes, the control reflects the new value.
<a href="#">Text</a>		To set the font and font style of a text string, prefix the string of displayed characters with <code>{\style}</code> or <code>{&amp;style}</code> . Where <code>style</code> is an identifier listed in the <code>TextStyle</code> column of the <a href="#">TextStyle table</a> . If neither of these are present, but the <b>DefaultUIFont</b> property is defined as a valid text style, that font will be used. To specify the number of characters the user can enter, append <code>{n}</code> after any font specifications. Where <code>n</code> is a positive integer.
<a href="#">Visible</a>	0x00000000 0x00000001	Hidden control. Visible control.  Include this bit in the bit word of the <code>Attributes</code> column in the <a href="#">Control table</a> to make the control visible or hidden upon its creation.  You can also hide or show a control by using the <a href="#">ControlCondition table</a> .
<a href="#">Enabled</a>	0x00000000 0x00000002	Control in a disabled state. Control in an enabled state.  Include this bit in the bit word in the <code>Attributes</code> column of the <a href="#">Control</a> to

		<p>enable the control on creation.</p> <p>You can also enable or disable a control by using the <a href="#">ControlCondition table</a>.</p>
<a href="#">Sunken</a>	<p>0x00000000 0x00000004</p>	<p>Displays the default visual style.</p> <p>Displays the control with a sunken, 3-D, look.</p> <p>Include these bits in the bit word in the Attributes column of the <a href="#">Control table</a>.</p>
<a href="#">Indirect</a>	<p>0x00000000 0x00000008</p>	<p>The control displays or changes the value of the property in the Property column of the <a href="#">Control table</a>.</p> <p>The control displays or changes the value of the property that has the Identifier listed in the Property column of the Control table.</p> <p>Determines if the property associated with this control is referenced indirectly.</p>
<a href="#">Integer</a>	<p>0x00000000 0x00000010</p>	<p>Property associated with the control is a string value.</p> <p>Property associated with the control is an integer value.</p> <p>Include this bit in the bit word of the Attributes column of the <a href="#">Control table</a> to set this attribute on creation of the control.</p>
<a href="#">RTLRO</a>	<p>0x00000000 0x00000020</p>	<p>Text in the control is displayed in left-to-right reading order.</p> <p>Text in the control is displayed in right-to-left reading order.</p>

<a href="#">RightAligned</a>	0x00000000 0x00000040	Text in the control is aligned to the left. Text in the control is aligned to the right.
<a href="#">LeftScroll</a>	0x00000000 0x00000080	The scroll bar is located on the right side of the control. The scroll bar is located on the left side of the control.
<a href="#">BiDi</a>	0x000000E0	Set this value for a combination of the <a href="#">RTLRO</a> , <a href="#">RightAligned</a> , and <a href="#">LeftScroll</a> attributes.
<a href="#">MultiLine</a>	0x00010000	Creates a multiple line edit control with a vertical scroll bar. Include 65536 in the bit word in the Attributes column of the <a href="#">Control</a> to create a multiple line edit control with a vertical scroll bar.
<a href="#">Password</a>	0x00200000	Creates an edit control for entering passwords. Add 2097152 to the value in the Attributes column of the <a href="#">Control table</a> to create an edit control that displays each character as an asterisk (*) as they are typed into the control. Setting the Password Attribute prevents the installer from writing the property associated with the Edit control into the log file. For more information, see <a href="#">Preventing Confidential Information from Being Written into the Log File</a>

## Remarks

This control can be created from the EDIT class by using the **CreateWindowEx** function. It has the WS\_BORDER, WS\_CHILD, WS\_TABSTOP, and WS\_GROUP styles.

The length of text that can be entered can be limited by putting a number from 0 to 2147483646 in curly braces at the beginning of the Text field in the [Control table](#). For example, if the text field starts with {80}, the length of the string is limited at 80 characters. If no such limit is supplied in the table, or if 0 is specified, the length is set to the maximum possible (2147483646 characters). A negative or non-numeric value will generate an error.

For compatibility with screen readers, when authoring a dialog box with an Edit control as the first active control, you must make the text field belonging to the edit field the first active control in the [Dialog table](#). Since the static text cannot take focus, when the dialog box is created the edit field will have the focus initially as intended, but doing this ensures that screen readers show the correct information.

The property associated with the Edit control is only set when the control loses focus. Therefore you must tab from the control or select a different control for the property to be updated.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# GroupBox Control

The GroupBox control displays a rectangle, possibly with caption text, that serves to group other controls together on the dialog box.

## Control Attributes

You can use the following attributes with the GroupBox control. To change the value of an attribute using an event, subscribe the control to a ControlEvent in the [EventMapping table](#) and list the attribute's identifier in the Attribute column. Enter the identifier of the ControlEvent in the Event column.

Attribute identifier	Hexadecimal bit	Description
<a href="#">Position</a>		Position of control in the dialog box. Enter the control's width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the <a href="#">Control table</a> or <a href="#">BBControl table</a> . Use <a href="#">installer units</a> for length and distance.
<a href="#">Text</a>		Displays a caption in the upper left corner of the control. To set the font and font style of a text string, prefix the string of displayed characters with {style} or {&style}. Where style is an identifier listed in the TextStyle column of the <a href="#">TextStyle table</a> . If neither of these are present, but the <b>DefaultUIFont</b> property is defined as a valid text style, that font will be used.
<a href="#">Visible</a>	0x00000000 0x00000001	Hidden control. Visible control.  Include this bit in the bit word of the Attributes column in the <a href="#">Control table</a> or <a href="#">BBControl table</a> to make the control visible or

		hidden upon its creation. You can also hide or show a control by using the <a href="#">ControlCondition table</a> .
<a href="#">Sunken</a>	0x00000000 0x00000004	Displays the default visual style. Displays the control with a sunken, 3-D look. Include these bits in the bit word in the Attributes column of the <a href="#">Control table</a> .
<a href="#">RTLRO</a>	0x00000000 0x00000020	Text in the control is displayed in left-to-right reading order. Text in the control is displayed in right-to-left reading order.
<a href="#">RightAligned</a>	0x00000000 0x00000040	Text in the control is aligned to the left. Text in the control is aligned to the right.

## Remarks

This control can be created from the `BUTTON` class by using the **CreateWindowEx** function. It has the `BS_GROUPBOX`, `WS_CHILD`, and `WS_GROUP` styles.

There is always a gap between the top of the control's window and the visible frame, even when there is no caption.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Hyperlink Control

The Hyperlink control displays a HTML link to an address, which opens in the default browser for the computer. Links are not supported for protocols other than HTML.

**Windows Installer 4.5 or earlier:** Not supported. This Control is available beginning with Windows Installer 5.0.

The Text value of the HyperLink control uses the anchor <a> tag and the HREF attribute value to specify the URL and displayed text of the link.

```
<a href="http://www.blueyonderairlines.com">Blue Yonder Air
```

## Control Attributes

You can use the following attributes with the Hyperlink control. To change the value of an attribute using an event, subscribe the control to a ControlEvent in the [EventMapping table](#) and list the attribute's identifier in the Attribute column. Enter the identifier of the ControlEvent in the Event column.

Attribute identifier	Hexadecimal bit	Description
<a href="#">Position</a>		Position of control in the dialog box. Enter the control's width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the <a href="#">Control table</a> or <a href="#">BBControl table</a> . Use <a href="#">installer units</a> for length and distance.
<a href="#">Text</a>		Text displayed by the control. To set the font and font style of a text string, prefix the string of displayed characters with {\style} or {&style}. Where style is an identifier listed in the TextStyle column of the <a href="#">TextStyle table</a> . If neither of these are present, but the <a href="#">DefaultUIFont</a> property is defined as a valid

		text style, that font will be used. The text value will also resolve [Property] to the referenced property.
<b>Visible</b>	0x00000000 0x00000001	<p>Hidden control. Visible control.</p> <p>Include this bit in the bit word of the Attributes column in the <a href="#">Control table</a> or <a href="#">BBControl table</a>.to make the control visible or hidden upon its creation.</p> <p>You can also hide or show a control by using the <a href="#">ControlCondition table</a>.</p>
<b>Enabled</b>	0x00000000 0x00000002	<p>Control in a disabled state. Control in an enabled state.</p> <p>Include this bit in the bit word in the Attributes column of the <a href="#">Control</a> or <a href="#">BBControl tables</a> to enable the control on creation.</p> <p>You can also enable or disable a control by using the <a href="#">ControlCondition table</a>.</p>
<b>Sunken</b>	0x00000000 0x00000004	<p>Displays the default visual style. Displays the control with a sunken, 3-D, look.</p> <p>Include these bits in the bit word in the Attributes column of the <a href="#">Control table</a>.</p>
<b>Transparent</b>	0x00000000 0x00010000	<p>Opaque control. Background shows through control. The control has the WS_EX_TRANSPARENT style.</p> <p>Include this bit in the Attributes column of the <a href="#">Control</a> or <a href="#">BBControl tables</a>.</p>

## Remarks

This control can be created from the `WC_LINK` class by using the **CreateWindowEx** function. It has the `WS_CHILD`, `WS_TABSTOP` and `WS_GROUP` styles.

Do not place transparent [Text controls](#) on top of colored bitmaps. The text may not be visible if the user changes the display color scheme. For example, text may become invisible if the user sets the high contrast parameter for accessibility reasons.

If the text in the control is longer than the control width, the text wraps or truncates, depending on whether the height is sufficient to fit the wrapped text.

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Icon Control

The Icon control displays a static picture of an icon. The background of the image is transparent.

## Control Attributes

You can use the following attributes with this control. To change the value of an attribute using an event, subscribe the control to a ControlEvent in the [EventMapping table](#) and list the attribute's identifier in the Attribute column. Enter the identifier of the ControlEvent in the Event column.

Attribute identifier	Hexadecimal bit	Description
<a href="#">Position</a>		Position of control in the dialog box. Enter the control's width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the <a href="#">Control table</a> . Use <a href="#">installer units</a> for length and distance.
<a href="#">Text</a>		Contains the name of an icon stored in the <a href="#">Binary table</a> . To display an icon that is stored in the Binary table enter the name of the image's record appearing in the Binary table into the Text column of the <a href="#">Control table</a> record for this control.
<a href="#">Visible</a>	0x00000000 0x00000001	Hidden control. Visible control.  Include this bit in the bit word of the Attributes column in the <a href="#">Control table</a> to make the control visible or hidden upon its creation.  You can also hide or show a control by using the <a href="#">ControlCondition table</a> .
<a href="#">Sunken</a>	0x00000000 0x00000004	Displays the default visual style. Displays the control with a sunken, 3-D look.

		Include these bits in the bit word in the Attributes column of the <a href="#">Control table</a> .
<a href="#">FixedSize</a>	0x00000000 0x00100000	Stretches the icon image to fit the control. Crops or centers the icon image in the control.  Include this bit in the bit word of the Attributes column of the <a href="#">Control table</a> .
<a href="#">IconSize</a>	0x00000000 0x00200000 0x00400000 0x00600000	Loads the first image. Loads the first 16x16 image. Loads the first 32x32 image. Loads the first 48x48 image.  An icon file can contain different size images of the same icon. Include the value of the appropriate bit word in the Attributes column of the <a href="#">Control table</a>  If these bits are not set, the installer ignores the FixedSize attribute and the image is stretched to fit the control rectangle. If both the IconSize bits and the FixedSize bits are set, an image smaller than the control is centered and an image is larger than the control it is shrunk to fit.

## Remarks

This control can be created from the `STATIC` class by using the **CreateWindowEx** function. It has the `SS_ICON`, `SS_CENTERIMAGE`, `WS_CHILD`, and `WS_GROUP` styles.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Line Control

The Line control is a horizontal line.

## Control Attributes

You can use the following attributes with this control. To change the value of an attribute using an event, subscribe the control to a ControlEvent in the [EventMapping table](#) and list the attribute's identifier in the Attribute column. Enter the identifier of the ControlEvent in the Event column.

Attribute identifier	Hexadecimal bit	Description
<a href="#">Position</a>		Position of the control in the dialog box. Enter the control's width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the <a href="#">Control table</a> or <a href="#">BBControl table</a> . Use <a href="#">installer units</a> for length and distance.
<a href="#">Visible</a>	0x00000000 0x00000001	Hidden control. Visible control.  Include this bit in the bit word of the Attributes column in the <a href="#">Control table</a> or <a href="#">BBControl table</a> to make the control visible or hidden upon its creation.  You can also hide or show a control by using the <a href="#">ControlCondition table</a> .
<a href="#">Sunken</a>	0x00000000 0x00000004	Displays the default visual style. Displays the control with a sunken, 3-D look.  Include these bits in the bit word in the Attributes column of the <a href="#">Control table</a> .

## Remarks

This control can be created from the `STATIC` class by using the **CreateWindowEx** function. It has the `SS_ETCHEDHORZ` and `SS_SUNKEN` styles.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ListBox Control

The ListBox control is a regular list box that enables the user to make a single selection from a list of predetermined values. The possible values are read from the [Listbox table](#). You can associate a string or integer property by entering the property's name in the Property column of the [Control table](#).

## Control Attributes

You can use the following attributes with this control. To change the value of an attribute using an event, subscribe the control to a ControlEvent in the [EventMapping table](#) and list the attribute's identifier in the Attribute column. Enter the identifier of the ControlEvent in the Event column.

<b>Attribute identifier</b>	<b>Hexadecimal bit</b>	<b>Description</b>
<a href="#">IndirectPropertyName</a>		This is the name of an indirect property associated with the control. If the Indirect attribute bit is set, the control displays or changes the value of the property having this name. If the Indirect attribute bit is set, this name is also the value of the property listed in the Property column of the <a href="#">Control table</a> .
<a href="#">Position</a>		Position of the control in the dialog box. Enter the control's width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the <a href="#">Control table</a> . Use <a href="#">installer units</a> for length and distance.
<a href="#">PropertyName</a>		This is the name of the property associated with this control. If the Indirect attribute bit is not set, the

		control displays or changes the value of the property having this name. This attribute is specified in the Property column of the <a href="#">Control table</a> .
<a href="#">PropertyValue</a>		Current value of the property displayed or changed by this control. If the Indirect attribute bit is not set, this is the value of PropertyName. If the Indirect attribute bit is set, this is the value of IndirectPropertyName. If the attribute changes, the control reflects the new value.
<a href="#">Text</a>		Text displayed by screen readers. Enter the text to be displayed into the Text column of the <a href="#">Control table</a> . To set the font and font style of a text string, prefix the string of displayed characters with {\style} or {&style}. Where style is an identifier listed in the TextStyle column of the <a href="#">TextStyle table</a> . If neither of these are present, but the <b>DefaultUIFont</b> property is defined as a valid text style, that font will be used.
<a href="#">Visible</a>	0x00000000 0x00000001	Hidden control. Visible control.  Include this bit in the bit word of the Attributes column in the <a href="#">Control table</a> to make the control visible or hidden upon its creation.  You can also hide or show a control by using the <a href="#">ControlCondition table</a> .
<a href="#">Enabled</a>	0x00000000 0x00000002	Control in a disabled state. Control in an enabled state.  Include this bit in the bit word in the

		<p>Attributes column of the <a href="#">Control</a> to enable the control on creation.</p> <p>You can also enable or disable a control by using the <a href="#">ControlCondition table</a>.</p>
<a href="#">Sunken</a>	<p>0x00000000 0x00000004</p>	<p>Displays the default visual style. Displays the control with a sunken, 3-D, look.</p> <p>Include these bits in the bit word in the Attributes column of the <a href="#">Control table</a>.</p>
<a href="#">Indirect</a>	<p>0x00000000 0x00000008</p>	<p>The control displays or changes the value of the property in the Property column of the <a href="#">Control table</a>. The control displays or changes the value of the property that has the identifier listed in the Property column of the Control table.</p> <p>Determines if the property associated with this control is referenced indirectly.</p>
<a href="#">Integer</a>	<p>0x00000000 0x00000010</p>	<p>Property associated with the control is a string value. Property associated with the control is an integer value.</p> <p>Include this bit in the bit word of the Attributes column of the <a href="#">Control table</a> to set this attribute on creation of the control.</p>
<a href="#">RTLRO</a>	<p>0x00000000 0x00000020</p>	<p>Text in the control is displayed in left-to-right reading order. Text in the control is displayed in right-to-left reading order.</p>

<a href="#">RightAligned</a>	0x00000000 0x00000040	Text in the control is aligned to the left. Text in the control is aligned to the right.
<a href="#">LeftScroll</a>	0x00000000 0x00000080	The scroll bar is located on the right side of the control. The scroll bar is located on the left side of the control.
<a href="#">BiDi</a>	0x000000E0	Set this value for a combination of the <a href="#">RTLRO</a> , <a href="#">RightAligned</a> , and <a href="#">LeftScroll</a> attributes.
<a href="#">Sorted</a>	0x00000000 0x00010000	Items displayed in alphabetical order. Items displayed in order specified in the <a href="#">ListView table</a> .  Include this bit in the bit word in the Attributes column to display items in the order specified by the Order column of the ListView table.
<a href="#">UsersLanguage</a>	0x00000000 0x00100000	Fonts created in the database code page. Fonts created in the user's default UI code page.

## Remarks

This control can be created from the LISTBOX class by using the **CreateWindowEx** function. It has the WS\_TABSTOP, WS\_GROUP, and WS\_CHILD styles. If the Sorted control style bit is on, the control is created with the LBS\_NOTIFY, WS\_VSCROLL, and WS\_BORDER styles, otherwise, the control is created with the LBS\_STANDARD style.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Listview Control

The Listview control that displays a single column of values with an icon next to each item. It enables the user to select a single string or integer value for a property from a predetermined list. The possible values are read from the [Listview table](#). You can associate the control with an integer or string value by entering the property's name into the Property column of the [Control table](#).

The contents of the Text field of the [Control table](#) is never displayed by the Listview control. Instead this field specifies the style of text to be displayed by the control and contains a description of the control used by screen review utilities. To set the font and font style of a text string, prefix the string of displayed characters with `{style}` or `&{style}`. Where style is an identifier listed in the TextStyle column of the [TextStyle table](#). If neither of these are present, but the [DefaultUIFont](#) property is defined as a valid text style, that font will be used. The information following this is read by screen review utilities as the description of the control. See [Accessibility](#).

## Control Attributes

You can use the following attributes with this control. To change the value of an attribute using an event, subscribe the control to a ControlEvent in the [EventMapping table](#) and list the attribute's identifier in the Attribute column. Enter the identifier of the ControlEvent in the Event column.

<b>Attribute identifier</b>	<b>Hexadecimal bit</b>	<b>Description</b>
<a href="#">IndirectPropertyName</a>		This is the name of an indirect property associated with the control. If the Indirect attribute bit is set, the control displays or changes the value of the property having this name. If the Indirect attribute bit is set, this name is also the value of the property listed in the Property column of the <a href="#">Control table</a> .
<a href="#">Position</a>		Position of the control in the dialog

		<p>box.</p> <p>Enter the control's width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the <a href="#">Control table</a>. Use <a href="#">installer units</a> for length and distance.</p>
<a href="#">PropertyName</a>		<p>This is the name of the property associated with this control. If the Indirect attribute bit is not set, the control displays or changes the value of the property having this name. This attribute is specified in the Property column of the <a href="#">Control table</a>.</p>
<a href="#">PropertyValue</a>		<p>Current value of the property displayed or changed by this control. If the Indirect attribute bit is not set, this is the value of PropertyName. If the Indirect attribute bit is set, this is the value of IndirectPropertyName. If the attribute changes, the control reflects the new value.</p>
<a href="#">Text</a>		<p>To display text in screen readers, enter the text into the Text column of the <a href="#">Control table</a>. See <a href="#">Accessibility</a>.</p>
<a href="#">Visible</a>	<p>0x00000000 0x00000001</p>	<p>Hidden control. Visible control.</p> <p>Include this bit in the bit word of the Attributes column in the <a href="#">Control table</a> to make the control visible or hidden upon its creation.</p> <p>You can also hide or show a control by using the <a href="#">ControlCondition table</a>.</p>
<a href="#">Enabled</a>	<p>0x00000000 0x00000002</p>	<p>Control in a disabled state. Control in an enabled state.</p>

		<p>Include this bit in the bit word in the Attributes column of the <a href="#">Control</a> to enable the control on creation.</p> <p>You can also enable or disable a control by using the <a href="#">ControlCondition table</a>.</p>
<a href="#">Sunken</a>	<p>0x00000000 0x00000004</p>	<p>Displays the default visual style. Displays the control with a sunken, 3-D, look.</p> <p>Include these bits in the bit word in the Attributes column of the <a href="#">Control table</a>.</p>
<a href="#">Indirect</a>	<p>0x00000000 0x00000008</p>	<p>The control displays or changes the value of the property in the Property column of the <a href="#">Control table</a>. The control displays or changes the value of the property that has the identifier listed in the Property column of the Control table.</p> <p>Determines if the property associated with this control is referenced indirectly.</p>
<a href="#">Integer</a>	<p>0x00000000 0x00000010</p>	<p>Property associated with the control is a string value. Property associated with the control is an integer value.</p> <p>Include this bit in the bit word of the Attributes column of the <a href="#">Control table</a> to set this attribute on creation of the control.</p>
<a href="#">RTLRO</a>	<p>0x00000000 0x00000020</p>	<p>Text in the control is displayed in left-to-right reading order. Text in the control is displayed in</p>

		right-to-left reading order.
<a href="#">RightAligned</a>	0x00000000 0x00000040	Text in the control is aligned to the left. Text in the control is aligned to the right.
<a href="#">LeftScroll</a>	0x00000000 0x00000080	The scroll bar is located on the right side of the control. The scroll bar is located on the left side of the control.
<a href="#">BiDi</a>	0x000000E0	Set this value for a combination of the <a href="#">RTLRO</a> , <a href="#">RightAligned</a> , and <a href="#">LeftScroll</a> attributes.
<a href="#">Sorted</a>	0x00000000 0x00010000	Items displayed in alphabetical order. Items displayed in order specified in the <a href="#">ListView table</a> .  Include this bit in the bit word in the Attributes column to display items in the order specified by the Order column of the ListView table.
<a href="#">FixedSize</a>	0x00000000 0x00100000	Stretches the icon image to fit the control. Crops or centers the icon image in the control.  Include this bit in the bit word of the Attributes column of the <a href="#">Control table</a> .
<a href="#">IconSize</a>	0x00000000 0x00200000 0x00400000 0x00600000	Loads the first image. Loads the first 16x16 image. Loads the first 32x32 image. Loads the first 48x48 image.  An icon file can contain different size

	<p>images of the same icon. Include the value of the appropriate bit word in the Attributes column of the <a href="#">Control table</a></p> <p>If these bits are not set, the installer ignores the FixedSize attribute and the image is stretched to fit the control rectangle. If both the IconSize bits and the FixedSize bits are set, an image smaller than the control is centered and an image is larger than the control it is shrunk to fit.</p>
--	---

## Remarks

This control can be created from the WC\_LISTVIEW class by using the **CreateWindowEx** function. It has the LVS\_REPORT, LVS\_NOCOLUMNHEADER, WS\_VSCROLL, WS\_HSCROLL, LVS\_SHAREIMAGELISTS, LVS\_SINGLESEL, LVS\_SHOWSELALWAYS, WS\_BORDER, and WS\_TABSTOP styles. If the Sorted style bit is not on, the control also has the LVS\_SORTASCENDING style.

For compatibility with screen readers, when authoring a dialog with an ListView control as the first active control, you must make the text field belonging to the edit field the first active control in the [Dialog table](#). Since the static text cannot take focus, when the dialog is created the edit field will have the focus initially as intended, but doing this ensures that screen readers show the correct information.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MaskedEdit Control

The MaskedEdit Control is an edit field control that contains a mask in the text field of the control. You can associate the control with a string value property by entering the property name into the Property column of the [Control Table](#).

You can use the MaskedEdit Control to create a template for user entry of information such as a telephone number or Product ID code. For example, the **PIDKEY** Property can be entered by the user through a MaskedEdit Control that is specified by setting the **PIDTemplate** Property to a string like the following:

```
12345<### -%%%%%%%%>@@@@@
```

The string defines the masking template for the entry of the **PIDKEY** Property by the user. The visible segment of the string is enclosed by a pair of bracket (<>) characters.

The following table identified the syntax of the mask.

Character Meaning	
<	The left end of the visible segment of the template. This character and everything to its left are hidden in the user interface. There should be no more than one instance of this character in the template.
>	The right end of the visible segment of the template. This character and everything to its right are hidden in the user interface. This character is replaced by a dash during validation. If there is a visible segment begins with <, then it must be terminated with a matching >.
#	This character can be a digit (numeral.)
%	This character can be an alternate digit (numeral) that enables the mask to control the way a custom action differentiates fields.
@	This character can be a random digit (numeral.) This character should not appear in the visible part of the template.
&	This character can be any character.

^	This character can be an alternate character that enables the mask to control the way a custom action differentiates fields.
?	This character can be an alternate character that enables the mask to control the way a custom action differentiates fields.
`	Grave accent marks ` (ASCII value 96) can represent an alternate character that enables the mask to control the way a custom action differentiates fields.
_	This character is a literal underscore character.
=	This character is the field terminator. This must follow a #, %, ^, or `. This creates one more input position of the same type as the preceding positions and terminates the field with a '-' separator.

Any other character is treated as a literal constant.

For characters that can be edited, the control creates separate edit windows with one window for each block of contiguous characters of the same kind.

## Control Attributes

To change the value of an attribute that is using an event, subscribe the control to a Control event in the [EventMapping Table](#) and list the attribute identifier in the Attribute column. Enter the identifier of the Control event in the Event column. You can use the following attributes with the MaskedEdit Control.

<b>Attribute</b>	<b>Hexadecimal Bit</b>	<b>Description</b>
<a href="#">IndirectPropertyName</a>		This is the name of an indirect property that is associated with the control. If the indirect attribute bit is set, the control displays or changes the value of the property that has this name. If the indirect attribute bit is set, this name is also the value of the property that is listed in the Property

		column of the <a href="#">Control Table</a> .
<a href="#">Position</a>		<p>Position of the control in the dialog box.</p> <p>Enter the control width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the <a href="#">Control Table</a>. Use <a href="#">Installer Units</a> for length and distance.</p>
<a href="#">PropertyName</a>		<p>This is the name of the property that is associated with this control. If the indirect attribute bit is not set, the control displays or changes the value of the property that has this name. This attribute is specified in the Property column of the <a href="#">Control Table</a>.</p>
<a href="#">PropertyValue</a>		<p>Current value of the property that is displayed or changed by this control. If the indirect attribute bit is not set, this is the value of PropertyName. If the indirect attribute bit is set, this is the value of IndirectPropertyName. If the attribute changes, the control reflects the new value.</p>
<a href="#">Text</a>		<p>To set the font and font style of a text string, prefix the string of displayed characters with {\style} or {&amp;style}. Where style is an identifier listed in the Style column of the <a href="#">TextStyle Table</a>. If neither of these are present, but the <b>DefaultUIFont</b> Property is defined as a valid text style, that font is used. The string that specifies the masking template follows this prefix and uses the syntax described previously in this topic.</p>

Visible	0x00000000 0x00000001	<p>Hidden control. Visible control.</p> <p>Include this bit in the bit word of the Attributes column in the <a href="#">Control Table</a> to make the control visible or hidden when it is created.</p> <p>You can also hide or show a control by using the <a href="#">ControlCondition Table</a>.</p>
Enabled	0x00000000 0x00000002	<p>Control in a disabled state. Control in an enabled state.</p> <p>Include this bit in the bit word in the Attributes column of the <a href="#">Control Table</a> to enable the control on creation.</p> <p>You can also enable or disable a control by using the <a href="#">ControlCondition Table</a>.</p>
Sunken	0x00000000 0x00000004	<p>Displays the default visual style. Displays the control with a sunken, 3-D look.</p> <p>Include these bits in the bit word in the Attributes column of the <a href="#">Control Table</a>.</p>
Indirect	0x00000000 0x00000008	<p>The control displays or changes the value of the property in the Property column of the <a href="#">Control Table</a>.</p> <p>The control displays or changes the value of the property that has the identifier listed in the Property column of the <a href="#">Control Table</a>.</p> <p>Determines if the property that is associated with this control is referenced indirectly.</p>

## Remarks

The MaskedEdit Control creates one parent window of the **BUTTON** class with the BS\_OWNERDRAW and WS\_EX\_CONTROLPARENT styles. It creates several child windows to this window.

- For constant text parts, it creates STATIC windows with the SS\_LEFT and WS\_CHILD styles.
- For editable fields, it creates an EDIT window with the WS\_CHILD, WS\_BORDER, and WS\_TABSTOP styles.
- For numeric fields, the window also has the ES\_NUMBER style.

The alternate digit, %, and alternate alphanumeric characters, ^, ?, and ` fields allow custom actions to differentiate between fields in a way that can be controlled by the mask, for example, ^ can be used for fields that should be uppercase.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PathEdit Control

The PathEdit control displays an edit field that enables a user to select the end section of a path. This control supports entering the selected folder name or the entire path in the edit field. A user can also enter a Universal Naming Convention (UNC) path to a drive that has no drive letter. If the user enters an end segment for the path that is invalid for the present volume, PathEdit control cannot transfer the focus to the next control.

The PathEdit control, [DirectoryCombo](#), and [DirectoryList](#) controls are associated with the same string valued property. That property is the path selected by the user. Enter the property's name into the Property column of the [Control table](#). This property must have an initial value containing at least a one volume and one sublevel. Specify the initial value for the property in the Value column of the [Property table](#).

This control is intended to be used on a [Browse Dialog](#) together with the PathEdit and DirectoryList controls.

## Control Attributes

You can use the following attributes with this control. To change the value of an attribute using an event, subscribe the control to a ControlEvent in the [EventMapping table](#) and list the attribute's identifier in the Attribute column. Enter the identifier of the ControlEvent in the Event column.

Attribute identifier	Hexadecimal bit	Description
<a href="#">IndirectPropertyName</a>		This is the name of an indirect property associated with the control. If the Indirect attribute bit is set, the control displays or changes the value of the property having this name. If the Indirect attribute bit is set, this name is also the value of the property listed in the Property column of the <a href="#">Control table</a> .

Position		<p>Position of the control in the dialog box.</p> <p>Enter the control's width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the <a href="#">Control table</a>. Use <a href="#">installer units</a> for length and distance.</p>
PropertyName		<p>This is the name of the property associated with this control. If the Indirect attribute bit is not set, the control displays or changes the value of the property having this name. This attribute is specified in the Property column of the <a href="#">Control table</a>.</p>
PropertyValue		<p>Current value of the property displayed or changed by this control. If the Indirect attribute bit is not set, this is the value of PropertyName. If the Indirect attribute bit is set, this is the value of IndirectPropertyName. If the attribute changes, the control reflects the new value.</p>
Text		<p>To set the font and font style of a text string, prefix the string of displayed characters with {\style} or {&amp;style}. Where style is an identifier listed in the TextStyle column of the <a href="#">TextStyle table</a>. If neither of these are present, but the <b>DefaultUIFont</b> property is defined as a valid text style, that font will be used.</p> <p>To specify the number of characters the user can enter, append {n} after any font specifications, where n is a positive integer.</p>

Visible	0x00000000 0x00000001	<p>Hidden control. Visible control.</p> <p>Include this bit in the bit word of the Attributes column in the <a href="#">Control table</a> to make the control visible or hidden upon its creation.</p> <p>You can also hide or show a control by using the <a href="#">ControlCondition table</a>.</p>
Enabled	0x00000000 0x00000002	<p>Control in a disabled state. Control in an enabled state.</p> <p>Include this bit in the bit word in the Attributes column of the <a href="#">Control</a> to enable the control on creation.</p> <p>You can also enable or disable a control by using the <a href="#">ControlCondition table</a>.</p>
Sunken	0x00000000 0x00000004	<p>Displays the default visual style. Displays the control with a sunken, 3-D, look.</p> <p>Include these bits in the bit word in the Attributes column of the <a href="#">Control table</a>.</p>
Indirect	0x00000000 0x00000008	<p>The control displays or changes the value of the property in the Property column of the <a href="#">Control table</a>.</p> <p>The control displays or changes the value of the property that has the Identifier listed in the Property column of the Control table.</p> <p>Determines if the property associated with this control is referenced indirectly.</p>

<a href="#">RTLRO</a>	0x00000000 0x00000020	Text in the control is displayed in left-to-right reading order. Text in the control is displayed in right-to-left reading order.
<a href="#">RightAligned</a>	0x00000000 0x00000040	Text in the control is aligned to the left. Text in the control is aligned to the right.

## Remarks

The PathEdit control is derived from the [Edit](#) control.

For compatibility with screen readers, when authoring a dialog box with a PathEdit control as the first active control, you must make the text field belonging to the edit field the first active control in the [Dialog table](#). Since the static text cannot take focus, when the dialog box is created, the edit field will have the focus initially as intended; this ensures that screen readers show the correct information.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ProgressBar Control

The ProgressBar control displays a bar graph that changes length as it receives progress messages. This control subscribes to the [SetProgress ControlEvent](#). It can subscribe to a ControlEvent named after the action being monitored.

For related information, see [Authoring a ProgressBar Control](#), and [Adding Custom Actions to the ProgressBar](#).

## Control Attributes

You can use the following attributes with this control. To change the value of an attribute using an event, subscribe the control to a ControlEvent in the [EventMapping table](#) and list the attribute's identifier in the Attribute column. Enter the identifier of the ControlEvent in the Event column.

Attribute identifier	Hexadecimal bit	Description
<a href="#">Position</a>		Position of the control in the dialog box. Enter the control's width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the <a href="#">Control table</a> . Use <a href="#">installer units</a> for length and distance.
<a href="#">Progress</a>		<p>This attribute specifies how much of the ProgressBar is filled. The attribute is composed of two integers and a string. The first integer field is the current number of progress ticks and the second integer field is the default maximum number of progress ticks (1024). The third field is a string that is the name of the action in progress. If the current number of progress ticks is larger than the maximum, the installer changes it to the maximum.</p> <p>This attribute is set and changed by the <a href="#">SetProgress ControlEvent</a>. You must subscribe the control to this event in the <a href="#">EventMapping</a></p>

		<p><a href="#">table</a> by entering SetProgress into the Event column and Progress into the Attribute column.</p>
<a href="#">Text</a>		<p>Text displayed by the control.</p> <p>To set the font and font style of a text string, prefix the string of displayed characters with {\style} or {&amp;style}. Where style is an identifier listed in the TextStyle column of the <a href="#">TextStyle table</a>. If neither of these are present, but the <b>DefaultUIFont</b> property is defined as a valid text style, that font will be used.</p>
<a href="#">Visible</a>	<p>0x00000000 0x00000001</p>	<p>Hidden control. Visible control.</p> <p>Include this bit in the bit word of the Attributes column in the <a href="#">Control table</a> to make the control visible or hidden upon its creation.</p> <p>You can also hide or show a control by using the <a href="#">ControlCondition table</a>.</p>
<a href="#">Sunken</a>	<p>0x00000000 0x00000004</p>	<p>Displays the default visual style. Displays the control with a sunken, 3-D, look.</p> <p>Include these bits in the bit word in the Attributes column of the <a href="#">Control table</a>.</p>
<a href="#">RTLRO</a>	<p>0x00000000 0x00000020</p>	<p>Text in the control is displayed in left-to-right reading order. Text in the control is displayed in right-to-left reading order.</p>
<a href="#">Progress95</a>	<p>0x00000000 0x00010000</p>	<p>Progress bar drawn as a continuous bar. Progress bar drawn as a series of rectangles.</p> <p>Include these bits in the bit word in the Attributes column of the <a href="#">Control table</a>.</p>

## Remarks

This control can be created from the `PROGRESS_CLASS` class by using the **CreateWindowEx** function. It has the `WS_CHILD` and `WS_GROUP` styles.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PushButton Control

The PushButton control displays a basic push button, also known as a command button.

PushButton controls cannot be used to publish an [IgnoreChange ControlEvent](#), [SelectionDescription ControlEvent](#), [SelectionSize ControlEvent](#), [SelectionPath ControlEvent](#), [SelectionPathOn ControlEvent](#), [SelectionAction ControlEvent](#), [SelectionNoItems ControlEvent](#), [ActionText ControlEvent](#), [ActionData ControlEvent](#), [SetProgress ControlEvent](#), or [TimeRemaining ControlEvent](#).

You can use the following attributes with this control. To change the value of an attribute using an event, subscribe the control to a ControlEvent in the [EventMapping table](#) and list the attribute's identifier in the Attribute column. Enter the identifier of the ControlEvent in the Event column.

<b>Attribute identifier</b>	<b>Hexadecimal bit</b>	<b>Description</b>
<a href="#">Position</a>		Position of control in the dialog box. Enter the control's width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the <a href="#">Control table</a> . Use <a href="#">installer units</a> for length and distance.
<a href="#">Text</a>		This control attribute can specify the text displayed by the control, an image stored in the Binary table, or an image set at run time. To specify text, enter the text string into the Text column of the Control table. To set the font and font style of a text string, prefix the string of displayed characters with <code>{\style}</code> or <code>{&amp;style}</code> . Where style is an identifier listed in the TextStyle column of the <a href="#">TextStyle table</a> . If neither of these are present, but the <b>DefaultUIFont</b> property is defined as a valid text style, that font will be used.

		To specify an icon or bitmap image stored in the <a href="#">Binary table</a> , enter the primary key of the image's record from the Name column of the Binary table into the Text column of the <a href="#">Control table</a> record for the control.
<a href="#">Visible</a>	0x00000000 0x00000001	Hidden control. Visible control.  Include this bit in the bit word of the Attributes column in the <a href="#">Control table</a> to make the control visible or hidden upon its creation.  You can also hide or show a control by using the <a href="#">ControlCondition table</a> .
<a href="#">Enabled</a>	0x00000000 0x00000002	Control in a disabled state. Control in an enabled state.  Include this bit in the Attributes column of the <a href="#">Control</a> to enable the control on creation.  You can also enable or disable a control by using the <a href="#">ControlCondition table</a> .
<a href="#">Sunken</a>	0x00000000 0x00000004	Displays the default visual style. Displays the control with a sunken, 3-D, look.  Include these bits in the bit word in the Attributes column of the <a href="#">Control table</a> .
<a href="#">RTLRO</a>	0x00000000 0x00000020	Text in the control is displayed in left-to-right reading order. Text in the control is displayed in right-to-left reading order.
<a href="#">Bitmap</a>	0x00000000 0x00040000	Text in the control is specified in the Text column of the <a href="#">Control table</a> . The button has the BS_BITMAP style, text

		<p>in the control is replaced by a bitmap image. The Text column in the Control table is used as a foreign key to the <a href="#">Binary table</a>.</p> <p>Include this bit in the bit word in the Attributes column of the <a href="#">Control</a>.</p> <p>Do not set the Icon and Bitmap style bits simultaneously. The button cannot contain both a bitmap image and text.</p> <p>To set the font and font style of a text string, prefix the string of displayed characters with <code>{\style}</code> or <code>{&amp;style}</code>. Where style is an identifier listed in the TextStyle column of the TextStyle table. If neither of these are present, but the <b>DefaultUIFont</b> property is defined as a valid text style, that font will be used.</p>
<p><a href="#">Icon Control</a></p>	<p>0x00000000 0x00080000</p>	<p>Text in the control is specified in the Text column of the <a href="#">Control table</a>.</p> <p>The button has the BS_ICON style, text in the control is replaced by an icon image. The Text column in the Control table is used as a foreign key to the <a href="#">Binary table</a>.</p> <p>Include this bit in the bit word in the Attributes column of the <a href="#">Control</a>.</p> <p>Do not set the Icon and Bitmap bits simultaneously. The button cannot contain both an icon image and text.</p> <p>To set the font and font style of a text string, prefix the string of displayed characters with <code>{\style}</code> or <code>{&amp;style}</code>. Where style is an identifier listed in the TextStyle column of the TextStyle table. If neither of these are present, but the <b>DefaultUIFont</b> property is defined as a valid text style, that font will be used.</p>

<a href="#">FixedSize</a>	0x00000000 0x00100000	Stretch the icon image to fit the control. Crop or center the icon image in the control.  Include this bit in the bit word of the Attributes column of the <a href="#">BBControl table</a> or the <a href="#">Control table</a> .
<a href="#">IconSize</a>	0x00000000 0x00200000 0x00400000 0x00600000	Loads the first image. Loads the first 16x16 image. Loads the first 32x32 image. Loads the first 48x48 image.  An icon file can contain different size images of the same icon. Include the value of the appropriate bit word in the Attributes column of the <a href="#">Control table</a>  If these bits are not set, the installer ignores the FixedSize attribute and the image is stretched to fit the control rectangle. If both the IconSize bits and the FixedSize bits are set, an image smaller than the control is centered and an image is larger than the control it is reduced to fit.
<a href="#">ElevationShield</a>	0x00000000 0x00800000	The appearance of pushbutton is determined by the other icon attributes. Adds the <a href="#">User Account Control</a> (UAC) elevation icon (shield icon) to the pushbutton control.

## Remarks

This control can be created from the `BUTTON` class by using the `CreateWindowEx` function. It has the `BS_MULTILINE`, `WS_CHILD`, `WS_TABSTOP`, and `WS_GROUP` styles.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RadioButtonGroup Control

The RadioButtonGroup control is a group of radio buttons. It enables the user to select a string or integer value for a property from a predetermined list of values. You can associate this control with a property by entering the property's name into the Property column of the [Control table](#). Specify the possible values for selection in the Value column of the [RadioButton table](#). Note that the string displayed is not necessarily the same as the value that the user is choosing.

Every RadioButtonGroup control is associated with a property. The default value for this property must be initialized in the [Property table](#). Within each RadioButtonGroup specified in the [RadioButton table](#), there may be one radio button that has a value in the Value field that matches the default value for this property. This is the default button for the RadioButtonGroup control. The **Default** button is initially shown as selected in the control.

Note that the focus on a dialog box cannot move to a RadioButtonGroup control until one of the buttons in the group has been selected. To make the focus move to this button group, specify one of the buttons as a default button for the group.

RadioButtonGroup controls only set property values and cannot be used to send a [ControlEvent](#).

The implementation treats the whole group as one control, therefore it is not possible to hide or disable individual buttons within the group. Similarly all the buttons have to be of the same style, that is, either all of them have text or all of them have bitmaps (or other similar features). The position of the buttons is authored relative to the group. This way the entire group can be moved by changing only the coordinates of the group without changing the individual buttons. At creation the control verifies that the individual buttons do not extend beyond the boundaries of the group.

## Control Attributes

You can use the following attributes with this control. To change the value of an attribute using an event, subscribe the control to a [ControlEvent](#) in

the [EventMapping table](#) and list the attribute's identifier in the Attribute column. Enter the identifier of the ControlEvent in the Event column.

Attribute identifier	Hexadecimal bit	Description
<a href="#">IndirectPropertyName</a>		This is the name of an indirect property associated with the control. If the Indirect attribute bit is set, the control displays or changes the value of the property having this name. If the Indirect attribute bit is set, this name is also the value of the property listed in the Property column of the <a href="#">Control table</a> .
<a href="#">Position</a>		Position of the control in the dialog box. Enter the control's width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the <a href="#">Control table</a> . Use <a href="#">installer units</a> for length and distance.
<a href="#">PropertyName</a>		This is the name of the property associated with this control. If the Indirect attribute bit is not set, the control displays or changes the value of the property having this name. This attribute is specified in the Property column of the <a href="#">Control table</a> .
<a href="#">PropertyValue</a>		Current value of the property displayed or changed by this control. If the Indirect attribute bit is not set, this is the value of PropertyName. If the Indirect attribute bit is set, this is the value of IndirectPropertyName. If the attribute changes, the control reflects the new value.

Text		<p>This control attribute can specify the text displayed by the control, an image stored in the Binary table, or an image set at run time.</p> <p>To specify text, enter the text string into the Text column of the <a href="#">Control table</a>. To set the font and font style of this text, prefix the string of displayed characters with {\style} or {&amp;style}. Where style is an identifier listed in the TextStyle column of the <a href="#">TextStyle table</a>. If neither of these are present, but the <b>DefaultUIFont</b> property is defined as a valid text style, that font will be used.</p> <p>To specify an icon or bitmap image stored in the <a href="#">Binary table</a> enter the primary key of the image's record from the Name column of the Binary table into the Text column of the <a href="#">Control table</a> record for the control.</p>
Visible	0x00000000 0x00000001	Hidden control. Visible control.  Include this bit in the bit word of the Attributes column in the <a href="#">Control table</a> to make the control visible or hidden upon its creation.  You can also hide or show a control by using the <a href="#">ControlCondition table</a> .
Enabled	0x00000000 0x00000002	Control in a disabled state. Control in an enabled state.  Include this bit in the Attributes column of the <a href="#">Control</a> to enable the control on creation.  You can also enable or disable a

		control by using the <a href="#">ControlCondition table</a> .
<a href="#">Sunken</a>	0x00000000 0x00000004	Displays the default visual style. Displays the control with a sunken, 3-D, look.  Include these bits in the bit word in the Attributes column of the <a href="#">Control table</a> .
<a href="#">Indirect</a>	0x00000000 0x00000008	The control displays or changes the value of the property in the Property column of the <a href="#">Control table</a> . The control displays or changes the value of the property that has the identifier listed in the Property column of the Control table.  Determines if the property associated with this control is referenced indirectly.
<a href="#">Integer</a>	0x00000000 0x00000010	Property associated with the control is a string value. Property associated with the control is an integer value.  Include this bit in the bit word of the Attributes column of the <a href="#">Control table</a> to set this attribute on creation of the control.
<a href="#">RTLRO</a>	0x00000000 0x00000020	Text in the control is displayed in left-to-right reading order. Text in the control is displayed in right-to-left reading order.
<a href="#">RightAligned</a>	0x00000000 0x00000040	Text in the control is aligned to the left.

		Text in the control is aligned to the right.
<a href="#">PushLike</a>	0x00000000 0x00020000	<p>Control is drawn with its usual appearance.</p> <p>Control has the BS_PUSHLIKE style, and is drawn to appear as a push button.</p> <p>Include this bit in the bit word of the Attributes column of the <a href="#">Control table</a> to set this attribute on creation of the control.</p>
<a href="#">Bitmap</a>	0x00000000 0x00040000	<p>Text in the control is specified in the Text column of the <a href="#">Control table</a>. The control has the BS_BITMAP style, text in the control is replaced by a bitmap image. The Text column in the <a href="#">Control table</a> is used as a foreign key to the <a href="#">Binary table</a>.</p> <p>Include this bit in the bit word in the Attributes column of the <a href="#">Control table</a>.</p> <p>Do not set the Icon and Bitmap style bits simultaneously. The button cannot contain both a bitmap image and text.</p> <p>To set the font and font style of a text string, prefix the string of displayed characters with {\style} or {&amp;style}. Where style is an identifier listed in the TextStyle column of the TextStyle table. If neither of these are present, but the <a href="#">DefaultUIFont</a> property is defined as a valid text style, that font will be used.</p>

<p>Icon</p>	<p>0x00000000 0x00080000</p>	<p>Text in the control is specified in the Text column of the <a href="#">Control table</a>. The control has the BS_ICON style, text in the control is replaced by an icon image. The Text column in the <a href="#">Control table</a> is used as a foreign key to the <a href="#">Binary table</a>.</p> <p>Include this bit in the bit word in the Attributes column of the <a href="#">Control table</a>.</p> <p>Do not set the Icon and Bitmap bits simultaneously. The button cannot contain both an icon image and text.</p> <p>To set the font and font style of a text string, prefix the string of displayed characters with {\style} or {&amp;style}. Where style is an identifier listed in the TextStyle column of the TextStyle table. If neither of these are present, but the <b>DefaultUIFont</b> property is defined as a valid text style, that font will be used.</p>
<p>FixedSize</p>	<p>0x00000000 0x00100000</p>	<p>Stretch the icon image to fit the control. Crop or center the icon image in the control.</p> <p>Include this bit in the bit word of the Attributes column of the <a href="#">Control table</a>.</p>
<p>IconSize</p>	<p>0x00000000 0x00200000 0x00400000 0x00600000</p>	<p>Loads the first image. Loads the first 16x16 image. Loads the first 32x32 image. Loads the first 48x48 image.</p> <p>An icon file can contain different size</p>

		<p>images of the same icon. Include the value of the appropriate bit word in the Attributes column of the <a href="#">Control table</a></p> <p>If these bits are not set, the installer ignores the FixedSize attribute and the image is stretched to fit the control rectangle. If both the IconSize bits and the FixedSize bits are set, an image smaller than the control is centered and an image is larger than the control it is shrunk to fit.</p>
<a href="#">HasBorder</a>	<p>not set 0x01000000</p>	<p>No border and no text. Displays border and text.</p> <p>Include 16777216 in the bit word in the Attributes column of the <a href="#">Control</a> to display a border and text.</p>

## Remarks

Because of the way Windows draws the frame, there is a gap between the top of the control's window and the visible frame, even when there is no caption.

This control can be created from the `BUTTON` class by using the **CreateWindowEx** function. If the [HasBorder](#) bit is set, it has the `BS_GROUPBOX` style, otherwise it has the `BS_OWNERDRAW` style.

The `RadioButtonGroup` control should not overlap other controls and other controls should not overlap a `RadioButtonGroup`. Overlapping this control and another can cause the controls to function or display incorrectly. To provide screen-reader programs with extra descriptive text about a `RadioButtonGroup` control, follow the example provided in [Adding Extra Text to Radio Buttons](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ScrollableText Control

This control displays a long string of text that cannot fit entirely on the page. A common use for this control is displaying the license agreement.

Note that the string of text used with this control cannot contain an embedded property. To display text with embedded properties use instead the [Text Control](#).

## Control Attributes

You can use the following attributes with this control. To change the value of an attribute using an event, subscribe the control to a `ControlEvent` in the [EventMapping table](#) and list the attribute's identifier in the Attribute column. Enter the identifier of the `ControlEvent` in the Event column.

Attribute identifier	Hexadecimal bit	Description
<a href="#">Position</a>		Position of control in the dialog box. Enter the control's width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the <a href="#">Control table</a> or <a href="#">BBControl table</a> . Use <a href="#">installer units</a> for length and distance.
<a href="#">Text</a>		Text displayed by the control. Enter the RTF text string into the Text column of the <a href="#">Control table</a> .
<a href="#">Visible</a>	0x00000000 0x00000001	Hidden control. Visible control.  To make the control visible or hidden on its creation, include this bit in the bit word of the Attributes column in the <a href="#">Control table</a> or <a href="#">BBControl table</a> .  You can also hide or show a control by using the <a href="#">ControlCondition table</a> .

<a href="#">Enabled</a>	0x00000000 0x00000002	Control in a disabled state. Control in an enabled state.  Include this bit in the Attributes column of the <a href="#">Control</a> or <a href="#">BBControl</a> tables to enable the control on creation.  You can also enable or disable a control by using the <a href="#">ControlCondition</a> table.
<a href="#">Sunken</a>	0x00000000 0x00000004	Display the default visual style. Display the control with a sunken, 3D, look.  Include these bits in the bit word in the Attributes column of the <a href="#">Control</a> table.
<a href="#">RTLRO</a>	0x00000000 0x00000020	Text in the control is displayed in a left-to-right reading order. Text in the control is displayed in a right-to-left reading order.
<a href="#">RightAligned</a>	0x00000000 0x00000040	Text in the control is aligned on the left. Text in the control is aligned on the right.
<a href="#">LeftScroll</a>	0x00000000 0x00000080	The scroll bar is located on the right side of the control. The scroll bar is located on the left side of the control.
<a href="#">BiDi</a>	0x000000E0	Set this value for a combination of the <a href="#">RTLRO</a> , <a href="#">RightAligned</a> , and <a href="#">LeftScroll</a> attributes.

## Remarks

This control can be created from the RICHEDIT class by using the **CreateWindowEx** function. It has the ES\_MULTILINE, WS\_VSCROLL, ES\_READONLY, WS\_TABSTOP, ES\_AUTOVSCROLL, WS\_CHILD, WS\_GROUP, and ES\_NOOLEDRAGDROP styles.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SelectionTree Control

This control enables a user to change the selection state of features listed in the [Feature table](#). The control is associated with a string valued property that the user can set by a [Browse dialog](#). You can associate the control with a property by entering the property's name in the Property column of the [Control table](#).

The SelectionTree control automatically publishes the following [Control Events](#) on Windows XP or earlier operating systems. The SelectionTree control publishes these events when the selected item is changed from one node to another. If the selection tree has no nodes, the control publishes these events and erases the contents of controls that subscribe to the event. These ControlEvents are not required to be listed in the [ControlEvent table](#).

Control event	Description
<a href="#">SelectionAction</a>	Publishes a string from the <a href="#">UIText table</a> describing the highlighted item.
<a href="#">SelectionBrowse</a>	Generates a Browse dialog box used to modify the path of the highlighted item.
<a href="#">SelectionDescription</a>	Publishes a string from the <a href="#">Feature table</a> describing the highlighted item.
<a href="#">SelectionNoItems</a>	Deletes the descriptive text or disables the buttons of an obsolete item.
<a href="#">SelectionPath</a>	Publishes the path for the highlighted item.
<a href="#">SelectionPathOn</a>	Publishes whether or not there is a selection path associated with the currently selected feature.
<a href="#">SelectionSize</a>	Publishes the size of the highlighted item.

Beginning with the Windows Server 2003 systems, SelectionTree controls publish all of the events in the above table, and in addition, publish a [DoAction ControlEvent](#) or a [SetProperty ControlEvent](#). Records must be added to the ControlEvent table to publish DoAction or

## SetProperty ControlEvents.

Control event	Description
<a href="#">DoAction</a>	Notifies the installer to execute a custom action.
<a href="#">SetProperty</a>	Sets a property to a new value.

Beginning with Windows Installer version 3.0, SelectionTree controls publish an event that runs [custom actions](#) listed in the ControlEvent table. The SelectionTree control publishes this event whenever the feature selection changes in the control or whenever a different selection state is chosen for the current feature. The custom actions run each time the event is published. The SelectionTree control sends information to the custom action by setting the values of the following properties. All these properties are all cleared when the SelectionTree control is closed.

**Windows Installer 2.0:** Not supported. The SelectionTree control does not publish the event and does not set the following properties.

Property	Description
MsiSelectionTreeSelectedFeature	The selected feature's name in the Feature field of the Feature table.
MsiSelectionTreeSelectedAction	The selected feature's installation action state. The value may be INSTALLSTATE_ABSENT, INSTALLSTATE_LOCAL, INSTALLSTATE_SOURCE, or INSTALLSTATE_ADVERTISED.
MsiSelectonTreeChildrenCount	Number of direct child nodes.
MsiSelectionTreeInstallingChildrenCount	Number of direct child nodes that are INSTALLSTATE_LOCAL, INSTALLSTATE_SOURCE, or INSTALLSTATE_ADVERTISED.
MsiSelectionTreeSelectedCost	Cost of installing the selected feature in units of 512 bytes.

MsiSelectionTreeChildrenCost	Cost of installing all the children features in units of 512 bytes.
MsiSelectionTreeSelectedPath	Path where the selected feature is being installed. Defined only if the feature is being installed as INSTALLSTATE_LOCAL.

## Note

The contents of the Text field of the [Control table](#) is never displayed by the SelectionTree control. Instead this field specifies the style of text to be displayed by the control and contains a description of the control used by screen review utilities. To set the font and font style of a text string, prefix the string of displayed characters with {style} or {&style}. Where style is an identifier listed in the TextStyle column of the [TextStyle table](#). If neither of these are present, but the [DefaultUIFont](#) property is defined as a valid text style, that font is used. The information following this is read by screen review utilities as the description of the control. See [Accessibility](#).

## Control Attributes

You can use the following attributes with this control. To change the value of an attribute using an event, subscribe the control to a ControlEvent in the [EventMapping table](#) and list the attribute's identifier in the Attribute column. Enter the identifier of the ControlEvent in the Event column.

<b>Attribute identifier</b>	<b>Hexadecimal bit</b>	<b>Description</b>
<a href="#">IndirectPropertyName</a>		Name of an indirect property associated with the control. If the Indirect attribute bit is set, the control displays or changes the value of the property having this name. If the Indirect attribute bit is set, this name is also the value of the property listed in the Property column of the <a href="#">Control table</a> .

<a href="#">Position</a>		<p>Position of the control in the dialog box.</p> <p>Enter the control's width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the <a href="#">Control table</a>. Use <a href="#">installer units</a> for length and distance.</p>
<a href="#">PropertyName</a>		<p>Name of the property associated with this control. If the Indirect attribute bit is not set, the control displays or changes the value of the property having this name. This attribute is specified in the Property column of the <a href="#">Control table</a>.</p>
<a href="#">PropertyValue</a>		<p>Current value of the property displayed or changed by this control. If the Indirect attribute bit is not set, this is the value of PropertyName. If the Indirect attribute bit is set, this is the value of IndirectPropertyName. If the attribute changes, the control reflects the new value.</p>
<a href="#">Text</a>		<p>Displays text in screenreaders according to text entered into the Text column of the <a href="#">Control table</a>. See <a href="#">Accessibility</a>.</p>
<a href="#">Visible</a>	<p>0x00000000 0x00000001</p>	<p>Hidden control. Visible control.</p> <p>Include this bit in the bit word of the Attributes column in the <a href="#">Control table</a> to make the control visible or hidden upon its creation.</p> <p>You can also hide or show a control by using the <a href="#">ControlCondition table</a>.</p>
<a href="#">Enabled</a>	0x00000000	Control in a disabled state.

	0x00000002	<p>Control in an enabled state.</p> <p>Include this bit in the bit word in the Attributes column of the <a href="#">Control</a> to enable the control on creation.</p> <p>You can also enable or disable a control by using the <a href="#">ControlCondition table</a>.</p>
<a href="#">Sunken</a>	0x00000000 0x00000004	<p>Displays the default visual style.</p> <p>Displays the control with a sunken, 3D, look.</p> <p>Include these bits in the bit word in the Attributes column of the <a href="#">Control table</a>.</p>
<a href="#">Indirect</a>	0x00000000 0x00000008	<p>The control displays or changes the value of the property in the Property column of the <a href="#">Control table</a>.</p> <p>The control displays or changes the value of the property that has the Identifier listed in the Property column of the Control table.</p> <p>Determines if the property associated with this control is referenced indirectly.</p>
<a href="#">RTLRO</a>	0x00000000 0x00000020	<p>Text in the control is displayed in left-to-right reading order.</p> <p>Text in the control is displayed in right-to-left reading order.</p>
<a href="#">RightAligned</a>	0x00000000 0x00000040	<p>Text in the control is aligned to the left.</p> <p>Text in the control is aligned to the right.</p>
<a href="#">LeftScroll</a>	0x00000000	<p>The scroll bar is located on the right</p>

	0x00000080	side of the control. The scroll bar is located on the left side of the control.
<a href="#">BiDi</a>	0x000000E0	Set this value for a combination of the <a href="#">RTLRO</a> , <a href="#">RightAligned</a> , and <a href="#">LeftScroll</a> attributes.

## Remarks

This control can be created from the `WC_TREEVIEW` class by using the **CreateWindowEx** function. It has the `WS_BORDER`, `TVS_HASLINES`, `TVS_HASBUTTONS`, `TVS_LINESATROOT`, `TVS_DISABLEDRAHDROP`, `TVS_SHOWSELALWAYS`, `WS_CHILD`, `WS_TABSTOP`, and `WS_GROUP` styles.

The selection tree is only populated if the [CostInitialize action](#) and [CostFinalize action](#) have been called.

The following string in the [UIText table](#) is related to this control.

Term	Description
AbsentPath	The path displayed for an item in the absent state.

The following six strings are used to display the number of children selected and the size associated with the highlighted item:

- SelChildCostPos
- SelChildCostNeg
- SelParentCostPosPos
- SelParentCostPosNeg
- SelParentCostNegPos

- SelParentCostNegNeg

The following strings are used to display the available selection options for an item in a popup menu:

- MenuAbsent
- MenuLocal
- MenuCD
- MenuNetwork
- MenuAllLocal
- MenuAllCD
- MenuAllNetwork

The following strings are used to explain the present selection in the [SelectionDescription](#) ControlEvent.

- SelAbsentAbsent
- SelAbsentLocal
- SelAbsentCD
- SelAbsentNetwork
- SelLocalAbsent
- SelLocalLocal
- SelLocalCD
- SelLocalNetwork
- SelCDAbsent
- SelNetworkAbsent
- SelCDLocal
- SelNetworkLocal
- SelCDCD
- SelNetworkNetwork

The following four localized strings are used in formatting the size of a file:

- Bytes
- KB
- MB
- GB

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Text Control

The Text control displays static text, which may use a predefined style.

The recommended method for displaying text with specified line breaks is to use multiple one-line text controls located below each other. The character sequences `\n`, `\r\n`, or `\n\r` in the text field for the control are not displayed as a line break. These character sequences are literally displayed by the control.

## Control Attributes

You can use the following attributes with the Text control. To change the value of an attribute using an event, subscribe the control to a `ControlEvent` in the [EventMapping table](#) and list the attribute's identifier in the Attribute column. Enter the identifier of the `ControlEvent` in the Event column.

Attribute identifier	Hexadecimal bit	Description
<a href="#">Position</a>		Position of control in the dialog box. Enter the control's width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the <a href="#">Control table</a> or <a href="#">BBControl table</a> . Use <a href="#">installer units</a> for length and distance.
<a href="#">Text</a>		Text displayed by the control. To set the font and font style of a text string, prefix the string of displayed characters with <code>{style}</code> or <code>&amp;style</code> . Where style is an identifier listed in the TextStyle column of the <a href="#">TextStyle table</a> . If neither of these are present, but the <a href="#">DefaultUIFont</a> property is defined as a valid text style, that font will be used.
<a href="#">TimeRemaining</a>		This attribute enables a Text control to

		<p>display the approximate number of minutes and seconds remaining for an installation. Subscribe the Text control to the <a href="#">TimeRemaining ControlEvent</a> in the <a href="#">Eventmapping table</a> and enter TimeRemaining into the Attribute column.</p> <p>The installer publishes a record containing one integer representing the number of seconds remaining in the installation. Include a row in the <a href="#">UIText table</a> with TimeRemaining in the Key column. Enter a formatted text string into the Text column authored to display minutes and seconds. Format this string as described for <a href="#">MsiFormatRecord</a>.</p>
<a href="#">Visible</a>	0x00000000 0x00000001	<p>Hidden control. Visible control.</p> <p>Include this bit in the bit word of the Attributes column in the <a href="#">Control table</a> or <a href="#">BBControl table</a> to make the control visible or hidden upon its creation.</p> <p>You can also hide or show a control by using the <a href="#">ControlCondition table</a>.</p>
<a href="#">Enabled</a>	0x00000000 0x00000002	<p>Control in a disabled state. Control in an enabled state.</p> <p>Include this bit in the bit word in the Attributes column of the <a href="#">Control</a> or <a href="#">BBControl tables</a> to enable the control on creation.</p> <p>You can also enable or disable a control by using the <a href="#">ControlCondition table</a>.</p>
<a href="#">Sunken</a>	0x00000000 0x00000004	<p>Displays the default visual style. Displays the control with a sunken, 3-D, look.</p>

		Include these bits in the bit word in the Attributes column of the <a href="#">Control table</a> .
<a href="#">RTLRO</a>	0x00000000 0x00000020	Text in the control is displayed in left-to-right reading order. Text in the control is displayed in right-to-left reading order.
<a href="#">RightAligned</a>	0x00000000 0x00000040	Text in the control is aligned to the left. Text in the control is aligned to the right.
<a href="#">Transparent</a>	0x00000000 0x00010000	Opaque control. Background shows through control. The control has the WS_EX_TRANSPARENT style.  Include this bit in the Attributes column of the <a href="#">Control</a> or <a href="#">BBControl tables</a> .
<a href="#">NoPrefix</a>	0x00000000 0x00020000	Use & in a text string to display the next character as underscored. The character & in a string is displayed as itself.  Include this bit in the bit word in the Attributes column of the <a href="#">Control</a> or <a href="#">BBControl tables</a> .
<a href="#">NoWrap</a>	0x00000000 0x00040000	Text wraps. Text is displayed on a single line. If the text extends beyond the control's margins, it is clipped and an ellipsis ("...") is inserted.  Include this bit in the bit word in the Attributes column of the <a href="#">Control</a> or <a href="#">BBControl tables</a> .
<a href="#">UsersLanguage</a>	0x00000000 0x00100000	Fonts created in the database code page. Fonts created in the user's default UI code page.

<p><a href="#">FormatSize</a> <a href="#">Control</a> <a href="#">Attribute</a></p>	<p>0x00000000 0x00080000</p>	<p>Formatted as text. If this bit is set the control attempts to format the displayed text as a number representing a count of bytes. For proper formatting, the control's text must be set to a string representing a number expressed in units of 512 bytes. The displayed value will then be formatted in terms of kilobytes (KB), megabytes (MB), or gigabytes (GB), and displayed with the appropriate string representing the units.</p>
---	----------------------------------	--

## Remarks

This control can be created from the `STATIC` class by using the **CreateWindowEx** function. It has the `SS_LEFT`, `WS_CHILD`, and `WS_GROUP` styles.

Do not place transparent Text controls on top of colored bitmaps. The text may not be visible if the user changes the display color scheme. For example, text may become invisible if the user sets the high contrast parameter for accessibility reasons.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# VolumeCostList Control

The VolumeCostList control presents information about the cost associated with the selection on the different volumes. The control shows all the volumes involved in the current installation plus all volumes that are of the types specified in the attribute bits. If the required disk space exceeds the amount available on some drive, the volume is highlighted in the table. Clicking a column header sorts the volumes according to the chosen column.

The following strings are used for the column headings in the control and must be specified in the [UIText table](#):

- **VolumeCostAvailable**
- **VolumeCostDifference**
- **VolumeCostRequired**
- **VolumeCostSize**
- **VolumeCostVolume**

The following four localized strings are used in formatting the size of a file:

- **Bytes**
- **KB**
- **MB**
- **GB**

Authors can set the column widths in the VolumeCostList control by appending column widths after any font specifications. Column widths are entered as a series of positive integers enclosed in curly braces. Empty curly braces or {} hide the column. A negative number or a string that cannot be converted into a positive integer is an invalid column width. When an invalid column width is encountered in the series, the remaining columns are hidden. A maximum of five column widths can be specified.

The contents of the Text field of the [Control table](#) are never displayed by the VolumeCostList control. Instead this field specifies the style of text to

be displayed by the control and contains a description of the control used by screen review utilities. To set the font and font style of a text string, prefix the string of displayed characters with {\style} or {&style}. Where style is an identifier listed in the TextStyle column of the [TextStyle table](#). If neither of these are present, but the [DefaultUIFont](#) property is defined as a valid text style, that font will be used. The information following this is read by screen review utilities as the description of the control. See [Accessibility](#).

Note that the installer does not update the content of the VolumeCostControl when a user enters a different path into a [PathEdit control](#), a [Browse dialog box](#), [DirectoryList control](#), or [DirectoryCombo control](#) because there is no property associated with the VolumeCostList control.

## Control Attributes

You can use the following attributes with this control. To change the value of an attribute using an event, subscribe the control to a ControlEvent in the [EventMapping table](#) and list the attribute's identifier in the Attribute column. Enter the identifier of the ControlEvent in the Event column.

Attribute identifier	Hexadecimal bit	Description
<a href="#">Position</a>		Position of control in the dialog box. Enter the control's width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the <a href="#">Control table</a> . Use <a href="#">installer units</a> for length and distance.
<a href="#">Text</a>		To display text in screen readers, enter the text into the Text column of the <a href="#">Control table</a> . See <a href="#">Accessibility</a> . To set the column widths in the VolumeCostList control, append

		<p>the column widths to any font specifications. Column widths are entered as a series of positive integers enclosed in curly braces. Empty curly braces or {0} hide the column. A negative number or a string that cannot be converted into a positive integer is an invalid column width. When an invalid column width is encountered in the series, the remaining columns are hidden. A maximum of five column widths can be specified.</p>
Visible	<p>0x00000000 0x00000001</p>	<p>Hidden control. Visible control.</p> <p>Include this bit in the bit word of the Attributes column in the <a href="#">Control table</a> to make the control visible or hidden upon its creation.</p> <p>You can also hide or show a control by using the <a href="#">ControlCondition table</a>.</p>
Enabled	<p>0x00000000 0x00000002</p>	<p>Control in a disabled state. Control in an enabled state.</p> <p>Include this bit in the bit word in the Attributes column of the <a href="#">Control</a> to enable the control on creation.</p> <p>You can also enable or disable a control by using the <a href="#">ControlCondition table</a>.</p>
Sunken	<p>0x00000000</p>	<p>Displays the default visual style.</p>

	0x00000004	Displays the control with a sunken, 3-D, look.  Include these bits in the bit word in the Attributes column of the <a href="#">Control table</a> .
<a href="#">RTLRO</a>	0x00000000 0x00000020	Text in the control is displayed a left-to-right reading order. Text in the control is displayed a right-to-left reading order.
<a href="#">RightAligned</a>	0x00000000 0x00000040	Text in the control is aligned to the left. Text in the control is aligned to the right.
<a href="#">LeftScroll</a>	0x00000000 0x00000080	The scroll bar is located on the right side of the control. The scroll bar is located on the left side of the control.
<a href="#">BiDi</a>	0x000000E0	Set this value for a combination of the <a href="#">RTLRO</a> , <a href="#">RightAligned</a> , and <a href="#">LeftScroll</a> attributes.
<a href="#">RemovableVolume</a>	0x00010000	Control lists removable drives. Include in the bit word in the Attributes column of the <a href="#">Control table</a> .
<a href="#">FixedVolume</a>	0x00020000	Control lists fixed internal hard drives. Include in the bit word in the Attributes column of the <a href="#">Control table</a> .
<a href="#">RemoteVolume</a>	0x00040000	Control lists remote volumes. Include in the bit word in the Attributes column of the <a href="#">Control</a>

		table.
CDROMVolume	0x00080000	Control lists CD-ROM volumes. Include in the bit word in the Attributes column of the <a href="#">Control table</a> .
RAMDiskVolume	0x00100000	Control lists RAM disks. Include in the bit word in the Attributes column of the <a href="#">Control table</a> .
FloppyVolume	0x00200000	Control lists floppy drives. Include in the bit word in the Attributes column of the <a href="#">Control table</a> .
ControlShowRollbackCost	0x00000000 0x00400000	<p>If <b>PROMPTROLLBACKCOST</b> = P, and this attribute is not set, the rollback, backup files are not included in the cost displayed by the VolumeCostList Control.</p> <p>If <b>PROMPTROLLBACKCOST</b> = P, and this attribute is set, the rollback, back-up files are included in the cost displayed by the VolumeCostList control.</p> <p>This control attribute is ignored if <b>PROMPTROLLBACKCOST</b> = D or F. If <b>PROMPTROLLBACKCOST</b> = F, the cost of the rollback, backup files is included. If <b>PROMPTROLLBACKCOST</b> = D, or <b>DISABLEROLLBACK</b></p>

	= 1, the cost of the rollback, back-up files is not included.
--	--

## Remarks

This control can be created from the `WC_LISTVIEW` class by using the **CreateWindowEx** function. It has the `LVS_REPORT`, `WS_VSCROLL`, `WS_HSCROLL`, `LVS_SHAREIMAGELISTS`, `LVS_AUTOARRANGE`, `LVS_SINGLESEL`, `WS_BORDER`, `WS_CHILD`, `WS_TABSTOP`, and `WS_GROUP` styles.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# VolumeSelectCombo Control

The VolumeSelectCombo control enables the user to select a volume from an alphabetical list of volumes. The types of volumes displayed in the list are specified using bits associated with [RemovableVolume](#), [FixedVolume](#), [RemoteVolume](#), [CDROMVolume](#), [RAMDiskVolume](#), and [FloppyVolume](#) control attributes.

You can associate this control with a property by entering the property's name in the Property column of the [Control table](#).

## Control Attributes

You can use the following attributes with this control. To change the value of an attribute using an event, subscribe the control to a ControlEvent in the [EventMapping table](#) and list the attribute's identifier in the Attribute column. Enter the identifier of the ControlEvent in the Event column.

<b>Attribute identifier</b>	<b>Hexadecimal bit</b>	<b>Description</b>
<a href="#">IndirectPropertyName</a>		This is the name of an indirect property associated with the control. If the Indirect attribute bit is set, the control displays or changes the value of the property having this name. If the Indirect attribute bit is set, this name is also the value of the property listed in the Property column of the <a href="#">Control table</a> .
<a href="#">Position</a>		Position of control in the dialog box. Enter the control's width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the <a href="#">Control table</a> . Use <a href="#">installer units</a> for length and distance.
<a href="#">PropertyName</a>		This is the name of the property associated with this control. If the

		Indirect attribute bit is not set, the control displays or changes the value of the property having this name. This attribute is specified in the Property column of the <a href="#">Control table</a> .
<a href="#">PropertyValue</a>		Current value of the property displayed or changed by this control. If the Indirect attribute bit is not set, this is the value of <code>PropertyName</code> . If the Indirect attribute bit is set, this is the value of <code>IndirectPropertyName</code> . If the attribute changes, the control reflects the new value.
<a href="#">Text</a>		To set the font and font style of a text string, prefix the string of displayed characters with <code>{\style}</code> or <code>{&amp;style}</code> . Where style is an identifier listed in the <code>TextStyle</code> column of the <a href="#">TextStyle table</a> . If neither of these are present, but the <b>DefaultUIFont</b> property is defined as a valid text style, that font will be used.
<a href="#">Visible</a>	0x00000000 0x00000001	Hidden control. Visible control.  Include this bit in the bit word of the <code>Attributes</code> column in the <a href="#">Control table</a> to make the control visible or hidden upon its creation.  You can also hide or show a control by using the <a href="#">ControlCondition table</a> .
<a href="#">Enabled</a>	0x00000000 0x00000002	Control in a disabled state. Control in an enabled state.  Include this bit in the bit word in the <code>Attributes</code> column of the <a href="#">Control</a> to enable the control on creation.

		You can also enable or disable a control by using the <a href="#">ControlCondition table</a> .
<a href="#">Sunken</a>	0x00000000 0x00000004	Displays the default visual style. Displays the control with a sunken, 3-D, look.  Include these bits in the bit word in the Attributes column of the <a href="#">Control table</a> .
<a href="#">Indirect</a>	0x00000000 0x00000008	The control displays or changes the value of the property in the Property column of the <a href="#">Control table</a> . The control displays or changes the value of the property that has the Identifier listed in the Property column of the Control table.  Determines if the property associated with this control is referenced indirectly.
<a href="#">RTLRO</a>	0x00000000 0x00000020	Text in the control is displayed in left-to-right reading order. Text in the control is displayed in right-to-left reading order.
<a href="#">RightAligned</a>	0x00000000 0x00000040	Text in the control is aligned to the left. Text in the control is aligned to the right.
<a href="#">LeftScroll</a>	0x00000000 0x00000080	The scroll bar is located on the right side of the control. The scroll bar is located on the left side of the control.

BiDi	0x000000E0	Set this value for a combination of the <a href="#">RTLRO</a> , <a href="#">RightAligned</a> , and <a href="#">LeftScroll</a> attributes.
RemovableVolume	0x00010000	Control lists removable drives. Include in the bit word in the Attributes column of the <a href="#">Control table</a> .
FixedVolume	0x00020000	Control lists fixed internal hard drives. Include in the bit word in the Attributes column of the <a href="#">Control table</a> .
RemoteVolume	0x00040000	Control lists remote volumes. Include in the bit word in the Attributes column of the <a href="#">Control table</a> .
CDROMVolume	0x00080000	Control lists CD-ROM volumes. Include in the bit word in the Attributes column of the <a href="#">Control table</a> .
RAMDiskVolume	0x00100000	Control lists RAM disks. Include in the bit word in the Attributes column of the <a href="#">Control table</a> ..
FloppyVolume	0x00200000	Control lists floppy drives. Include in the bit word in the Attributes column of the <a href="#">Control table</a> .

## Remarks

This control can be created from the COMBOBOX class by using the

**CreateWindowEx** function. It has the CBS\_DROPDOWNLIST, CBS\_OWNERDRAWFIXED, CBS\_HASSTRINGS, WS\_VSCROLL, WS\_CHILD, WS\_GROUP, WS\_TABSTOP, and CBS\_SORT styles. For information about developing a user interface with Windows, see [User Interface Design and Development](#).

For compatibility with screen readers, when authoring a dialog box with a VolumeSelectCombo control as the first active control, you must make the text field belonging to the edit field the first active control in the [Dialog table](#). Since the static text cannot take focus, when the dialog box is created, the edit field will have the focus initially as intended. This ensures that screen readers show the correct information.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Control Attributes

For information on control attributes, see the link to the particular control that you need to create in [Controls](#) as well as the links to particular control attributes in the following lists.

The following methods are used for specifying the attributes of a control:

- Use the [ControlCondition table](#) to disable, enable, hide, or show a control according to the value of a property or conditional statement. You can also use this table to override the default control specified in the [Dialog table](#).
- Subscribe the control to a ControlEvent in the [EventMapping table](#). Enter the attribute's identifier in the Attribute column and the ControlEvent's identifier in the Event column of this table.
- Set the control attribute bit flags for the control in the Attribute column of the [Control table](#). This sets the attributes upon the creation of the control.

Some attributes cannot be set for every control or be specified by all of the above methods. See the particular control and attribute topics for details.

The initial values of some control attributes can be set with bits in the [Control table](#).

Attribute	Decimal	Hexadecimal	Constant
<a href="#">BiDi</a>	224	0x000000E0	msidbControlAttributesBiDi
<a href="#">Enabled</a>	2	0x00000002	msidbControlAttributesEnabled
<a href="#">Indirect</a>	8	0x00000008	msidbControlAttributesIndirect
<a href="#">Integer Control</a>	16	0x00000010	msidbControlAttributesInteger
<a href="#">LeftScroll</a>	128	0x00000080	msidbControlAttributesLeftScroll
<a href="#">RightAligned</a>	64	0x00000040	msidbControlAttributesRightAligned

RTLRO	32	0x00000020	msidbControlAttributesRTLRO
Sunken	4	0x00000004	msidbControlAttributesSunken
Visible	1	0x00000001	msidbControlAttributesVisible

These attributes of Text controls are set with bits.

Attribute	Decimal	Hexadecimal	Constant
FormatSize	524288	0x00080000	msidbControlAttributesFormatSize
NoPrefix	131072	0x00020000	msidbControlAttributesNoPrefix
NoWrap	262144	0x00040000	msidbControlAttributesNoWrap
Password	2097152	0x00200000	msidbControlAttributesPasswordInput
Transparent	65536	0x00010000	msidbControlAttributesTransparent
UsersLanguage	1048576	0x00100000	msidbControlAttributesUsersLanguage

This attribute of the ProgressBar control is set with a bit.

Attribute	Decimal	Hexadecimal	Constant
Progress95	65536	0x00010000	msidbControlAttributesProgress95

These attributes of Volume and Directory SelectCombo controls are set with bits.

Attribute	Decimal	Hexadecimal	Constant
CDROMVolume	524288	0x00080000	msidbControlAttributesCDROMVolume
FixedVolume	131072	0x00020000	msidbControlAttributesFixedVolume
FloppyVolume	2097152	0x00200000	msidbControlAttributesFloppyVolume
RAMDiskVolume	1048576	0x00100000	msidbControlAttributesRAMDiskVolume

RemoteVolume	262144	0x00040000	msidbControlAttributesRemoteVo
RemovableVolume	65536	0x00010000	msidbControlAttributesRemovabl

These attributes of ListBox and ComboBox controls are set with bits.

Attribute	Decimal	Hexadecimal	Constant
ComboList Control	131072	0x00020000	msidbControlAttributesComboList
Sorted Control	65536	0x00010000	msidbControlAttributesSorted

This attribute of the Edit control is set with a bit.

Attribute	Decimal	Hexadecimal	Constant
MultiLine	65536	0x00010000	msidbControlAttributesMultiline

These attributes of PictureBox controls are set with bits.

Attribute	Decimal	Hexadecimal	Constant
Bitmap	262144	0x00040000	msidbControlAttributesBitmap
FixedSize	1048576	0x00100000	msidbControlAttributesFixedSize
Icon	524288	0x00080000	msidbControlAttributesIcon
IconSize16	2097152	0x00200000	msidbControlAttributesIconSize16
IconSize32	4194304	0x00400000	msidbControlAttributesIconSize32
IconSize48	6291456	0x00600000	msidbControlAttributesIconSize48
PushLike Control	131072	0x00020000	msidbControlAttributesPushLike

This attribute of RadioButton control is set with a bit.

<b>Attribute</b>	<b>Decimal</b>	<b>Hexadecimal</b>	<b>Constant</b>
<a href="#">HasBorder</a>	16777216	0x01000000	msidbControlAttributesHasBorder

This attribute of PushButton control is set with a bit.

<b>Attribute</b>	<b>Decimal</b>	<b>Hexadecimal</b>	<b>Constant</b>
<a href="#">ElevationShield</a>	8388608	0x00800000	msidbControlAttributesElevationShield

This attribute of VolumeCostList control is set with a bit.

<b>Attribute</b>	<b>Decimal</b>	<b>Hexadecimal</b>	<b>Constant</b>
<a href="#">ControlShowRollbackCost</a>	4194304	0x00400000	msidbControlShowRollbackCost

The following control attributes are not set with bits. These attributes are authored into the user interface tables or are set using [Control Events](#).

[BillboardName](#)

[IndirectPropertyName](#)

[Position](#)

[Progress Control](#)

[PropertyName](#)

[PropertyValue](#)

[Text Control](#)

[TimeRemaining](#)

See [Adding Controls and Text](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# BiDi Control Attribute

This is a combination of the right-to-left reading order [RTLRO](#), the [RightAligned](#), and [LeftScroll](#) attributes.

## Valid Controls

[ScrollableTextVolumeCostList](#)  
[ComboBox](#)  
[DirectoryList](#)  
[DirectoryCombo](#)  
[Edit](#)  
[ListBox](#)  
[ListView](#)  
[SelectionTree](#)  
[VolumeSelectCombo](#)

## Value

Decimal	Hexadecimal	Constant
224	0x000000E0	msidbControlAttributesBiDi

## Remarks

To set this attribute on a control, include the BiDi bit in the Attributes column of the control's record in the [Control table](#).

See [Control Attributes](#) and the control you need to create under [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# BillboardName Control Attribute

This attribute returns the name of the currently running billboard, or sets and displays a billboard by name. This attribute is used in preview mode.

## Valid Controls

[Billboard](#)

## Associated Bit Flags

This attribute does not use bit flags.

## Remarks

See [Control Attributes](#) and the control you need to create under [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Bitmap Control Attribute

If the Bitmap Control bit is set, the text in the control is replaced by a bitmap image. The Text column in the [Control table](#) is a foreign key into the [Binary table](#).

If this bit is not set, the text in the control is specified in the Text column of the [Control table](#).

## Valid Controls

[CheckBoxPushButton](#)  
[RadioButtonGroup](#)

## Value

Decimal	Hexadecimal	Constant
262144	0x00040000	msidbControlAttributesBitmap

## Remarks

To set this attribute on a control, include the Bitmap bit in the Attributes column of the control's record in the [Control table](#).

The Text column in the Control table is used as a foreign key to the [Binary table](#), therefore the control cannot contain both an icon image and text.

Do not set both the [Icon](#) and Bitmap bits.

See [Control Attributes](#) and the control you need to create under [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# CDROMVolume Control Attribute

If the CDROMVolume Control bit is set, the control shows all the volumes in the current installation plus all the CD-ROM volumes.

If this bit is not set, the control shows all the volumes in the current installation.

## Valid Controls

[DirectoryComboVolumeCostList](#)  
[VolumeSelectCombo](#)

## Value

Decimal	Hexadecimal	Constant
524288	0x00080000	msidbControlAttributesCDROMVolume

## Remarks

To set this attribute on a control, include the CDROMVolume bit in the Attributes column of the control's record in the [Control table](#).

See [Control Attributes](#) and the control you need to create under [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ComboBox Control Attribute

If the ComboBox Control bit is set on a combo box, the edit field is replaced by a static text field. This prevents a user from entering a new value and requires the user to choose only one of the predefined values.

If this bit is not set, the combo box has an edit field.

## Valid Controls

[ComboBox](#)

## Value

Decimal	Hexadecimal	Description
131072	0x00020000	msidbControlAttributesComboBox

## Remarks

To set this attribute on a control, include the ComboBox bit in the Attributes column of the control's record in the [Control table](#).

See [Control Attributes](#) and the control you need to create under [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ControlShowRollbackCost Control Attribute

This attribute specifies whether or not the rollback backup files are included in the costs displayed by the [VolumeCostList control](#).

If this bit is set and **PROMPTROLLBACKCOST** property = P, the rollback backup files are included in the cost displayed by the VolumeCostList Control.

If this bit is not set and **PROMPTROLLBACKCOST** property = P, the rollback backup files are not included in the cost displayed by the VolumeCostList Control.

## Valid Controls

[VolumeCostList](#)

## Value

Decimal	Hexadecimal	Constant
4194304	0x00400000	msidbControlShowRollbackCost

## Remarks

This control attribute is ignored if PROMPTROLLBACKCOST = D or F.

If PROMPTROLLBACKCOST = F, the cost of the rollback, backup files is included.

If PROMPTROLLBACKCOST = D, or **DISABLEROLLBACK** property = 1, the cost of the rollback, backup files is not included.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Enabled Control Attribute

This attribute specifies if the given control is enabled or disabled. Most controls appear gray when disabled.

If this bit is set, the control is created as enabled.

If this bit is not set, the control is created as disabled.

## Valid Controls

All controls. The appearance of some controls that are not associated with a property, such as Bitmaps and Icons, are unaffected by setting this attribute.

## Value

Decimal	Hexadecimal	Constant
2	0x00000002	msidbControlAttributesEnabled

## Remarks

You can also use the [ControlCondition table](#) to disable or enable a control according to the value of a property or conditional statement.

You can also enable or disable a control by subscribing the control to a [ControlEvent](#). Enter the attribute's identifier in the Attribute column and the event's identifier in the Event column of the [EventMapping table](#).

See [Control Attributes](#) and the control you need to create under [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ElevationShield Attribute

This attribute adds the *User Account Control* (UAC) elevation icon (shield icon) to the [PushButton control](#).

If this bit is set, and the installation is not yet running with *elevated* privileges, the pushbutton control is created using the User Account Control (UAC) elevation icon (shield icon).

If this bit is set, and the installation is already running with elevated privileges, the pushbutton control is created using the other icon attributes.

If this bit is not set, the pushbutton control is created using the other icon attributes.

## Valid Controls

[PushButton](#)

## Value

Decimal	Hexadecimal	Constant
8388608	0x00800000	msidbControlAttributesElevationShield

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# FixedSize Control Attribute

If the FixedSize Control bit is set, the picture is cropped or centered in the control without changing its shape or size.

If this bit is not set the picture is stretched to fit the control.

## Valid Controls

[BitmapCheckBox](#)

[Icon](#)

[PushButton](#)

[RadioButtonGroup](#)

## Value

Decimal	Hexadecimal	Constant
1048576	0x00100000	msidbControlAttributesFixedSize

## Remarks

To set this attribute on a control, include the FixedSize bit in the Attributes column of the control's record in the [Control table](#).

Setting the FixedSize bit has no effect on a [CheckBox](#), [PushButton](#), or [RadioButtonGroup](#) if neither the [Bitmap](#) or the [Icon](#) have been set.

Setting the FixedSize bit has no effect on an [Icon](#) control, or a [PushButton](#) associated with an icon, if the [IconSize](#) bits is not set.

See [Control Attributes](#) and the control you need to create under [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# FixedVolume Control Attribute

If the FixedVolume Control bit is set, the control shows all the volumes involved in the current installation plus all the fixed internal hard drives.

If this bit is not set, the control lists the volumes in the current installation.

## Valid Controls

[DirectoryComboVolumeCostList](#)  
[VolumeSelectCombo](#)

Decimal	Hexadecimal	Constant
131072	0x00020000	msidbControlAttributesFixedVolume

## Remarks

To set this attribute on a control, include the FixedVolume bit in the Attributes column of the control's record in the [Control table](#).

See [Control Attributes](#) and the control you need to create under [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# FloppyVolume Control Attribute

If the FloppyVolume Control bit is set, the control shows all the volumes involved in the current installation plus all the floppy volumes.

If this bit is not set, the control lists volumes in the current installation.

## Valid Controls

[DirectoryComboVolumeCostList](#)  
[VolumeSelectCombo](#)

## Value

Decimal	Hexadecimal	Constant
2097152	0x00200000	msidbControlAttributesFloppyVolume

## Remarks

To set this attribute on a control, include the FloppyVolume bit in the Attributes column of the control's record in the [Control table](#).

See [Control Attributes](#) and the control you need to create under [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# FormatSize Control Attribute

If this bit is set for a static text control, the control automatically attempts to format the displayed text as a number that represents a count of bytes. For proper formatting, the control's text must be set to a string that represents a number expressed in units of 512 bytes. The displayed value is then formatted in kilobytes (KB), megabytes (MB), or gigabytes (GB), and displayed with the appropriate string that represents the units. For more information, see [Text Control](#).

Numerical value of original text	Unit string used
Less than 20480	KB
Less than 20971520	MB
Less than 10737418240	GB

## Valid Controls

Decimal	Hexadecimal	Control
524288	0x00080000	msidbControlAttributesFormatSize

## Remarks

To set this attribute on a control, include the FormatSize bits in the Attributes column of the control's record in the [Control Table](#). The control's text must be set to a string representing a number expressed in units of 512 bytes. The text of the unit strings are defined in the [UIText Table](#). The positioning of the unit string is controlled by the [LeftUnit Property](#). If the [LeftUnit Property](#) is defined to be any value, the unit string appears before the numerical value. If anything other than numerical characters appears in the text associated with the control, the displayed value is undefined.

At run time, the installer resolves the [PrimaryVolumeSpaceRequired](#)

Property to the total number of bytes required for the installation in units of 512. A static text control with FormatSize bit can be used to automatically format and label the total number of bytes required for the installation in KB, MB, or GB as appropriate. For the purposes of this example, assume the total number of bytes is 18,336,768. The installer sets the value of the PrimaryVolumeSpaceRequired property to 18,336,768 divided by 512, or 35,814. The number displayed by the text control with FormatSize would be 17MB.

The numerical values of the original text are given in units of 512. In the table above, the string 20,480 corresponds to the KB string because 20,480 times 512 yields a result of 10,485,760 bytes or 10,240 KB.

The unit strings listed in the previous table refer to keys in the [UIText Table](#), where the text of the unit string is defined.

The positioning of the unit string is controlled by the **LeftUnit** Property. If the **LeftUnit** Property is defined to be any value, the unit string appears before the numerical value.

If anything other than numerical characters appears in the text associated with the control, the displayed value is undefined.

For more information, see [Control Attributes](#) and [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# HasBorder Control Attribute

If this bit is set, the [RadioButtonGroup](#) has text and a border displayed around it.

If the style bit is not set, the border is not displayed and no text is displayed on the group.

## Valid Controls

[RadioButtonGroup](#)

## Valid

Decimal	Hexadecimal	Constant
16777216	0x01000000	msidbControlAttributesHasBorder

## Remarks

To set this attribute on a control, include the HasBorder bits in the Attributes column of the control's record in the [Control table](#).

See [Control Attributes](#) and the control you need to create under [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Icon Control Attribute

If this bit is set, text is replaced by an icon image and the Text column in the [Control table](#) is a foreign key into the [Binary table](#).

If this bit is not set, text in the control is specified in the Text column of the [Control table](#).

## Valid Controls

[CheckBoxPushButton](#)

[RadioButtonGroup](#)

## Value

Decimal	Hexadecimal	Constant
5242288	0x00080000	msidbControlAttributesIcon

## Remarks

To set this attribute on a control, include the Icon bits in the Attributes column of the control's record in the [Control table](#).

The Text column in the Control table is used as a foreign key to the [Binary table](#), therefore the control cannot contain both an icon image and text.

Do not set both the Icon and [Bitmap](#) bits.

See [Control Attributes](#) and the control you need to create under [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IconSize Control Attribute

An icon file can hold several different sizes of the same icon image. These bits specify which size of the icon image to load. If none of the bits are set, the first image is loaded. If only **msidbControlAttributesIconSize16** is set, the first 16x16 image is loaded. If only the **msidbControlAttributesIconSize32** is set, the first 32x32 image is loaded. If **msidbControlAttributesIconSize48** is set, the first 48x48 image is loaded.

## Valid Controls

[CheckBoxIcon](#)

[PushButton](#)

[RadioButtonGroup](#)

## Value

Decimal	Hexadecimal	Description
2097152	0x00200000	<b>msidbControlAttributesIconSize16</b>
4194304	0x00400000	<b>msidbControlAttributesIconSize32</b>
6291456	0x00600000	<b>msidbControlAttributesIconSize48</b>

## Remarks

To set this attribute on a control, include the IconSize bits in the Attributes column of the control's record in the [Control table](#).

If the [FixedSize](#) bit is not set, the loaded image is shrunk or stretched to fit the icon control. If the [FixedSize](#) bit is set, and the loaded image is smaller than the icon control, the picture is displayed centered inside the control. If the [FixedSize](#) bit is set, and the loaded image is larger than the icon control, the picture is reduced to fit the control.

See [Control Attributes](#) and the control you need to create under [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Indirect Control Attribute

The Indirect control attribute specifies whether the value displayed or changed by this control is referenced indirectly. If this bit is set, the control displays or changes the value of the property that has the identifier listed in the Property column of the [Control table](#). If this bit is not set, the control displays or changes the value of the property in the Property column of the Control table.

## Valid Controls

All active controls.

## Value

Decimal	Hexadecimal	Constant
8	0x00000008	msidbControlAttributesIndirect

## Remarks

See [Control Attributes](#) and the control you need to create under [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IndirectPropertyName Control Attribute

This attribute is the name of an indirect property associated with the control. If the [Indirect](#) bit is set, the control displays or changes the value of the property having this name and this name is the value of the property listed in the Property column of the [Control table](#).

For inert controls this attribute returns Null.

## Valid Controls

All active controls.

## Associated Bit Flags

This attribute does not use bit flags.

## Remarks

See [Control Attributes](#) and the control you need to create under [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Integer Control Attribute

If this bit is set on a control, the associated property specified in the Property column of the [Control table](#) is an integer. If this bit is not set, the property is a string value.

## Valid Controls

[CheckBoxComboBox](#)

[Edit](#)

[ListBox](#)

[RadioButtonGroup](#)

## Value

Decimal	Hexadecimal	Constant
16	0x00000010	msidbControlAttributesInteger

## Remarks

To set this attribute on a control, include the Integer bit in the Attributes column of the control's record in the [Control table](#).

See [Control Attributes](#) and [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# LeftScroll Control Attribute

If this bit is set, the scroll bar is located on the left side of the control.

If this bit is not set, the scroll bar is on the right side of the control.

## Valid Controls

[ScrollableTextVolumeCostList](#)

[ComboBox](#)

[DirectoryList](#)

[DirectoryCombo](#)

[Edit](#)

[ListBox](#)

[ListView](#)

[SelectionTree](#)

[VolumeSelectCombo](#)

## Value

Decimal	Hexadecimal	Constant
128	0x00000080	msidbControlAttributesLeftScroll

## Remarks

To set this attribute on a control, include the LeftScroll bit in the Attributes column of the control's record in the [Control table](#).

See [Control Attributes](#) and [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MultiLine Control Attribute

If this bit is set on an [Edit control](#), the installer creates a multiple line edit control with a vertical scroll bar.

If this bit is not set, it specifies displaying a normal edit control.

## Valid Controls

[Edit control](#).

Decimal	Hexadecimal	Constant
65536	0x00010000	msidbControlAttributesMultiline

## Remarks

To set this attribute on a control, include the MultiLine bit in the Attributes column of the control's record in the [Control table](#).

See [Control Attributes](#) and [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# NoPrefix Control Attribute

If this bit is set on a text control, the occurrence of the character "&" in a text string is displayed as itself. If this bit is not set, then the character following "&" in the text string is displayed as an underscored character.

## Valid Controls

[Text](#)

## Value

Decimal	Hexadecimal	Constant
131072	0x20000	msidbControlAttributesNoPrefix

## Remarks

To set this attribute on a control, include the NoPrefix bit in the Attributes column of the control's record in the [Control table](#).

See [Control Attributes](#) and [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# NoWrap Control Attribute

If this bit is set the text in the control is displayed on a single line. If the text extends beyond the control's margins it is truncated and an ellipsis ("...") is inserted at the end to indicate the truncation. If this bit is not set, text wraps.

## Valid Controls

[Text](#)

## Value

Decimal	Hexadecimal	Constant
262144	0x00040000	msidbControlAttributesNoWrap

## Remarks

To set this attribute on a control, include the NoWrap bit in the Attributes column of the control's record in the [Control table](#).

See [Control Attributes](#) and [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Password Control Attribute

This attribute creates an edit control for entering passwords. The control displays each character as an asterisk (\*) as they are typed into the control.

Setting the Password Control Attribute prevents the installer from writing the property associated with the Edit control into the log file. For more information, see [Preventing Confidential Information from Being Written into the Log File](#).

## Valid Controls

[Edit control](#)

## Associated Bit Flags

This attribute does not use bit flags.

## Remarks

For more information, see [Control Attributes](#) and [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Position Control Attribute

This attribute determines the position of the control on the dialog box. The four fields of the attribute are: left, top, width, and height. Enter the control's width, height, and coordinates of the control's left corner into the Width, Height, X, and Y columns of the [Control table](#). Use [installer units](#) for units of length and distance.

## Valid Controls

All active controls.

## Associated Bit Flags

This attribute does not use bit flags.

## Remarks

See [Control Attributes](#) and the control you need to create under [Controls](#).

An installer unit is equal to one-twelfth the height of the 10-point MS Sans Serif font size.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Progress Control Attribute

This attribute measures how far the progress indicator bar is filled. The record contains two integers and a string. The first number is the progress, the second number is the range (the maximal possible number for the progress). On setting, the integers can be entered as integer fields or strings containing the integers. If the second number is missing it is assumed to be the default (1024). If the progress is larger than the range then it is changed to be the range. The third field is a string, the name of the action whose progress is displayed.

For related information, see [Authoring a ProgressBar Control](#), and [Adding Custom Actions to the ProgressBar](#).

## Valid Controls

[ProgressBar](#)

## Associated Bit Flags

This attribute does not use bit flags.

## Remarks

See [Control Attributes](#) and the control you need to create under [Controls](#).  
[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Progress95 Control Attribute

If this bit is set on a [ProgressBar control](#), the bar is drawn as a series of small rectangles in Microsoft Windows 95-style. If this bit is not set, the progress indicator bar is drawn as a single continuous rectangle.

## Valid Controls

[ProgressBar](#)

## Value

Decimal	Hexadecimal	Constant
65536	0x00010000	msidbControlAttributesProgress95

## Remarks

To set this attribute on a control, include the Progress95 bit in the Attributes column of the control's record in the [Control table](#).

See [Control Attributes](#) and the control you need to create under [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PropertyName Control Attribute

This is the name of the property associated with this control. If the [Indirect](#) bit is not set, the control displays or changes the value of the property having this name. This attribute is specified in the Property column of the [Control table](#). This attribute returns Null for inert controls.

## Valid Controls

All controls.

## Associated Bit Flags

This attribute does not use bit flags.

## Remarks

See [Control Attributes](#) and the control you need to create under [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PropertyValue Control Attribute

This attribute is the present value of the property displayed or changed by this control. If the [Indirect](#) bit is not set, this is the value of `PropertyName`. If the `Indirect` attribute bit is set, this is the value of `IndirectPropertyName`. If the attribute changes, the control reflects the new value as a string or integer depending on the [Integer](#) attribute of the control.

## Valid Controls

All active controls.

## Associated Bit Flags

This attribute does not use bit flags.

## Remarks

See [Control Attributes](#) and the control you need to create under [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PushLike Control Attribute

If this bit is set on a check box or a radio button group, the button is drawn with the appearance of a push button, but its logic stays the same. If the bit is not set, the controls are drawn in their usual style.

## Valid Controls

[CheckBoxRadioButtonGroup](#)

## Value

Decimal	Hexadecimal	Constant
131072	0x00020000	msidbControlAttributesPushLike

## Remarks

To set this attribute on a control, include the PushLike bit in the Attributes column of the control's record in the [Control table](#).

See [Control Attributes](#) and [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RAMDiskVolume Control Attribute

If this bit is set, the control shows all the volumes involved in the current installation plus all the RAM disk volumes. If this bit is not set the control lists volumes in the current installation.

## Valid Controls

[DirectoryComboVolumeCostList](#)  
[VolumeSelectCombo](#)

## Value

Decimal	Hexadecimal	Constant
1048567	0x00100000	msidbControlAttributesRAMDiskVolume

## Remarks

To set this attribute on a control, include the RAMDiskVolume bit in the Attributes column of the control's record in the [Control table](#).

See [Control Attributes](#) and [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RemoteVolume Control Attribute

If this bit is set, the control shows all the volumes involved in the current installation plus all the remote volumes. If this bit is not set, the control lists volumes in the current installation.

## Valid Controls

[DirectoryComboVolumeCostList](#)  
[VolumeSelectCombo](#)

## Value

Decimal	Hexadecimal	Constant
262144	0x00040000	msidbControlAttributesRemoteVolume

## Remarks

To set this attribute on a control, include the RemoteVolume bit in the Attributes column of the control's record in the [Control table](#).

See [Control Attributes](#) and [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RemovableVolume Control Attribute

If this bit is set, the control shows all the volumes involved in the current installation plus all the removable volumes. If this bit is not set, the control lists volumes in the current installation.

## Valid Controls

[DirectoryComboVolumeCostList](#)  
[VolumeSelectCombo](#)

## Value

Decimal	Hexadecimal	Constant
65536	0x00010000	msidbControlAttributesRemovableVolume

## Remarks

To set this attribute on a control, include the RemovableVolume bit in the Attributes column of the control's record in the [Control table](#).

See [Control Attributes](#) and [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RightAligned Control Attribute

If this style bit is set, text in the control is aligned to the right.

## Valid Controls

[GroupBoxScrollableText](#)  
[Text](#)  
[VolumeCostList](#)  
[CheckBox](#)  
[ComboBox](#)  
[DirectoryList](#)  
[DirectoryCombo](#)  
[Edit](#)  
[PathEdit](#)  
[ListBox](#)  
[ListView](#)  
[RadioButtonGroup](#)  
[SelectionTree](#)  
[VolumeSelectCombo](#)

Decimal	Hexadecimal	Description
64	0x00000040	msidbControlAttributesRightAligned

## Remarks

To set this attribute on a control, include the RightAligned bit in the Attributes column of the control's record in the [Control table](#).

See [Control Attributes](#) and [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RTLRO Control Attribute

If this style bit is set the text in the control is displayed in a right-to-left reading order.

## Valid Controls

GroupBoxProgressBar  
PushButton  
ScrollableText  
Text  
VolumeCostList  
CheckBox  
ComboBox  
DirectoryList  
DirectoryCombo  
Edit  
PathEdit  
ListBox  
ListView  
RadioButtonGroup  
SelectionTree  
VolumeSelectCombo

## Value

Decimal	Hexadecimal	Constant
32	0x00000020	msidbControlAttributesRTLRO

## Remarks

To set this attribute on a control, include the RTLRO bit in the Attributes column of the control's record in the [Control table](#).

See [Control Attributes](#) and [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Sorted Control Attribute

If this bit is set, the items listed in the control are displayed in a specified order. If the bit is not set, items are displayed in alphabetical order.

## Valid Controls

[ComboBoxListBox](#)  
[ListView Control](#)

## Value

Decimal	Hexadecimal	Constant
65536	0x00010000	msidbControlAttributesSorted

## Remarks

To sort the items in a [ComboBox](#), include the Sorted bit in the Attributes column of the [Control table](#) and specify the item order in the Order column of the [ComboBox table](#).

To sort the items in a [ListBox](#), include the Sorted bit in the Attributes column of the [Control table](#) and specify the item order in the Order column of the [ComboBox table](#).

To sort the items in a [ListView](#), include the Sorted bit in the Attributes column of the [Control table](#) and specify the order of items in the [ComboBox table](#).

See [Control Attributes](#) and [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Sunken Control Attribute

If this bit is set, the control is displayed with a sunken, three dimensional look. The effect of this style bit is different on different controls and versions of Windows. On some controls it has no visible effect. If the system does not support the Sunken control attribute, the control is displayed in the default visual style. If this bit is not set, the control is displayed with the default visual style.

## Valid Controls

All controls.

## Value

Decimal	Hexadecimal	Constant
4	0x00000004	msidbControlAttributesSunken

## Remarks

To set this attribute on a control, include the Sunken bit in the Attributes column of the control's record in the [Control table](#).

See [Control Attributes](#) and [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Text Control Attribute

This attribute is the text string displayed in the control. On setting, if the field 0 of the record is not null, the record is formatted using FormatText. If the field 0 is null, the first field of the record defines the text. On getting the value is always returned in the first field. For some controls this text may not be visible. In Windows the accelerator key for a control is defined by placing a "&" in front of the desired character in this string.

## Valid Controls

All controls except the [Line Control](#).

## Associated Bit Flags

None.

## Remarks

See [Control Attributes](#) and the control you need to create under [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# TimeRemaining Control Attribute

This attribute enables a text control to display the approximate number of minutes and seconds remaining for an installation. A record passed to the text control contains one integer representing the number of seconds remaining. The text control searches the [UIText table](#) for a string named TimeRemaining and formats the integer into an appropriate string displaying minutes and seconds.

## Valid Controls

[Text](#)

## Associated Bit Flags

This attribute does not use bit flags.

## Remarks

See [Control Attributes](#) and the control you need to create under [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Transparent Control Attribute

If the Transparent Control bit is set on a text control, the control is displayed transparently with the background showing through the control where there are no characters. If this bit is not set the text control is opaque.

## Valid Controls

[Text](#)

## Value

Decimal	Hexadecimal	Constant
65536	0x00010000	msidbControlAttributesTransparent

## Remarks

See [Control Attributes](#) and the control you need to create under [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UsersLanguage Control Attribute

If this bit flag is set, fonts are created using the user's default UI code page. If the bit flag is not set, fonts are created using the database code page.

## Valid Controls

[TextListBox](#)  
[ComboBox](#)

## Value

Decimal	Hexadecimal	Constant
1048576	0x00100000	msidbControlAttributesUsersLanguage

## Remarks

This control attribute can be used to display text that the user has entered into an [Edit](#) or [PathEdit](#) control.

The Edit control, PathEdit control, [DirectoryList control](#) and [DirectoryCombo control](#) always use the fonts created in the user's default UI code page. This can cause problems if additional languages have been installed on the operating system. In this case, the fonts on the computer may not be able to represent all the characters in the added languages. To determine what fonts can be used, look up the values in **HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\FontLink\SystemLink**.

For more information, see [Control Attributes](#) and [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Visible Control Attribute

If the Visible Control bit is set, the control is visible on the dialog box. If this bit is not set, the control is hidden on the dialog box. The visible or hidden state of the Visible control attribute can be later changed by a [Control Event](#).

## Valid Controls

All controls.

## Value

Decimal	Hexadecimal	Constant
1	0x00000001	msidbControlAttributesVisible

## Remarks

You can use the [ControlCondition table](#) to show or hide a control according to the value of a property or conditional statement. You can also show or hide a control by subscribing the control to a [ControlEvent](#). Enter the attribute's identifier in the Attribute column and the event's identifier in the Event column of the [EventMapping table](#).

See [Control Attributes](#) and [Controls](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Control Events

A ControlEvent specifies an action to be taken by the installer or a change in the attributes of one or more controls in a dialog box. For more information about ControlEvents, see [ControlEvent Overview](#).

The following table provides links to more information about particular ControlEvents.

Control event	Brief description of ControlEvent
<a href="#">ActionData</a>	Publishes data on the latest action.
<a href="#">ActionText</a>	Publishes the name of the present action.
<a href="#">AddLocal</a>	Notifies the installer to run features locally.
<a href="#">AddSource</a>	Notifies the installer to run features from their source.
<a href="#">CheckExistingTargetPath</a>	Notifies the installer to verify that the path can be written.
<a href="#">CheckTargetPath</a>	Notifies the installer to verify that the path is valid.
<a href="#">DirectoryListNew</a>	Notifies the DirectoryList control to create a new folder.
<a href="#">DirectoryListOpen</a>	Selects the directory in the DirectoryList control.
<a href="#">DirectoryListUp</a>	Notifies the DirectoryList control to select the parent of the present directory.
<a href="#">DoAction</a>	Dialog box notifies the installer to execute a custom action.
<a href="#">EnableRollback</a>	Used to turn rollback capabilities off and on.
<a href="#">EndDialog</a>	Notifies the installer to remove a modal dialog box.
<a href="#">IgnoreChange</a>	Published by the DirectoryList control when a folder is highlighted but not opened.
<a href="#">MsiLaunchApp</a>	This control event runs a specified file.

	<b>Windows Installer 4.5 and earlier:</b> Not supported.
MsiPrint	Enables the user to print the contents of <a href="#">ScrollableText Control</a> .  <b>Windows Installer 4.5 and earlier:</b> Not supported.
NewDialog	Notifies the installer to change a modal dialog box into another dialog box.
Reinstall	Initiates a reinstallation of features.
ReinstallMode	Specifies the validation mode during a reinstallation.
Remove	Notifies the installer when features are selected for removal.
Reset	Resets all the property values to the default values used when the dialog box was created.
RmShutdownAndRestart	Use the Restart Manager to shutdown all applications that have files in use and to restart them at the end of the installation.
ScriptInProgress	Displays a string while the execution script is compiled.
SelectionAction	Published by SelectionTree to describe an item.
SelectionBrowse	Published by SelectionTree to spawn a dialog box.
SelectionDescription	Published by SelectionTree to provide a string in the Description field of the Feature Table.
SelectionNoItems	Used by SelectionTree to delete text or disable buttons.
SelectionPath	Published by SelectionTree to provide the path of an item.

<a href="#">SelectionPathOn</a>	Published by SelectionTree to indicate whether there is a path associated with a feature.
<a href="#">SelectionSize</a>	Published by SelectionTree control to provide the size of an item.
<a href="#">SetInstallLevel</a>	The installer changes installation level to a specified value.
<a href="#">SetProgress</a>	Published by the installer to provide installation progress.
<a href="#">SetProperty</a>	Sets a specified property.
<a href="#">SetTargetPath</a>	Notifies the installer to check and set a path.
<a href="#">SpawnDialog</a>	Notifies the installer to create a child of a modal box.
<a href="#">SpawnWaitDialog</a>	Triggers a specified dialog box.
<a href="#">TimeRemaining</a>	Published by the installer to provide the time remaining in the progress sequence.
<a href="#">ValidateProductID</a>	Sets ProductID to the full Product ID.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ActionData ControlEvent

The installer uses this event to publish data on the latest action. The data can be displayed on a dialog box by a [Text Control](#) that subscribes to this ControlEvent. This event should be authored in the [EventMapping table](#).

This ControlEvent can be handled by a user interface run at the [basic UI](#), [reduced UI](#), or [full UI](#) levels. For information about UI levels, see [User Interface Levels](#).

## Published By

This ControlEvent is published by the installer.

## Argument

This ControlEvent does not use an argument.

## Action on Subscribers

The subscribers are hidden when a new action starts and shown when new action data arrives from the installer.

## Typical Use

A [Text Control](#) on a modeless dialog box subscribes to this event through the [EventMapping table](#). The [Text Control Attribute](#) is listed in the Attributes field of this table to receive the current action data. Typically used to display the names of the files copied during file copy.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ActionText ControlEvent

The installer uses this event to publish the name of the present action. The name can be displayed on a dialog box by a [Text Control](#) that subscribes to this ControlEvent. This event should be authored in the [EventMapping table](#).

This ControlEvent can be handled by a user interface run at the [basic UI](#), [reduced UI](#), or [full UI](#) levels. For information about UI levels, see [User Interface Levels](#).

## Published By

This ControlEvent is published by the installer.

## Argument

This ControlEvent does not use an argument.

## Action on Subscribers

The subscribers are shown when a new progress data arrives from the installer.

## Typical Use

A [Text Control](#) on a modeless dialog subscribes to this event through the [EventMapping table](#). The [Text Control Attribute](#) should be listed in the Attributes field of this table to receive the name of the current action.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# AddLocal ControlEvent

This event notifies the installer, while keeping the present dialog running, that a feature or all features are to be run locally. This event can be published by a [PushButton Control](#), [Checkbox Control](#), or a [SelectionTree control](#). This event should be authored into the [ControlEvent table](#).

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

## Published By

This ControlEvent is published by the installer.

## Argument

A string, either the name of the feature or "ALL".

## Action on Subscribers

This ControlEvent does not perform an action on subscribers.

## Typical Use

A [PushButton](#) control on a modal dialog box is tied to this event and used to notify the installer without stopping the dialog box.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# AddSource ControlEvent

This event notifies the installer, while keeping the present dialog running, that a feature or all features are to be run from source. This event can be published by a [PushButton Control](#), [Checkbox Control](#), or a [SelectionTree control](#). This event should be authored into the [ControlEvent table](#).

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

## Published By

This ControlEvent is published by the installer.

## Argument

A string, either the name of the feature or "ALL".

## Action on Subscribers

This ControlEvent does not perform an action on subscribers.

## Typical Use

A [PushButton](#) control on a modal dialog box is tied to this event and used to notify the installer without stopping the dialog box.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# CheckExistingTargetPath ControlEvent

This event notifies the installer that it has to verify that the selected path is writable. If the path is cannot be written, then the event blocks further ControlEvents associated with the control.

This event can be published by a [PushButton Control](#) or a [SelectionTree control](#). This event should be authored into the [ControlEvent table](#).

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

## Published By

This ControlEvent is published by the installer.

## Argument

The name of the property containing the path. If the property is indirected, then the property name is enclosed in square brackets.

## Action on Subscribers

This ControlEvent does not perform an action on subscribers.

## Typical Use

A [PushButton](#) control on a browse dialog is tied to this event in the [ControlEvent](#) table to check the selected path before returning to the selection dialog.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# CheckTargetPath ControlEvent

This event notifies the installer that it has to verify that the selected path is valid. If the path is not valid, then this event blocks further ControlEvents associated with the control.

This event can be published by a [PushButton Control](#) or a [SelectionTree control](#). This event should be authored into the [ControlEvent table](#).

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

## Published By

This ControlEvent is published by the installer.

## Argument

The name of the property containing the path. If the property is indirected, then the property name is enclosed in square brackets.

## Action on Subscribers

This ControlEvent does not perform an action on subscribers.

## Typical Use

A [PushButton](#) control on a browse dialog box is tied to this event in the [ControlEvent](#) table to check the selected path before returning to the selection dialog box.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IgnoreChange ControlEvent

The IgnoreChange ControlEvent is published by the [DirectoryList control](#) when the selected folder is changed from one directory to another directory in the control. This event should be authored in the [EventMapping table](#).

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

## Published By

[DirectoryList](#)

## Argument

This ControlEvent does not use an argument.

## Action on Subscribers

This ControlEvent does not perform an action on subscribers.

## Typical Use

The [DirectoryCombo control](#) commonly employs the IgnoreChange ControlEvent.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DirectoryListNew ControlEvent

This event notifies the [DirectoryList control](#) that a new folder must be created; it creates the new folder, and selects the name field of the folder for editing. The default name of the new folder may be authored in the [UIText table](#). Enter "NewFolder" into the Key column. Enter the value for the default name of the new folder into the Text column. This value must be in the form of a [Filename](#) column data type and use the SFN|LFN syntax. If this value is not present in the UIText table or is an invalid value, the installer uses a default value of "Fldr|New Folder." For related information, see [Browse Dialog](#).

This event should be published by a [PushButton Control](#) located on the same dialog box as the control subscribing to this event. The event should be authored in the [ControlEvent table](#).

This ControlEvent requires the user interface to be run at the [full UI](#) level. This event will not work with a [reduced UI](#) or [basic UI](#). For information, see [User Interface Levels](#).

Note that if this ControlEvent is called again when a new folder already exists, a second new folder will not be created. In this case, calling DirectoryListNew selects the existing new folder's name for editing.

## Published By

[DirectoryList](#)

## Argument

This ControlEvent does not use an argument.

## Action on Subscribers

This ControlEvent does not perform an action on subscribers.

## Typical Use

A [PushButton](#) control on the same modal dialog box as the DirectoryList

is used to trigger the creation of a new folder.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DirectoryListOpen ControlEvent

This event selects and highlights a directory in the [DirectoryList control](#) that the user has chosen to open. For related information, see [Browse Dialog](#).

This event should be published by a [PushButton Control](#) located on the same dialog box as the control subscribing to this event. The event should be authored in the [ControlEvent table](#).

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

If no directory has been selected in the [DirectoryList control](#), a DirectoryListOpen event disables any other controls that publish a DirectoryListOpen event.

## Published By

[DirectoryList](#)

## Argument

This ControlEvent does not use an argument.

## Action on Subscribers

This ControlEvent performs no action on subscribers.

## Typical Use

A [PushButton](#) control on the same modal dialog box as the [DirectoryList](#) is used to trigger stepping down in the path.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DirectoryListUp ControlEvent

This event notifies the [DirectoryList control](#) that the user wants to select the parent of the present directory. For related information, see [Browse Dialog](#).

This event should be published by a [PushButton Control](#) located on the same dialog box as the control subscribing to this event. The event should be authored in the [ControlEvent table](#).

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

If the current directory is the root directory of the drive, a DirectoryListUp event disables any other controls that publish a DirectoryListUp event.

## Published By

[DirectoryList](#)

## Argument

This ControlEvent does not use an argument.

## Action on Subscribers

This ControlEvent does not perform an action on subscribers.

## Typical Use

A [PushButton](#) control on the same modal dialog box as the [DirectoryList](#) is used to trigger stepping up in the path.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DoAction ControlEvent

The DoAction ControlEvent notifies the installer to execute a custom action. This event can be published by a [PushButton](#) control, [CheckBox](#) control, or a [SelectionTree](#) control. This event should be authored into the [ControlEvent](#) table.

Note that custom actions launched by a DoAction ControlEvent can send a message with the [Message Method](#), but cannot send a message with [MsiProcessMessage](#). On systems prior to Windows Server 2003, custom actions launched by a DoAction ControlEvent cannot send messages with [MsiProcessMessage](#) or [Message Method](#). For more information, see [Sending Messages to Windows Installer Using MsiProcessMessage](#).

This ControlEvent requires the user interface to be run at the [full UI](#) level. This event will not work with a [reduced UI](#) or [basic UI](#). For more information, see [User Interface Levels](#).

## Published By

This ControlEvent is published by the installer.

## Argument

A string, the name of the custom action to be executed.

## Action on Subscribers

This ControlEvent does not perform an action on subscribers.

## Typical Use

A PushButton control on a dialog box is tied to this event in the ControlEvent table to invoke a custom action.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# EnableRollback ControlEvent

This ControlEvent can be used to turn on or turn off the installer's rollback capabilities.

This event can be published by a [PushButton Control](#) or a [SelectionTree control](#). This event should be authored into the [ControlEvent table](#).

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

## Published By

This ControlEvent is published by the installer.

## Argument

False turns off the installer's rollback capabilities. True turns on the installer's rollback capabilities.

## Action on Subscribers

This ControlEvent does not perform an action on subscribers.

## Typical Use

Can be used to turn off rollback capabilities if the installer detects that there is insufficient disk space available to complete the installation. For this case, the ControlEvent should be used with the [OutOfDiskSpace](#), [OutOfNoRbDiskSpace](#), and the [PROMPTROLLBACKCOST](#) properties.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# EndDialog ControlEvent

This event notifies the installer to remove a modal dialog box. In all cases the installer removes the present dialog box.

This event can be published by a [PushButton Control](#) or a [SelectionTree control](#). This event should be authored into the [ControlEvent table](#).

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

The following table lists the action of the event resulting from different arguments entered into the [ControlEvent table](#).

Argument	Action by the installer
Exit	The wizard sequence is closed and the control returns to the installer with the UserExit value. This argument cannot be used in a dialog box that is a child of another dialog box.
Retry	The wizard sequence is closed and the control returns to the installer with the Suspend value. This argument cannot be used in a dialog box that is a child of another dialog box.
Ignore	The wizard sequence is closed and the control returns to the installer with the Finished value. This argument cannot be used in a dialog box that is a child of another dialog box.
Return	The control returns to the parent of the present dialog box, or if there is no parent, the control returns to the installer with the Success value.

## Published By

This ControlEvent is published by the installer.

## Argument

On regular dialog boxes the Argument column of the [ControlEvent table](#)

can be "Return", "Exit", "Retry", or "Ignore".

On error dialog boxes the Argument column of the [ControlEvent table](#) can be "ErrorOk", "ErrorCancel", "ErrorAbort", "ErrorRetry", "ErrorIgnore", "ErrorYes", or "ErrorNo".

## Action on Subscribers

None.

## Typical Use

A [PushButton](#) control on a modal dialog is tied to this event in the [ControlEvent](#) table to close a dialog box.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiLaunchApp ControlEvent

This control event runs a specified file. If the file does not exist, or if the event fails, Windows Installer logs the error in the verbose log without displaying a dialog box containing an error message.

**Windows Installer 4.5 or earlier:** Not supported. This ControlEvent is available beginning with Windows Installer 5.0.

## Published By

This ControlEvent is published by the installer.

## Argument

The fields of the argument value are delimited by spaces. The first field contains a string value that specifies the file that is to be run. Use a string value of [#filekey] to identify the file and replace *filekey* with the file's identifier appearing in the File column of the [File](#) table. Any remaining fields of the argument can contain parameters being used by the file being run.

## Action on Subscribers

This ControlEvent performs no actions on subscribers.

## Typical Use

To enable a user to choose to run a file at the end of an installation. This event can be conditioned on a property set by a [CheckBox](#) control displayed on the final dialog box of the installation. The CheckBox control should not be displayed during the removal of the package.

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiPrint ControlEvent

This event is published by the [ScrollableText Control](#) to enable the user to print the contents of that control.

**Windows Installer 4.5 or earlier:** Not supported. This ControlEvent is available beginning with Windows Installer 5.0.

This event should be subscribed to by a [PushButton Control](#) located on the same dialog box as the ScrollableText Control. TheMsiPrint ControlEvent should be authored in the [ControlEvent table](#).

## Published By

ScrollableText

## Argument

This ControlEvent does not use an argument.

## Action on Subscribers

This ControlEvent does not perform an action on subscribers.

## Typical Use

A [PushButton](#) control on the same dialog box as the ScrollableText Control can be used to display a modal dialog box enabling the user to print the contents of the ScrollableText Control.

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# NewDialog ControlEvent

This event notifies the installer to transition a modal dialog box to another dialog box. The installer removes the present dialog box and creates the new one with the name indicated in the argument.

This event can be published by a [PushButton Control](#) or a [SelectionTree control](#). This event should be authored into the [ControlEvent table](#).

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

## Published By

This ControlEvent is published by the installer.

## Argument

A string that is the name of the new dialog.

## Action on Subscribers

This ControlEvent does not perform an action on subscribers.

## Typical Use

A [PushButton](#) control on a modal dialog box is tied to this event in the [ControlEvent](#) table to signal a transition to the next or previous dialog box of the same wizard sequence.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Reinstall ControlEvent

The Reinstall ControlEvent allows the author to initiate a reinstall of some or all features, while the current dialog box is running.

This event can be published by a [PushButton control](#) or a [SelectionTree control](#). This event should be authored into the [ControlEvent table](#), and requires the user interface to be run at the *full UI* level. This event does not work with a *reduced UI* or *basic UI*. For more information, see [User Interface Levels](#).

## Published By

This ControlEvent is published by the installer.

## Argument

A string that is either the name of the feature or "ALL".

## Action on Subscribers

This ControlEvent does not perform an action on subscribers.

## Typical Use

This event is tied to a [PushButton](#) control on a modal dialog box. The same [PushButton](#) control should also be tied to a [ReinstallMode ControlEvent](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ReinstallMode ControlEvent

The ReinstallMode ControlEvent allows the author to specify the validation mode or modes during a reinstallation, and while the current dialog box is running.

This event can be published by a [PushButton Control](#) or a [SelectionTree control](#). This event should be authored into the [ControlEvent table](#), and requires the user interface to be run at the *full UI* level. This event does not work with a *reduced UI* or *basic UI*. For more information, see [User Interface Levels](#).

## Published By

This ControlEvent is published by the installer.

## Argument

A string. For a list of possible values, see [REINSTALLMODE](#) property.

## Action on Subscribers

This ControlEvent does not perform an action on subscribers.

## Typical Use

This event is tied to a [PushButton](#) control on a modal dialog box. The same [PushButton](#) control should also be tied to a [Reinstall](#) ControlEvent.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Remove ControlEvent

The installer is notified through this event when a feature or all features are selected for removal while keeping the present dialog box running.

This event can be published by a [PushButton control](#), [CheckBox control](#), or a [SelectionTree control](#). This event should be authored into the [ControlEvent table](#).

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

## Published By

This ControlEvent is published by the installer.

## Argument

A string that is either the name of the feature or "ALL".

## Action on Subscribers

This ControlEvent does not perform an action on subscribers.

## Action by Publisher When Subscriber Activated

The installer keeps the present dialog box running and notifies the installer.

## Typical Use

A [PushButton](#) control on a modal dialog box tied to this event.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Reset ControlEvent

The dialog box is reset. In other words, all the controls on the dialog box are called to undo the property changes they have performed. This event resets all the property values to what they were when the dialog box was created.

This event can be published by a [PushButton Control](#) or a [SelectionTree control](#). This event should be authored into the [ControlEvent table](#).

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

There is a case, which should occur rarely in practice, that is not handled properly by this ControlEvent. If there are two or more controls on the same dialog box linked indirectly to the same property and at least one of them started having some other property, then this property value will sometimes be reset to its second value.

## Published By

This ControlEvent is published by the installer.

## Argument

None.

## Action on Subscribers

None.

## Typical Use

A [PushButton](#) control on a modal dialog box is used to reset all the values on the dialog box and to cancel all the changes on the page.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RmShutdownAndRestart ControlEvent

This event notifies the Windows Installer to use the Restart Manager to shutdown all applications that have files in use and to restart them at the end of the installation.

This ControlEvent has no effect if any of the following are true.

- The operating system running the installation is not Windows Vista or Windows Server 2008.
- Interactions with the Restart Manager have been disabled by the [MSIRESTARTMANAGERCONTROL](#) property or the [DisableAutomaticApplicationShutdown](#) policy.
- There is currently no open Restart Manager session.
- Any calls from the Windows Installer to the Restart Manager returns a failure.

## Typical Use

The RmShutdownAndRestart control event is published only by a [PushButton](#) control on the [MsiRMFilesInUse Dialog](#) box.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ScriptInProgress ControlEvent

The installer uses this event to display an informational string while the installation's execution script is being compiled. The informational string can be displayed on a dialog box by a [Text Control](#) that subscribes to this ControlEvent. This event should be authored in the [EventMapping table](#).

This ControlEvent can be handled by a user interface run at the [basic UI](#), [reduced UI](#), or [full UI](#) levels. For information about UI levels, see [User Interface Levels](#).

## Published By

This ControlEvent is published by the installer.

## Argument

None.

## Action on Subscribers

A [Text control](#) subscribing to ScriptInProgress will display text string specified in [UIText table](#).

## Typical Use

While the execution script is being compiled, the installer displays a ProgressBar indicating the time remaining before the beginning of script execution. The package author can display a preliminary message at this time explaining the ProgressBar. To display a preliminary message, include a [Text control](#) on the same modeless dialog box as the ProgressBar. Specify that this Text control subscribe to the ScriptInProgress ControlEvent via the [EventMapping table](#). Include an entry in the [UIText table](#) with ScriptInProgress specified in the Key field. Specify the preliminary message as a text string in the Text field of the UIText table. Then during script compilation, the installer will display this string within the text control. The displayed text disappears as soon as the script compilation is finished.

The same text control that subscribes to the ScriptInProgress ControlEvent can also subscribe to the [TimeRemaining ControlEvent](#). In this case, as text of the preliminary ScriptInProgress string disappears, it is replaced by the "Time Remaining: xx minutes" string.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SelectionAction ControlEvent

The [SelectionTree control](#) uses this event to publish a string describing the highlighted item. The string is one of the "Sel\*" strings from the [UIText table](#). This event should be authored in the [EventMapping table](#).

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

This event can only affect controls that are located on the same dialog box as the SelectionTree control.

## Published By

[SelectionTree](#)

## Argument

None.

## Action on Subscribers

None.

## Typical Use

A [Text](#) control in the same modal dialog box as the SelectionTree should subscribe to this event via the [EventMapping table](#). The Text Control displays the explanation of the selected item.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SelectionBrowse ControlEvent

The [SelectionTree control](#) uses the SelectionBrowse event to spawn a **Browse** dialog box making it possible to modify the path of the highlighted item.

This event should be published by a [PushButton Control](#) located on the same dialog box as the control subscribing to this event. The event should be authored in the [ControlEvent table](#).

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

Any controls that publish a SelectionBrowse event become disabled if a feature has been selected that is already installed, not configurable, or not selected for local installation. To be configurable, the feature must have a [public property](#) entered in the Directory\_ field of the [Feature table](#).

## Published By

[SelectionTree](#)

## Argument

The name of the dialog to be spawned.

## Action on Subscribers

None.

## Typical Use

A [PushButton](#) control on the same modal dialog box as the SelectionTree uses this event to trigger the **Browse** dialog box.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SelectionDescription ControlEvent

The [SelectionTree control](#) uses the SelectionDescription event to publish a string that contains the description for the highlighted item. This string is the Description field of the [Feature table](#). This event should be authored in the [EventMapping table](#).

This event can only affect controls that are located on the same dialog box as the SelectionTree control.

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For more information, see [User Interface Levels](#).

## Published By

[SelectionTree](#)

## Argument

None.

## Action on Subscribers

None.

## Typical Use

A [Text](#) control on the same modal dialog as the SelectionTree subscribes to this ControlEvent via the [EventMapping table](#). The Text Control uses this event to display the description of the highlighted item.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SelectionNoItems ControlEvent

The [SelectionTree control](#) uses the SelectionNoItems event to delete obsolete item description text or to disable buttons that have become useless. This event should be authored in the [EventMapping table](#).

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

This event can only affect controls that are located on the same dialog box as the SelectionTree control.

## Published By

[SelectionTree control](#) if the selection has no nodes.

## Argument

None.

## Action on Subscribers

Deletes obsolete item description text or disables obsolete buttons.

## Typical Use

May be used to disable the Next, Reset, and DiskCost buttons on the [Selection Dialog](#) when a selection has no nodes and these buttons become useless.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SelectionPath ControlEvent

The [SelectionTree control](#) uses the SelectionPath event to publish the path for the highlighted item. If the item is selected to run from source, then this is the path of the source. If the item is selected to be absent, then the string is the **AbsentPath** string from the [UIText table](#). This event should be authored in the [EventMapping table](#).

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

This event can only affect controls that are located on the same dialog box as the SelectionTree control.

## Published By

[SelectionTree](#)

## Argument

None.

## Action on Subscribers

None.

## Typical Use

A [Text](#) control on the same modal dialog as the SelectionTree should subscribe to the event via the [EventMapping table](#). The Text Control displays the path of the highlighted item.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SelectionPathOn ControlEvent

The [SelectionTree control](#) uses the SelectionPathOn event to publish a Boolean value indicating whether there is a selection path associated with the currently selected feature. This event should be authored in the [EventMapping table](#).

This event can only affect controls that are located on the same dialog box as the SelectionTree control.

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

The SelectionPathOn ControlEvent is never published during a [Maintenance Installation](#).

## Published By

[SelectionTree](#)

## Argument

None.

## Action on Subscribers

None.

## Typical Use

A [Text](#) control on the same modal dialog as the SelectionTree should subscribe to the event via the [EventMapping table](#). The Text Control displays the caption of the selection path. This text control is visible or hidden depending on the event.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SelectionSize ControlEvent

The [SelectionTree control](#) uses the SelectionSize event to publish the size of the highlighted item. If it is a parent item, then the number of children selected is also published. The string is one of the **SelChildCostPos**, **SelChildCostNeg**, **SelParentCostPosPos**, **SelParentCostPosNeg**, **SelParentCostNegPos** or **SelParentCostNegNeg** strings from the [UIText table](#). This event should be authored in the [EventMapping table](#).

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

This event can only affect controls that are located on the same dialog box as the SelectionTree control.

## Published By

[SelectionTree](#)

## Argument

None.

## Action on Subscribers

None.

## Typical Use

A [Text](#) control on the same modal dialog as the SelectionTree Control via the [EventMapping table](#). The Text Control uses this event to display the size of the highlighted item.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SetInstallLevel ControlEvent

The SetInstallLevel event changes the installation level to the value specified by the argument.

This event can be published by a [PushButton Control](#) or a [SelectionTree control](#). This event should be authored into the [ControlEvent table](#).

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

## Published By

This ControlEvent is published by the installer.

## Argument

An integer that is the new value of the installation level.

## Action on Subscribers

None.

## Typical Use

A [PushButton](#) control on a modal dialog box is tied to this event in the [ControlEvent](#) table to change the installation level.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SetProgress ControlEvent

The installer uses the SetProgress event to publish information on the installation's progress. A [ProgressBar Control](#) or [Billboard Control](#) should subscribe to the event via the [EventMapping table](#) by naming the Action whose progress is being indicated. This event should be authored in the [EventMapping table](#).

This ControlEvent can be handled by a user interface run at the [basic UI](#), [reduced UI](#), or [full UI](#) levels. For information about UI levels, see [User Interface Levels](#).

## Published By

This ControlEvent is published by the installer.

## Argument

None.

## Action on Subscribers

None.

## Typical Use

A [ProgressBar](#) control on a modeless dialog box subscribes to this event using the [Progress](#) attribute to receive the progress information.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SetProperty ControlEvent

The syntax for the SetProperty event is different than for other ControlEvents. In place of the name of the event one puts the property in square brackets: [property\_name]. The argument is the new value of the property. To unset the property, the argument must be {}. This is necessary, because the argument is a primary key in the table and so cannot be null. The argument will be formatted using the FormatText method, therefore the argument can contain any expression that the FormatText method can handle. The property values are evaluated at the moment of the ControlEvent.

This event can be published by a [PushButton Control](#), [CheckBox Control](#), or a [SelectionTree control](#). This event should be authored into the [ControlEvent table](#).

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

## Published By

This ControlEvent is published by the installer.

## Argument

The new value of the property. {} for null.

## Action on Subscribers

None.

## Typical Use

A [PushButton](#) control on a dialog box linked to this event so it changes a property when the button is pushed.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SetTargetPath ControlEvent

The SetTargetPath event notifies the installer to check and set the selected path. If the path is not valid to be written to, then the installer blocks further ControlEvents associated with the control.

This event can be published by a [PushButton Control](#) or a [SelectionTree control](#). This event should be authored into the [ControlEvent table](#).

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

## Published By

This ControlEvent is published by the installer.

## Argument

The name of the property containing the path. If the property is indirected, then the property name is enclosed in square brackets.

## Action on Subscribers

None.

## Typical Use

A [PushButton](#) control on a browse dialog is tied to this event in the [ControlEvent](#) table to check the selected path before returning to the selection dialog.

## Remarks

Do not attempt to configure the target path if the components using those paths are already installed for the current user or for a different user. Check the [ProductState](#) property before publishing the SetTargetPath ControlEvent to determine if the product containing the component is

installed.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SpawnDialog ControlEvent

The SpawnDialog ControlEvent notifies the installer to create a child of a modal dialog box while keeping the present dialog box running.

This event can be published by a [PushButton Control](#) or a [SelectionTree control](#). This event should be authored into the [ControlEvent table](#).

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

## Published By

This ControlEvent is published by the installer.

## Argument

A string that is the name of the new dialog box.

## Action on Subscribers

None.

## Typical Use

A [PushButton](#) control on a modal dialog is tied to this event in the [ControlEvent](#) table to create a child dialog, such as an "Are you sure you want to cancel?" dialog box.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SpawnWaitDialog ControlEvent

The SpawnWaitDialog ControlEvent triggers the dialog box specified by the Argument column of the [ControlEvent table](#), if the expression in the Condition column evaluates as FALSE. The dialog box remains up for as long as the conditional expression remains FALSE and is removed as soon as the condition evaluates TRUE.

This event can be published by a [PushButton Control](#) or a [SelectionTree control](#). This event should be authored into the [ControlEvent table](#).

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

## Published By

This ControlEvent is published by the installer.

## Argument

A dialog box in the [Dialog table](#).

## Action on Subscribers

None.

## Typical Use

The SpawnWaitDialog ControlEvent can be used to wait for any background task the completion of which can be tested using a conditional expression such as the state of a property. For an example of using the SpawnWaitDialog ControlEvent, see the section [Authoring a Conditional "Please wait . . ." Message Box](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# TimeRemaining ControlEvent

The installer uses the TimeRemaining event to publish the approximate time remaining, in seconds, for the current progress sequence. The time remaining can be displayed on a dialog box by a [Text Control](#) that subscribes to this ControlEvent. This event should be authored in the [EventMapping table](#).

This ControlEvent can be handled by a user interface run at the [basic UI](#), [reduced UI](#), or [full UI](#) levels. For information about UI levels, see [User Interface Levels](#).

## Published By

This ControlEvent is published by the installer.

## Argument

None.

## Action on Subscribers

None.

## Typical Use

A [Text Control](#) on a modeless dialog box subscribes to this event through the [EventMapping table](#) and uses the [TimeRemaining attribute](#) to display the time remaining.

The same text control that subscribes to the TimeRemaining ControlEvent can also subscribe to the [ScriptInProgress ControlEvent](#). In this case, as the preliminary ScriptInProgress message string, it is replaced by the "Time Remaining: xx minutes" string.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ValidateProductID ControlEvent

The ValidateProductID event sets the **ProductID** property to the full Product ID. If the validation is unsuccessful, then this event puts up an error message and modal dialog box for a user entry of the Product ID.

This event can be published by a [PushButton Control](#) or a [SelectionTree control](#). This event should be authored into the [ControlEvent table](#).

This ControlEvent requires the user interface to be run at the *full UI* level. This event will not work with a *reduced UI* or *basic UI*. For information, see [User Interface Levels](#).

## Published By

This ControlEvent is published by the installer.

## Argument

None.

## Action on Subscribers

None.

## Typical Use

A [PushButton](#) control on a modal dialog box is tied to this event and is used to check the user's entry of the Product ID. If the user's entry is valid, then the next dialog box will open.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Standard Actions

An action encapsulates a typical function performed during the installation or maintenance of an application. Windows Installer has many built-in standard actions. Developers of installation packages can also author their own [custom actions](#).

Examples of installer standard actions include:

- [CreateShortcuts action](#): Manages the creation of shortcuts to files installed with the current application. This action uses the [Shortcut table](#) for its data.
- [InstallFiles action](#): Copies selected files from the source directory to the destination directory. This action uses the [File table](#) for its data.
- [WriteRegistryValues action](#): Sets up registry information the application requires in the registry. This action uses the [Registry table](#) for its data.

The following sections discuss standard actions.

- [About Standard Actions](#)
- [Using Standard Actions](#)
- [Standard Actions Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# About Standard Actions

The Windows Installer has many built-in standard actions that are used in the [sequence tables](#). The topics in this section cover information about standard actions.

- [Action Execution Order](#)
- [Actions with Sequencing Restrictions](#)
- [Actions without Sequencing Restrictions](#)

See also [Using Standard Actions](#) or [Standard Actions Reference](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Action Execution Order

The order of action execution is determined by the sequence of actions that have been authored into the [sequence tables](#) and by the order in which the installer runs the sequence tables. For details, see the suggested action sequences in [Using a Sequence Table](#).

The installer runs sequence tables in response to a request for an installation, [advertisement](#), or an [administrative installation](#). For example, in response to using the /I, /J, or /A [command line options](#), the [INSTALL](#), [ADVERTISE](#), and [ADMIN](#) actions are not called from within the action sequence. These high-level actions are instead passed to the installer when the installer is initialized.

If the installer is passed the [INSTALL](#) action and the installation package has been authored with a user interface, the installer first runs the actions in [InstallUISequence table](#) and then executes the actions in the [InstallExecuteSequence table](#) in order. If the package has no user interface, the installer executes the actions in the [InstallExecuteSequence table](#) in order.

If the installer is passed the [ADMIN](#) action, and the installation package has been authored with a user interface, the installer first runs the [AdminUISequence table](#) and then runs the [AdminExecuteSequence table](#). If the package has no user interface, the installer runs the [AdminExecute table](#).

If the installer is passed the [ADVERTISE](#) action, the installer runs the [AdvtExecuteSequence table](#).

**Note** The installer does not use the [AdvtUISequence table](#). The [AdvtUISequence table](#) should not exist in the installation database or it should be left empty.

When the installer runs a sequence table, it executes actions in the order of the sequence numbers listed in the [Sequence](#) column. The action order is always linear with no branching or looping. Package developers can conditionally prevent a particular action from being executed by authoring a logical expression into the [Condition](#) column. The installer skips the action whenever the condition evaluates to [False](#). See [Using a Sequence Table](#) and [Conditional Statement Syntax](#).

All sequence tables have the following columns.

<b>Column</b>	<b>Description</b>
Action	The primary key for the table; the action name must be unique.
Condition	A Boolean expression used to determine whether to perform the action. The action is executed if this field is either blank or contains an expression that evaluates to True. The action is not executed if the expression evaluates to False.
Sequence	A relative sequence number used to determine the order in which actions are executed.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Actions with Sequencing Restrictions

The sequencing of many actions in the actions tables is restricted. See [Standard Actions Reference](#) for a complete list of actions with links to detailed sequencing information.

The sequencing of the following actions is restricted.

[AllocateRegistrySpace](#)

[BindImage](#)

[CCPSearch](#)

[CostFinalize](#)

[CostInitialize](#)

[CreateFolders](#)

[CreateShortcuts](#)

[DeleteServices](#)

[DuplicateFiles](#)

[ExecuteAction](#)

[FileCost](#)

[ForceReboot](#)

[InstallExecute](#)

[InstallFiles](#)

[InstallFinalize](#)

[InstallInitialize](#)

[InstallValidate](#)

[LaunchConditions](#)

[MoveFiles](#)

[InstallODBC](#)

InstallServices  
PatchFiles  
PublishFeatures  
PublishProduct  
RegisterClassInfo  
RegisterExtensionInfo  
RegisterFonts  
RegisterMIMEInfo  
RegisterProgIdInfo  
RegisterTypeLibraries  
RemoveEnvironmentStrings action  
RemoveFiles  
RemoveDuplicateFiles  
RemoveFolders  
RemoveIniValues  
RemoveRegistryValues  
RemoveShortcuts  
ResolveSource  
SelfRegModules  
SelfUnregModules  
SetODBCFolders  
StartServices  
StopServices  
UnregisterClassInfo  
UnregisterExtensionInfo  
UnregisterFonts  
UnregisterMIMEInfo

[UnregisterProgIdInfo](#)

[UnregisterTypeLibraries](#)

[WriteEnvironmentStrings](#)

[WriteIniValues](#)

[WriteRegistryValues](#)

[RegisterComPlus](#)

[UnregisterComPlus](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Actions without Sequencing Restrictions

The sequencing of the following actions is unrestricted. These actions can be placed anywhere in the action sequence.

[AppSearch](#)

[DisableRollback](#)

[InstallAdminPackage](#)

[RemoveODBC](#)

[ProcessComponents](#)

[PublishComponents](#)

[RegisterProduct](#)

[RegisterUser](#)

[RMCCPSearch](#)

[ScheduleReboot](#)

[UnpublishComponents](#)

[UnpublishFeatures](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Standard Actions

An action is executed in the Windows Installer either by calling the [MsiDoAction](#) function or including the action in a sequence table. Because most actions encapsulate a single purpose, the most common way to use actions is to order a series of actions into a sequence to accomplish a larger task. The installer has three standard top-level actions that call an associated set of sequence tables. These associated sequence tables may contain standard actions, custom actions, and user-interface elements. Each action in a sequence table has an associated sequence number and may also have an associated conditional expression. All actions in a sequence table are visited in order and are only executed if the conditional expression evaluates to True.

While a standard action can have any sequence number associated with it, many have sequence restrictions which must be followed for the action to function properly. For example the [FileCost action](#), must be called after the [CostInitialize action](#). For more information on standard action sequencing restrictions, see [Actions with Sequencing Restrictions](#), [Actions without Sequencing Restrictions](#), or [Standard Actions Reference](#).

The following topics provide more information about using standard actions.

- [Publishing Products, Features, and Components](#)
- [File Searching](#)
- [File Costing](#)
- [File Installation](#)
- [Modifying the Registry](#)
- [Running Actions](#)

See also [About Standard Actions](#) or [Standard Actions Reference](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Publishing Products, Features, and Components

To [publish](#) a product, component, or feature, use one of the publishing actions. The [PublishProduct action](#) registers the product information with the system. After executing the PublishProduct action, publish the components with the [PublishComponents action](#), which in turn uses the [PublishComponent table](#) to determine the components that are set as advertised. To publish on a per-feature basis, invoke the [PublishFeatures action](#). This action uses the [FeatureComponents table](#) as data to resolve which features are advertised.

There are also two corresponding actions that unpublish a feature or a component: the [UnpublishComponents action](#) and the [UnpublishFeatures action](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# File Searching

During installation, there is often the need to search for a specific file. The installer has four standard search actions to serve this purpose. The [AppSearch action](#) uses the [AppSearch table](#) to determine if a file exists on a user's system. During an upgrade installation, the [CCPSearch action](#) and the [RMCCPSearch action](#) can use their corresponding tables to check the user's system for a previous version of a product.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# File Costing

Costing is the process of determining the total disk space requirements for an installation. The elements calculated in the file costing process include the amount of disk space in which files are installed or removed, as well as the amount of disk space taken up by registry entries, shortcuts, and other miscellaneous files. Existing files scheduled to be overwritten are also calculated in the disk cost totals.

Total costs are accumulated on a per-[component](#) basis and consist of three separate parts: local costs, source costs, and removal costs. These parts correspond to the disk cost that is incurred if the component is installed locally, installed to run from the source media, or removed.

All calculations involving the cost of installing files depend upon the disk volume to which the file is to be installed or removed. Each time the directory associated with a component changes, the costs of the installation files controlled by that component must be recalculated. For example, because a directory change might also imply a volume change, the clustered file sizes must be recalculated. In addition, the new directory must be checked to determine whether any existing files that may be overwritten must be taken into account.

After the [CostInitialize](#) action is called, the [FileCost](#) action must be called. The [CostInitialize](#) action initializes the installer's internal routines that dynamically calculate the disk costs involved with the standard installation actions. No other dynamic cost calculations are done at this point.

Next, the [CostFinalize](#) action must be called. This action finalizes all cost calculations and makes the costing data available through the [Component](#) table.

After the [CostFinalize](#) action completes execution, the [Component](#) table is fully initialized and a user interface dialog box sequence containing a [SelectionTree](#) control can be initiated if needed. The user interface dialog boxes may offer the option to change the selection state or destination directory of any feature in the [Feature](#) table to the user. The process is similar when the selection state of a component changes; however, in this case, the dynamic cost of the changed component only is recalculated.

Once the user has completed selecting features in the user interface, the [InstallValidate](#) action should be called. This action verifies that all volumes to which cost has been attributed have sufficient space for the installation.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# File Installation

Once [file costing](#) has been completed, the installer has all the information required to process the two file manipulation actions: [DuplicateFiles](#) and [MoveFiles](#).

Use the [DuplicateFiles action](#) to specify the files the installer is to duplicate in the [DuplicateFile table](#). Use the [MoveFiles action](#) to determine which files to move by querying the [MoveFile table](#).

Note that the Options field of the [MoveFile table](#) specifies whether or not the original file is to be deleted from its current location. Files that are moved or copied by the MoveFiles action are not deleted when the product is uninstalled.

The [InstallFiles action](#) is called once all other file manipulation operations have completed. The InstallFiles action processes the [Media table](#), the [File table](#), and the [Component table](#) to determine which files will be copied from the source to the destination.

The [InstallFiles action](#) creates the directory structure necessary to install all files. If an explicitly empty folder is required to be installed, the [CreateFolders action](#) may be called to create it.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Modifying the Registry

Registry keys can be written to the system registry once all selected components and their related files have been installed. The standard actions related to modifying the registry must be sequenced after the file installation standard actions because registry keys cannot be written unless the corresponding component and file have been successfully installed.

The [RegisterClassInfo action](#) accesses the [Class table](#) to register the COM class information of the installed components.

The [RegisterExtensionInfo action](#) queries the [Extension table](#) and [Verb table](#) and registers the corresponding extensions and command-verb information with the operating system.

The [RegisterProgIdInfo action](#) manages the registration of OLE ProgId information with the operating system.

The [RegisterMIMEInfo action](#) processes the [MIME table](#) to register the association between a MIME context type, the file name extension, and the CLSID.

The [WriteRegistryValues action](#) processes the [Registry table](#) and writes the keys for all components that have been either installed locally or to run from source. The Registry table allows keys to be written to the **HKEY\_CLASSES\_ROOT**, **HKEY\_CURRENT\_USER**, **HKEY\_LOCAL\_MACHINE**, and **HKEY\_USERS** registry hives.

The [RemoveRegistryValues action](#) removes the keys that have been marked to be deleted in the Name column of the [Registry table](#) or the [RemoveRegistry Table](#).

The [RegisterTypeLibraries action](#) processes the [TypeLib table](#) and registers the installed type libraries with the system.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Running Actions

The installer functions can be used to run specific actions or action sequences. These actions can either be [standard](#) or [custom](#) actions. The following procedure describes how to run actions:

## ▶ To run an action sequence

1. Run a sequence of actions defined in a table by calling the [MsiSequence](#) function.  
The installer queries the indicated table and runs each action if its conditional expression evaluates to TRUE.
2. Check conditional expressions by calling the [MsiEvaluateCondition](#) function.
3. Run the action by calling the [MsiDoAction](#) function. The action can be a standard action, a custom action, or a user interface dialog box.
4. If an error occurred during the execution of this action, call the [MsiProcessMessage](#) function. The installer will process the error.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Standard Actions Reference

The Windows Installer has the following standard actions.

Action name	Brief description of action
ADMIN	A top-level action used for an administrative installation.
ADVERTISE	A top-level action called to install or remove advertised components.
AllocateRegistrySpace	Validates that the free space specified by <b>AVAILABLEFREEREG</b> exists in the registry.
AppSearch	Searches for previous versions of products and determines that upgrades are installed.
BindImage	Binds executables to imported DLLs.
CCPSearch	Uses file signatures to validate that qualifying products are installed on a system before an upgrade installation is performed.
CostFinalize	Ends the internal installation costing process begun by the <b>CostInitialize</b> action.
CostInitialize	Starts the installation costing process.
CreateFolders	Creates empty folders for components.
CreateShortcuts	Creates shortcuts.
DeleteServices	Removes system services.
DisableRollback	Disables rollback for the remainder of the installation.
DuplicateFiles	Duplicates files installed by the InstallFiles action.
ExecuteAction	Checks the <b>EXECUTEACTION</b> property to determine which top-level action begins the execution sequence, then runs that action.

<a href="#">FileCost</a>	Initializes disk cost calculation with the installer. Disk costing is not finalized until the CostFinalize action is executed.
<a href="#">FindRelatedProducts</a>	Detects correspondence between the <a href="#">Upgrade table</a> and installed products.
<a href="#">ForceReboot</a>	Used in the action sequence to prompt the user for a restart of the system during the installation.
<a href="#">INSTALL</a>	A top-level action called to install or remove components.
<a href="#">InstallAdminPackage</a>	Copies the installer database to the administrative installation point.
<a href="#">InstallExecute</a>	Runs a script containing all operations in the action sequence since either the start of the installation or the last InstallFinalize action. Does not end the transaction.
<a href="#">InstallFiles</a>	Copies files from the source to the destination directory.
<a href="#">InstallFinalize</a>	Runs a script containing all operations in the action sequence since either the start of the installation or the last InstallFinalize action. Marks the end of a transaction.
<a href="#">InstallInitialize</a>	Marks the beginning of a transaction.
<a href="#">InstallSFPCatalogFile</a>	The InstallSFPCatalogFile action installs the catalogs used by Windows Me for Windows File Protection.
<a href="#">InstallValidate</a>	Verifies that all volumes with attributed costs have sufficient space for the installation.
<a href="#">IsolateComponents</a>	Processes the <a href="#">IsolatedComponent table</a>
<a href="#">LaunchConditions</a>	Evaluates a set of conditional statements contained in the LaunchCondition table that must all evaluate to True before the installation can proceed.

MigrateFeatureStates	Migrates current feature states to the pending installation.
MoveFiles	Locates existing files and moves or copies those files to a new location.
MsiConfigureServices	Configures a service for the system.  <b>Windows Installer 4.5 and earlier:</b> Not supported.
MsiPublishAssemblies action	Manages the advertisement of common language runtime assemblies and Win32 assemblies that are being installed.
MsiUnpublishAssemblies	Manages the advertisement of common language runtime assemblies and Win32 assemblies that are being removed.
InstallODBC	Installs the ODBC drivers, translators, and data sources.
InstallServices	Registers a service with the system.
PatchFiles	Queries the Patch table to determine which patches are applied to specific files and then performs the byte-wise patching of the files.
ProcessComponents	Registers components, their key paths, and component clients.
PublishComponents	Advertises the components specified in the PublishComponent table.
PublishFeatures	Writes the feature state of each feature into the system registry
PublishProduct	Publishes product information with the system.
RegisterClassInfo	Manages the registration of COM class information with the system.
RegisterComPlus	The RegisterComPlus action registers COM+ applications.

RegisterExtensionInfo	Registers extension related information with the system.
RegisterFonts	Registers installed fonts with the system.
RegisterMIMEInfo	Registers MIME information with the system.
RegisterProduct	Registers product information with the installer and stores the installer database on the local computer.
RegisterProgIdInfo	Registers OLE ProgId information with the system.
RegisterTypeLibraries	Registers type libraries with the system.
RegisterUser	Registers user information to identify the user of a product.
RemoveDuplicateFiles	Deletes files installed by the DuplicateFiles action.
RemoveEnvironmentStrings	Modifies the values of environment variables.
RemoveExistingProducts	Removes installed versions of a product.
RemoveFiles	Removes files previously installed by the InstallFiles action.
RemoveFolders	Removes empty folders linked to components set to be removed.
RemoveIniValues	Deletes .ini file information associated with a component specified in the IniFile table.
RemoveODBC	Removes ODBC data sources, translators, and drivers.
RemoveRegistryValues	Removes an application's registry keys that were created from the Registry table..
RemoveShortcuts	Manages the removal of an advertised shortcut whose feature is selected for uninstallation.
ResolveSource	Determines the source location and sets the <b>SourceDir</b> property.

RMCCPSearch	Uses file signatures to validate that qualifying products are installed on a system before an upgrade installation is performed.
ScheduleReboot	Prompts the user for a system restart at the end of the installation.
SelfRegModules	Processes modules in the SelfReg table and registers them if they are installed.
SelfUnregModules	Unregisters the modules in the SelfReg table that are set to be uninstalled.
SEQUENCE	Runs the actions in a table specified by the <b>SEQUENCE</b> property.
SetODBCFolders Action	Checks the system for existing ODBC drivers and sets target directory for new ODBC drivers.
StartServices	Starts system services.
StopServices	Stops system services.
UnpublishComponents	Manages the unadvertisement of components from the PublishComponent table and removes information about published components.
UnpublishFeatures	Removes the selection-state and feature-component mapping information from the system registry.
UnregisterClassInfo	Manages the removal of COM classes from the system registry.
UnregisterComPlus	The UnregisterComPlus action removes COM+ applications from the registry.
UnregisterExtensionInfo	Manages the removal of extension-related information from the system.
UnregisterFonts	Removes registration information about installed fonts from the system.
UnregisterMIMEInfo	Unregisters MIME-related information from the system registry.

<a href="#">UnregisterProgIdInfo</a>	Manages the unregistration of OLE ProgId information with the system.
<a href="#">UnregisterTypeLibraries</a>	Unregisters type libraries with the system.
<a href="#">ValidateProductID</a>	Sets <b>ProductID</b> property to the full product identifier.
<a href="#">WriteEnvironmentStrings</a>	Modifies the values of environment variables.
<a href="#">WriteIniValues</a>	Writes .ini file information.
<a href="#">WriteRegistryValues</a>	Sets up registry information.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ADMIN Action

The ADMIN action is a top-level action used to perform [administrative installations](#).

## Sequence Restrictions

There are no sequence restrictions.

## ActionData Messages

There are no ActionData messages.

## Remarks

The ADMIN action is not called from within the action table sequence, Windows Installer executes this action when [MsiInstallProduct](#) is called with the *szCommandLine* parameter set to "ACTION=ADMIN" or the command line executable Msiexec.exe is called with the '/a' command line switch.

## See Also

[AdminUISequence Table](#)

[AdminExecuteSequence Table](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ADVERTISE Action

The ADVERTISE action is a top-level action called to install or remove advertised components.

## Sequence Restrictions

There are no sequence restrictions.

## ActionData Messages

There are no ActionData messages.

## Remarks

[Advertising](#) refers to the installer's ability to provide the loading and launching interfaces of an application without physically installing the application. The installer does not install the necessary components until a user or application activates an advertised interface. This concept is called *install-on-demand*.

The ADVERTISE action is not called from within the action table sequence, the Windows Installer executes this action when the command line executable Msiexec.exe is called with the '/j' command line switch or when **MsiInstallProduct** is called with the *szCommandLine* parameter set to ACTION=ADVERTISE.

## See Also

[AdvExecuteSequence Table](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# AllocateRegistrySpace Action

The AllocateRegistrySpace action ensures that the amount of free registry space specified by the [AVAILABLEFREEREG](#) property exists in the registry.

## Sequence Restrictions

The AllocateRegistrySpace action must be called after [InstallInitialize](#). It is advisable to schedule the AllocateRegistrySpace before all other actions to ensure there is enough registry space available to continue.

## ActionData Messages

Field	Description of action data
[1]	Registry space required in KB.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# AppSearch Action

The AppSearch action uses file signatures to search for existing versions of products. The AppSearch action may use this information to determine where upgrades are to be installed. The AppSearch action can also be used to set a property to the existing value of an registry or .ini file entry.

## Sequence Restrictions

AppSearch should be authored into the [InstallUISequence table](#) and [InstallExecuteSequence table](#). The installer prevents the AppSearch action from running in the InstallExecuteSequence sequence if the action has already run in InstallUISequence sequence.

## ActionData Messages

Field	Description of action data
[1]	Property holding file location.
[2]	File signature.

## Remarks

The AppSearch action requires that the [Signature](#) table be present in the installation package. File signatures are listed in the Signature table. A signature that is not in the Signature table denotes a directory and the action sets the property to the directory path for that signature.

The AppSearch action searches for file signatures using the [CompLocator](#) table first, the [RegLocator](#) table next, then the [IniLocator](#) table, and finally the [DrLocator](#) table.

## See Also

[AppSearch](#)  
[CompLocator](#)

[IniLocator](#)  
[RegLocator](#)  
[DrLocator](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# BindImage Action

The BindImage action binds each executable or DLL that must be bound to the DLLs imported by it. The BindImage action acts on each file in the [BindImage](#) table, but only those that are to be installed locally. The installer computes the virtual address of each function imported from all DLLs, then saves the computed virtual address in the importing image's *import address table* (IAT).

The BindImage Action internally calls the **BindImageEx** Windows API.

## Sequence Restrictions

The BindImage action must come after the [InstallFiles](#) action.

## ActionData Messages

Field	Description of action data
[1]	File identifier of executable.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# CCPSearch Action

The CCPSearch action uses file signatures to validate that qualifying products are installed on a system before an upgrade installation is performed.

## Sequence Restrictions

CCPSearch action should be authored into the [InstallUISequence table](#) and [InstallExecuteSequence table](#). The installer prevents the CCPSearch action from running in the InstallExecuteSequence sequence if the action has already run in InstallUISequence sequence. The CCPSearch action must come before the [RMCCPSearch](#) action.

## ActionData Messages

There are no ActionData messages.

## Remarks

The CCPSearch action searches for file signatures listed in the CCPSearch table on the system using the following tables in order: Signature, CompLocator, RegLocator, IniLocator, and DrLocator.

If any signature is determined to exist, the **CCP\_Success** property is set to 1 and the CCPSearch action terminates.

The absence of the signature from the Signature table denotes a directory.

## See Also

[CCPSearch](#)  
[Signature](#)  
[CompLocator](#)  
[RegLocator](#)  
[IniLocator](#)  
[DrLocator](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# CostFinalize Action

The CostFinalize action ends the internal installation *costing* process begun by the [CostInitialize](#) action.

## Sequence Restrictions

Any standard or [custom actions](#) that affect costing should be sequenced before the CostInitialize action. Call the [FileCost](#) action immediately following the CostInitialize action and then call the CostFinalize action to make all final cost calculations available to the installer through the [Component](#) table.

The CostFinalize action must be executed before starting any user interface sequence which allows the user to view or modify [Feature](#) table selections or directories.

## ActionData Messages

There are no ActionData messages.

## Remarks

The CostFinalize action queries the [Condition](#) table to determine which features are scheduled to be installed. Costing is done for each component in the Component table.

The CostFinalize action also verifies that all the target directories are writable before allowing the installation to continue.

**Note** During an [administrative installation](#), CostFinalize sets all features for installation except features having 0 authored in the Level column of the Feature table.

## See Also

[File Costing](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# CostInitialize Action

The CostInitialize action initiates the installation [costing](#) process.

## Sequence Restrictions

Any standard or [custom](#) actions that affect costing should be sequenced before the CostInitialize action. Call the [FileCost](#) action immediately following the CostInitialize action. Then call the [CostFinalize](#) action following the CostInitialize action to make all final cost calculations available to the installer through the [Component](#) table.

## ActionData Messages

There are no ActionData messages.

## Remarks

The CostInitialize action loads the Component table and [Feature](#) table into memory.

For more information on the costing process in the installer, see [File Costing](#).

## See Also

[File Costing](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# CreateFolders Action

The CreateFolders action creates empty folders for components that are set to be installed. The installer creates folders both for components that run locally and run from source. Newly created folders are registered with the appropriate component identifier.

The CreateFolders action queries the [CreateFolder table](#) and the [Component table](#).

## Sequence Restrictions

The CreateFolders action must be executed either before the [InstallFiles](#) action or before any action that adds files to folders.

## ActionData Messages

Field	Description of action data description
[1]	Directory identifier of folder.

## Remarks

The installer does not automatically remove folders created by the CreateFolders action during an uninstallation of the application. The installer only removes folders if the [RemoveFolders action](#) is included in the action sequence.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# CreateShortcuts Action

The CreateShortcuts action manages the creation of shortcuts.

## Sequence Restrictions

The CreateShortcuts action must come after the [InstallFiles](#) action and the [RemoveShortcuts](#) action.

## ActionData Messages

Field	Description of action data
[1]	Identifier of created shortcut.

## Remarks

The CreateShortcuts action creates shortcuts to the key files of components of features that are selected for installation or advertisement.

## See Also

[Shortcut Table](#)

[MsiShortcutProperty Table](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DeleteServices Action

The DeleteServices action stops a service and removes its registration from the system. This action queries the [ServiceControl table](#).

## Sequence Restrictions

The services actions must be used in the following sequence:

1. [StopServices](#)
2. DeleteServices
3. Any of the following actions: [InstallFiles](#), [RemoveFiles](#), [DuplicateFiles](#), [MoveFiles](#), [PatchFiles](#), and [RemoveDuplicateFiles](#) actions.
4. [InstallServices action](#)
5. [StartServices](#)

## ActionData Messages

Field	Description of action data
[1]	Service display name.
[2]	Service name.

## Remarks

This action requires the user be an administrator or have elevated privileges with permission to delete services or that the application be part of a managed installation.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DisableRollback Action

The DisableRollback Action disables [rollback](#) for the remainder of the installation. Rollback is disabled only for the actions that are sequenced after the DisableRollback action in the sequence tables. Rollback is disabled for the entire installation if the DisableRollback action is sequenced before the [InstallInitialize action](#).

## Sequence Restrictions

There are no sequence restrictions.

## ActionData Message

There are no ActionData messages.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DuplicateFiles Action

The DuplicateFiles action duplicates files installed by the [InstallFiles](#) action. The duplicate files can be copied to the same directory with a different name or to a different directory with the original name.

## Sequence Restrictions

The DuplicateFiles action must come after the [InstallFiles](#) action. The DuplicateFiles Action must also come after the [PatchFiles](#) action to prevent duplicating the unpatched version of the file.

## ActionData Messages

Field	Description of action data
[1]	Identifier of duplicated file.
[6]	Size of duplicated file.
[9]	Identifier of directory holding duplicated file.

## Remarks

The DuplicateFiles action processes a [DuplicateFile](#) table entry only if the component linked to that entry is being installed locally. For more information, see [Component](#) table.

The string in the DestFolder field is a property name whose value is expected to resolve to a fully qualified path. This property can either be any of the directory entries in the [Directory](#) table, any pre-defined folder property ([CommonFilesFolder](#), for example), or a property set by any entry in the [AppSearch](#) table. If the [DestFolder](#) property does not evaluate to a valid path the DuplicateFiles action does nothing for that entry.

If the name of the destination file in the DestName column of the DuplicateFile table is left blank, the destination file name will be the same

as the original file name.

Files installed by the DuplicateFiles action are removed by the [RemoveDuplicateFiles](#) action when appropriate.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ExecuteAction Action

The ExecuteAction action initiates the execution sequence using the [EXECUTEACTION](#) property to determine which type of installation to perform.

## Sequence Restrictions

This action should be sequenced after all information collection necessary to begin the installation is complete. Additional actions may be sequenced after ExecuteAction action in the [InstallUISequence table](#), and [AdminUISequence table](#). A sequence will typically begin with *costing* actions, such as the [CostInitialize action](#), followed by the user interface actions, and then the ExecuteAction action.

## ActionData Messages

There are no ActionData messages.

## Remarks

The ExecuteAction action is run with system privileges if the installer service is enabled. The top-level actions, such as the [INSTALL action](#), [ADVERTISE action](#), and [ADMIN action](#) include internal logic that determines whether calling the ExecuteAction action requires either the execution sequence or the user interface sequence to run.

## See Also

[InstallUISequence table](#)  
[AdminUISequence table](#)  
[CostInitialize action](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# FileCost Action

The FileCostaction initiates dynamic *costing* of standard installation actions.

## ActionData Messages

There are no ActionData messages.

## Sequence Restrictions

Any standard or [custom actions](#) that affect costing should be sequenced before the [CostInitialize](#) action. Call the FileCost action immediately following the CostInitialize action. Then call the [CostFinalize](#) action following the CostInitialize action to make all final cost calculations available to the installer through the [Component](#) table.

The CostInitialize action must be executed before the FileCost action. The installer then determines the disk-space cost of every file in the [File](#) table, on a per-component basis (See Component Table), taking both [volume](#) clustering and the presence of existing files that may need to be overwritten into account. All actions that consume or release disk space are also considered. If an existing file is found, a file version check is performed to determine whether the new file actually needs to be installed or not. If the existing file is of an equal or greater version number, the existing file is not overwritten and no disk-space cost is incurred. In all cases, the installer uses the results of version number checking to set the installation state of each file.

The FileCost action initializes cost calculation with the installer. Actual dynamic costing does not occur until the CostFinalize action is executed.

## See Also

[File Costing](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# FindRelatedProducts Action

The FindRelatedProducts action runs through each record of the [Upgrade table](#) in sequence and compares the upgrade code, product version, and language in each row to products installed on the system. When FindRelatedProducts detects a correspondence between the upgrade information and an installed product, it appends the product code to the property specified in the ActionProperty column of the UpgradeTable.

The FindRelatedProducts action only runs the first time the product is installed. The FindRelatedProducts action does not run during maintenance mode or uninstallation.

## Database Tables Queried

This action queries the following table:

[Upgrade Table](#)

## Properties Used

The FindRelatedProducts action uses the [UpgradeCode](#) property and the version and language information authored into the Upgrade table to detect installed products affected by the pending upgrade. It appends the product code of detected products to the property in the ActionProperty column of the UpgradeTable.

FindRelatedProducts only recognizes existing products that have been installed using the Windows Installer with an .msi that defines an [UpgradeCode](#) property, a [ProductVersion](#) property, and a value for the [ProductLanguage](#) property that is one of the languages listed in the [Template Summary](#) Property.

Note that FindRelatedProducts uses the language returned by [MsiGetProductInfo](#). For FindRelatedProducts to work correctly, the package author must be sure that the [ProductLanguage](#) property in the [Property table](#) is set to a language that is also listed in the [Template Summary](#) Property. See [Preparing an Application for Future Major Upgrades](#).

## Sequence Restrictions

FindRelatedProducts should be authored into the [InstallUISequence table](#) and [InstallExecuteSequence](#) tables. The installer prevents FindRelated Products from running in InstallExecuteSequence if the action has already run in InstallUISequence. The FindRelatedProducts action must come before the [MigrateFeatureStates action](#) and the [RemoveExistingProducts action](#).

## ActionData Messages

FindRelatedProducts sends an action data message for each related product it detects on the system.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ForceReboot Action

The ForceReboot action prompts the user for a restart of the system during the installation. The ForceReboot action is different from the [ScheduleReboot action](#) in that the ScheduleReboot action is used to schedule a prompt to restart at the end of the installation.

If the installation has a user interface, the installer displays a dialog box at each ForceReboot action which prompts the user to restart the system. The user must respond to this prompt before continuing with the installation. If the installation has no user interface, the system automatically restarts at the ForceReboot action.

If the installer determines that a restart is necessary, it automatically prompts the user to restart at the end of the installation, whether or not there are any ForceReboot or ScheduleReboot actions in the sequence. For example, the installer automatically prompts for a restart if it needs to replace any files used during the installation.

Suppress certain restart prompts by setting the **REBOOT** property.

If the Windows Installer encounters the ForceReboot or ScheduleReboot action during a [multiple-package installation](#), the installer will stop and roll back the installation. No package of the multiple-package installation will be installed.

## Sequence Restrictions

The following actions commonly occur together as a group in the action sequence. It is recommended that the ForceReboot action be scheduled to come after this group. If the ForceReboot action is scheduled before the [RegisterProduct action](#), the installer again requires the source of the installation package after the restart. Therefore, the preferred sequence for ForceReboot is immediately following this action sequence.

- [RegisterProduct](#)
- [RegisterUser](#)
- [PublishProduct](#)
- [PublishFeatures](#)

- [CreateShortcuts](#)
- [RegisterMIMEInfo](#)
- [RegisterExtensionInfo](#)
- [RegisterClassInfo](#)
- [RegisterProgIdInfo](#)

The ForceReboot action must come between [InstallInitialize](#) and [InstallFinalize](#) in the action sequence of the [InstallExecuteSequence](#) table.

## ActionData Messages

There are no ActionData messages.

## Remarks

The ForceReboot action must always be used with a conditional statement such that the installer triggers a restart only when necessary. For example, a restart may only be required if a particular file is replaced or a particular component is installed. Each product installation is unique and a custom action may be required to determine whether a restart is needed. The condition on the ForceReboot action commonly makes use of the [AFTERREBOOT](#) property.

ForceReboot runs system operations generated by any previous actions before prompting for a restart or restarting. For example, the system operations generated by [InstallFiles](#) and [WriteRegistryValues](#) are run before a restart.

The ForceReboot action writes a registry key that causes the installer to start after restarting. The location of this key is **HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVe**

## See Also

[System Reboots](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# INSTALL Action

The INSTALL action is a top-level action called to install or remove components. This action queries the [InstallUISequence Table](#) and [InstallExecuteSequence Table](#) for the action to execute, the condition for action execution, and the place of the action in the sequence:

## Sequence Restrictions

There are no sequence restrictions.

## ActionData Messages

There are no ActionData messages.

## Remarks

The INSTALL action is not called from within the action table sequence, it is passed to Windows Installer when [MsiInstallProduct](#) is called, or the command line executable Msiexec.exe is called with the '/i' command line switch, or when any installer function is called that may perform an installation task, such as [MsiConfigureFeature](#), [MsiProvideComponent](#), or [MsiInstallMissingFile](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# InstallAdminPackage Action

The InstallAdminPackage action copies the product database to the administrative installation point, which is defined by the **TARGETDIR** property.

## Sequence Restrictions

There are no sequence restrictions.

## ActionData Messages

Field	Description of action data
[1]	Name of install files.
[6]	Size of database.
[9]	Identifier of administrative installation–point directory.

## Remarks

The InstallAdminPackage action also updates the summary information of the database and removes any cabinet files located in streams after the database is copied.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# InstallExecute Action

The InstallExecute action runs a script containing all operations in the action sequence since either the start of the installation or the last InstallExecute action or [InstallExecuteAgain action](#). The InstallExecute action updates the system without ending the transaction.

## Sequence Restrictions

The InstallExecute action comes between the [InstallInitialize action](#) and the [InstallFinalize action](#).

## ActionData Messages

There are no ActionData messages.

## Remarks

The [InstallExecuteAgain action](#) does the same thing as the InstallExecute action. Because the sequence tables have only one primary key, the Action column, the same action cannot be repeated in a particular sequence table. Two actions exist that do the same thing, InstallExecute and InstallExecuteAgain, for cases where the functionality of InstallExecute is needed twice in the [InstallExecuteSequence table](#). For more information, see [Using a Sequence Table](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# InstallExecuteAgain Action

The InstallExecuteAgain action runs a script containing all operations in the action sequence since either the start of the installation or the last InstallExecuteAgain action or the last [InstallExecute action](#). The InstallExecute action updates the system without ending the transaction. InstallExecuteAgain performs the same operation as the [InstallExecute action](#) but should only be used after InstallExecute.

InstallExecuteAgain should always be set so that it does not run during removal. For information, see [Using Properties in Conditional Statements](#).

## Sequence Restrictions

The InstallExecute action comes between the [InstallInitialize action](#) and the [InstallFinalize action](#).

## ActionData Messages

There are no ActionData messages.

## Remarks

The InstallExecuteAgain action does the same thing as the [InstallExecute action](#). Because the sequence tables have only one primary key, the Action column, the same action cannot be repeated in a particular sequence table. Two actions exist that do the same thing, InstallExecute and InstallExecuteAgain, for cases where the functionality of InstallExecute is needed twice in the [InstallExecuteSequence table](#). For more information, see [Using a Sequence Table](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# InstallFiles Action

The InstallFiles action copies files specified in the File table from the source directory to the destination directory.

## Sequence Restrictions

The InstallFiles action must come after the [InstallValidate](#) action and before any file-dependent actions.

## ActionData Messages

Field	Description of action data
[1]	Identifier of installed file.
[6]	Size of installed file in bytes.
[9]	Identifier of directory holding installed file.

## Remarks

The InstallFiles action operates on files specified in the [File table](#). Each file is installed based on the installation state of the file's associated component in the [Component table](#). Only those files whose components are resolved to the *msiInstallStateLocal* state are eligible for copying.

The InstallFiles action implements the following columns of the File table.

- The FileName column specifies the target file name.
- The Version column specifies the file version.
- The Attributes column specifies the file and installation attribute flag bits.
- The File column specifies the unique file token.
- The FileSize column specifies the uncompressed file size in bytes.
- The Language column specifies the file language identifier.

- The Sequence column specifies the sequence number on media.

The InstallFiles action implements the following columns of the Component table.

- The Directory\_ column specifies a reference to a [Directory table](#) item.
- The Component column specifies a unique name for the component item.

The specified file is copied only if one of the following is true:

- The file is not currently installed on the local computer.
- The file is on the local computer but has a lower version number than the file in the [File table](#).
- The file is on the local computer, but there is no associated version number.

The source directory for each file to be copied is determined by the sourceMode, which in turn depends on the value in the Cabinet column of the Media table. For a full discussion of the source mode, see the [Media table](#).

If the source directory for a file to be copied resides on removable media such as a floppy disk or CD-ROM, the InstallFiles action verifies that the proper source media is inserted before attempting to copy the file. The InstallFiles searches for media of the same removable type with a [volume](#) label that matches the value given in the VolumeLabel column of the Media table. If a matching mounted volume is found, the file copying process proceeds. If no match is found, a dialog box requests that the user to insert the proper media. In this case, the dialog box uses the media name found in the DiskPrompt column of the Media table as part of the prompt.

Caution must be exercised because the InstallFiles action can delete an original file and not replace it. This occurs when the InstallFiles action experiences an error while replacing an older file and the user chooses to ignore the error. The default behavior of installer is to delete an old file before ensuring the new file is copied correctly.

For the file versioning rules used by the installer, see [File Versioning Rules](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# InstallFinalize Action

The InstallFinalize action runs a script that contains all operations in the action sequence since either the start of the installation or the execution of the [InstallExecute](#) or [InstallExecuteAgain](#) actions. This action marks the end of a transaction that begins with the [InstallInitialize](#) action.

## Sequence Restrictions

The InstallInitialize action must come before the InstallFinalize action.

## ActionData Messages

There are no ActionData messages.

## Remarks

If it is detected that the product is marked for complete removal, operations are automatically added to the script to remove the **Add/Remove Programs** in the **Control Panel** information for the product, to unregister and unpublish the product, and to remove the cached local database from %WINDOWS%, if it exists.

System changes made before the action may not be restored after the InstallFinalize action is executed. The InstallFinalize action updates the system with all operations determined to that point and then ends the transaction.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# InstallInitialize Action

The InstallInitialize action and [InstallFinalize](#) action mark the beginning and end of a sequence of actions that change the system.

## Sequence Restrictions

The InstallInitialize action must be sequenced before any actions that change the system, such as the [InstallFiles](#) action, the [WriteRegistryValues](#) action, the [SelfRegModules](#) action, and the [ProcessComponents](#) action. The InstallInitialize action must therefore be sequenced before the [InstallFinalize](#) action and the [InstallExecute](#) action.

[Custom actions](#) that modify the Windows Installer package, for example by adding rows to a table that handles installable resources, such as the [Registry](#) table or [DuplicateFile](#) table, must be sequenced before InstallInitialize action.

## ActionData Messages

There are no ActionData messages.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# InstallODBC Action

The InstallODBC action installs the drivers, translators, and data sources in the [ODBCDriver Table](#), [ODBCTranslator Table](#), and [ODBCDataSource Table](#). If a driver or translator already exists, the InstallODBC action makes SQL calls necessary for the installation.

## Sequence Restrictions

The InstallODBC action does not copy or remove files, and must be after actions that copy or remove files.

## ActionData Messages

The following table identifies the ActionData messages for each installed driver.

Field	Description
[1]	Driver description. The ODBC driver key.
[2]	ComponentId.
[3]	Folder.
[4, 5, ...]	Attribute and value pairs from <a href="#">ODBCAttribute</a> .

The following table identifies the ActionData messages for each installed translator.

Field	Description
[1]	Driver description. The ODBC driver key.
[2]	ComponentId.
[3]	Folder.
[4, 5, ...]	Attribute and value pairs from <a href="#">ODBCAttribute</a> .

The following table identifies the ActionData messages for each installed data source.

Field	Description
[1]	Driver description. The ODBC driver key.
[2]	ComponentId.
[3]	Registration: ODBC_ADD_DSN or ODBC_ADD_SYS_DSN.
[4, 5, ...]	Attribute and value pairs from <a href="#">ODBCAttribute</a> .

## Remarks

The ODBC Driver Manager must be authored in the Microsoft Installer package and a component named ODBCDriverManager must be included. The manager is installed if necessary.

To rename the component, set a property named ODBCDriverManager to the new name of the component. If a 64-bit ODBC Driver Manager is to be installed, the component that carries it should be named ODBCDriverManager64.

- The InstallODBC action queries the [ODBCDataSource Table](#) and the [ODBCSourceAttribute Table](#) for each data source to be installed, and the attributes of the data source.
- The InstallODBC action queries the [ODBCDriver Table](#) and the [ODBCTranslator Table](#) for each driver and translator that is selected for installation.
- The InstallODBC action queries the [ODBCAttribute Table](#) for the attributes of the drivers and translators.

## See Also

[Windows Installer Examples](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# InstallServices Action

The InstallServices action registers a service for the system. This action queries the [ServiceInstall table](#).

## Sequence Restrictions

The services action must be used in the following sequence.

[StopServices](#)

[DeleteServices](#)

Any of the following actions: [InstallFiles](#), [RemoveFiles](#), [DuplicateFiles](#), [MoveFiles](#), [PatchFiles](#), and [RemoveDuplicateFiles](#) actions.

InstallServices

[StartServices](#)

## ActionData Messages

There are no ActionData messages.

## Remarks

This action requires that the user be an administrator or have elevated privileges with permission to install services or that the application be part of a managed installation.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# InstallSFPCatalogFile Action

The InstallSFPCatalogFile action installs the catalogs used by Windows Me for Windows File Protection. InstallSFPCatalogFile queries the [Component table](#), [File table](#), [FileSFPCatalog table](#) and [SFPCatalog table](#). Catalogs are installed if they are associated with a file in a component that is set for local installation or if they are associated with a file being repaired in a locally installed component.

## Sequence Restrictions

The InstallSFPCatalogFile action must be sequenced before [InstallFiles](#) and after [CostFinalize](#).

## ActionData Messages

Field	Description of action data
[1]	Name of the catalog being installed.
[2]	Name of the catalog this catalog's installation depends upon.

## Remarks

A catalog that is dependent on another catalog installed after the parent catalog.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# InstallValidate Action

The InstallValidate action verifies that all [volumes](#) to which [cost](#) has been attributed have sufficient space for the installation. The InstallValidate action ends the installation with a fatal error if any volume is short of disk space.

The InstallValidate action also notifies the user if one or more files to be overwritten or removed are currently in use by an active process. For more information, see [System Reboots](#).

## Sequence Restrictions

The [CostFinalize](#) action and any UI dialog box sequences that allow the user to modify selection states and/or directories should be sequenced before the InstallValidate action.

[Custom actions](#) that change the installation state of features or components must be sequenced before the InstallValidate action.

## ActionData Messages

There are no ActionData messages.

## Remarks

Typically, an earlier UI dialog box sequence should perform the same verification as the InstallValidate action when the user attempts to initiate the copying of files. This UI dialog box sequence should present an **Out of Disk Space** dialog box if the volumes selected do not have enough space for the installation. The UI dialog boxes should be authored in a way to prevent the user from proceeding with the installation if there is insufficient disk space. In the case of a quiet installation, there is no user interface and the InstallValidate action terminates the installation if there is insufficient disk space. The cause of the premature termination is recorded in the log file if logging is enabled.

An entry is added to an internal FilesInUse table if any file is overwritten or removed while it is open for execution or modification by any process

during file costing. The FilesInUse table contains columns for the name and full path of the file. When the InstallValidate action executes, the installer queries the FilesInUse table for entries and determines the name of the process using the file. The InstallValidate action adds one record to the [ListBox](#) user interface table for each unique process identified by this query. The record contains the following values in each column:

**Property:** FileInUseProcess

**Value:** *Name of process*

**Text:** *Text contained in the caption of the main window of the process*

The InstallValidate action then displays the **Files In Use** dialog box. This dialog box displays the processes that must be shut down to avoid the requirement of restarting the system to replace files in use.

The InstallValidate action queries the [Dialog](#) table for an authored dialog box with the reserved name [FilesInUse](#) dialog and displays it. This dialog box must contain a [ListBox](#) control that is tied to a property named FileInUseProcess. By convention, this dialog box has an **Exit**, **Retry**, or **Ignore** button, but this is up to the UI author. Each button should be tied to an [EndDialog](#) ControlEvent in the [ControlEvent](#) table. The InstallValidate action responds as follows to the value returned by the [DoAction](#) ControlEvent, as dictated by one of these [EndDialog](#) arguments associated with the button pushed by the user:

**Retry:** All values added to the [ListBox](#) table are cleared, and the entire file costing procedure is repeated, rechecking for files that are still in use. If one or more processes are still identified as using files to be overwritten or deleted, the process repeats; otherwise, InstallValidate returns control to the installer with a status of `msiDoActionStatusSuccess`.

**Exit:** The InstallValidate action immediately returns control to the installer with a status of `msiDoActionStatusUserExit`. This terminates the installation.

**Any other return value:** The InstallValidate action immediately returns control to the installer with a status of `msiDoActionStatusSuccess`. In this case, since one or more files are still in use, the subsequent [InstallFiles](#) and/or [InstallAdminPackage](#) actions must schedule the in-use file(s) to be replaced or deleted when the system is restarted.

If there is no [ListBox](#) table in the database, InstallValidate exits silently without an error.

The semicolon is the list delimiter for transforms, sources, and patches, and should not be used in these file names or paths.

Files marked read-only in a read-only location are never considered in use by the installer.

A default **Out of Disk Space** dialog box containing **Abort** and **Retry** buttons is presented to the user if the user interface level is basic.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IsolateComponents Action

The IsolateComponents action installs a copy of a component (commonly a shared DLL) into a private location for use by a specific application (typically an .exe). This isolates the application from other copies of the component that may be installed to a shared location on the computer. For more information, see [Isolated Components](#).

The action refers to each record of the [IsolatedComponent table](#) and associates the files of the component listed in the Component\_Shared field with the component listed in the Component\_Application field. The installer installs the files of Component\_Shared into the same directory as Component\_Application. The installer generates a file in this directory, zero bytes in length, having the short filename name of the key file for Component\_Application (typically this is the same file name as the .exe) appended with .local. The IsolatedComponent action does not affect the installation of Component\_Application. Uninstalling Component\_Application also removes the Component\_Shared files and the .local file from the directory.

## Sequence Restrictions

The IsolateComponents action can be used only in the [InstallUISequence table](#) and the [InstallExecuteSequence table](#). This action must come after the [CostInitialize action](#) and before the [CostFinalize action](#).

## ActionData Messages

There are no ActionData messages.

## Remarks

If the Condition column for the IsolateComponents action evaluates to True, or is left blank, the installer isolates all the components listed in the [IsolatedComponent table](#). If the Condition column evaluates to False, the installer ignores the IsolatedComponent table and shares the components usual. The [RedirectedDllSupport](#) property may be used to condition this action. For more information, see [Using a Sequence Table](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# LaunchConditions Action

The LaunchConditions action queries the [LaunchCondition table](#) and evaluates each conditional statement recorded there. If any of these conditional statements fail, an error message is displayed to the user and the installation is terminated.

## Sequence Restrictions

The LaunchConditions action is optional. This action is normally the first in the sequence, but the [AppSearch Action](#) may be sequenced before the LaunchConditions action. If there are launch conditions that do not apply to all installation modes, the appropriate installation mode property should be used in a conditional expression in the appropriate sequence table.

## ActionData Messages

There are no ActionData messages.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MigrateFeatureStates Action

The MigrateFeatureStates action is used during upgrading and when installing a new application over a related application.

MigrateFeatureStates reads the feature states in the existing application and then sets these feature states in the pending installation. The method is only useful when the new feature tree has not greatly changed from the original.

The MigrateFeatureStates action only runs the first time the product is installed. The MigrateFeatureStates action does not run during maintenance mode or uninstallation.

MigrateFeatureStates action runs through each record of the [Upgrade table](#) in sequence and compares the upgrade code, product version, and language in each row to all products installed on the system. If MigrateFeatureStates action detects a correspondence, and if the msidbUpgradeAttributesMigrateFeatures bit flag is set in the Attributes column of the Upgrade table, the installer queries the existing feature states for the product and sets these states for the same features in the new application. The action only migrates the feature states if the [Preselected](#) property is not set.

## Sequence Restrictions

The MigrateFeatureStates action should come immediately after the [CostFinalize action](#). MigrateFeatureStates must be sequenced in both the [InstallUISequence table](#) and the [InstallExecuteSequence table](#). The installer prevents MigrateFeatureStates from running in InstallExecuteSequence if the action has already run in InstallUISequence.

## ActionData Messages

MigrateFeatureSettings sends an action data message for each product.

## Remarks

If more than one installed product shares a feature, the installation state

for that feature may differ between products. MigrateFeatureState action uses the following order of precedence when migrating feature installation states: run local, run from source, advertised, and uninstalled. For example, installed product A may have feature Y as INSTALLSTATE\_LOCAL and installed product B may have feature Y as INSTALLSTATE\_ABSENT. If an upgrade installs product C and migrates the installation state of feature Y, MigrateFeatureState sets the installation state of feature Y in product C to INSTALLSTATE\_LOCAL.

For more information about how to use the MigrateFeatureStates action for product upgrades, see [Preparing an Application for Future Major Upgrades](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MoveFiles Action

The MoveFiles action locates existing files on the user's computer and moves or copies those files to a new location. The MoveFiles action queries the [MoveFile table](#) and moves files specified there if the component linked to the entries is specified to be install locally or is being run from source.

## Sequence Restrictions

The MoveFiles action must come after the [InstallValidate](#) action and before the [InstallFiles](#) action.

## ActionData Messages

Field	Description of action data
[1]	Identifier of moved file.
[6]	Size of installed file in bytes.
[9]	Identifier of directory holding moved file.

## Remarks

The MoveFiles table contain a column named "options" which specifies the source files to be moved or copied. A moved source file is deleted after it has been copied to a new location. For the exact syntax, see the [MoveFile table](#).

The SourceFolder and DestFolder columns of the MoveFile table are property names whose values are expected to resolve to fully qualified paths. These properties can be any of the directory entries in the [Directory](#) table, any predefined folder property ([FavoritesFolder](#), for example), or a property set by any entry in the [AppSearch](#) table. These properties may contain a full path containing the file name to a specific file. For example, the AppSearch table can be authored to search for a particular file and set a property to the full path to that file. In this

example, the SourceName column in the MoveFile table can be left blank to indicate that the value in the SourceFolder property contains a full file path. The semicolon is the list delimiter for transforms, sources, and patches and should not be used in file names or paths.

The MoveFiles action does not act on entries in the MoveFile table in which either the SourceFolder or DestFolder property does not evaluate to a full path.

The MoveFiles action attempts to move or copy all files in the source directory that match the name given in the SourceName column of the MoveFiles table. The name in the SourceName column can include either the \* or ? wildcards which allow a group of files to be moved or copied. For example, the SourceName column may contain an entry of "\*.xls" and the MoveFiles action moves or copies every Microsoft Excel workbook from the source directory to the destination.

The name to be given to the destination file can be specified in the DestName column of the MoveFile table. The destination file name retains the source file name if this column is left blank.

If a "\*" wildcard is entered in the SourceName column of the [MoveFile table](#) and a destination file name is specified in the DestName column, all moved or copied files retain the names in the sources.

Files that are moved or copied by the MoveFiles action are not deleted when the product is uninstalled.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiConfigureServices Action

The MsiConfigureServices action configures a service for the system. This action queries the [MsiServiceConfig](#) and the [MsiServiceConfigFailureActions](#) tables.

**Windows Installer 4.5 or earlier:** Not supported. This action is available beginning with Windows Installer 5.0.

## Sequence Restrictions

This standard action must be used in the following sequence.

[StopServices](#)

[DeleteServices](#)

Any of the following actions: [InstallFiles](#), [RemoveFiles](#), [DuplicateFiles](#), [MoveFiles](#), [PatchFiles](#), and [RemoveDuplicateFiles](#) actions.

[InstallServices](#)

MsiConfigureServices

[StartServices](#)

## ActionData Messages

There are no ActionData messages.

## Remarks

This action requires that the user be an administrator or have elevated privileges with permission to install services or that the application be part of a managed installation.

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiPublishAssemblies Action

The MsiPublishAssemblies action manages the advertisement of common language runtime assemblies and Win32 assemblies. The action queries the [MsiAssembly table](#) to determine which assemblies have features being advertised or installed to the global assembly cache and which assemblies have a parent component being advertised or installed to a location isolated for a particular application. For information see, [Installation of Assemblies to the Global Assembly Cache](#) and [Installation of Win32 Assemblies](#).

On Microsoft Windows XP and later systems, Windows Installer can install Win32 assemblies as [side-by-side assemblies](#). For more information, see [About Isolated Applications and Side-by-side Assemblies](#).

## Sequence Restrictions

The MsiPublishAssemblies action must come after the [InstallInitialize action](#) in the [InstallExecuteSequence table](#) or [AdvtExecuteSequence table](#).

## ActionData Messages

Field	Description of action data
[1]	Application context.
[2]	Assembly name.

## Remarks

For more information, see [Assemblies](#).

The [MsiUnpublishAssemblies Action](#) manages the advertisement of assemblies being removed.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiUnpublishAssemblies Action

The MsiUnpublishAssemblies action manages the advertisement of common language runtime assemblies and Win32 assemblies that are being removed. The action queries the [MsiAssembly table](#) to determine which assemblies have advertised or installed features being removed from the global assembly cache and which assemblies have an advertised or installed parent component being removed from a location isolated for a particular application. For information see, [Installation of Assemblies to the Global Assembly Cache](#) and [Installation of Win32 Assemblies](#).

On systems running Windows XP and later, Windows Installer can install Win32 assemblies as [side-by-side assemblies](#). For more information, see [About Isolated Applications and Side-by-side Assemblies](#).

## Sequence Restrictions

The MsiUnpublishAssemblies action must come after the [InstallInitialize action](#) in the [InstallExecuteSequence table](#).

## ActionData Messages

Field	Description of action data
[1]	Application context.
[2]	Assembly name.

## Remarks

The [MsiPublishAssemblies Action](#) manages the advertisement of assemblies being advertised or installed.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PatchFiles Action

The PatchFiles action queries the [Patch table](#) to determine which patches are to be applied. The PatchFiles action also performs the byte-wise patching of the files.

## Sequence Restrictions

If the file that is to be patched is not installed, the PatchFiles action must come after the [InstallFiles action](#) that installs the file. Previously installed files may be patched at any point in the action sequence. The [DuplicateFiles Action](#) must come after the PatchFiles action to prevent duplicating the unpatched version of the file.

## ActionData Messages

Field	Description of action data
[1]	Identifier of patched file.
[2]	Identifier of directory holding patched file.
[3]	Size of patch in bytes.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ProcessComponents Action

The ProcessComponents action registers and unregisters components, their key paths, and the component clients. The ProcessComponents action queries the KeyPath column of the [Component table](#) to determine keypaths. This registration is used by [MsiGetComponentPath](#) to return the path of a component for a product client.

## Sequence Restrictions

The ProcessComponents action must come after the [InstallInitialize](#) action.

## ActionData Messages

For each component being registered.

Field	Description of action data
[1]	ProductId
[2]	ComponentId
[3]	The key path for the component.

For each component being unregistered.

Field	Description of action data
[1]	ProductId
[2]	ComponentId

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PublishComponents Action

The PublishComponents action manages the advertisement of the components from the [PublishComponent](#) table.

## Sequence Restrictions

There are no sequence restrictions.

## ActionData Messages

Field	Description of action data
[1]	GUID representing the component ID of an advertised feature.
[2]	Qualifier of component ID.

## Remarks

The PublishComponents action publishes all of the components from the PublishComponents table that belong to features that have been selected for advertisement or installation.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PublishFeatures Action

The PublishFeatures action writes each feature's state into the system registry. The feature state may be either absent, advertised, or installed. The PublishFeatures action also writes the feature-component mapping into the system registry for each installed feature.

The PublishFeatures action queries the [FeatureComponents table](#), [Feature table](#), and [Component table](#).

## Sequence Restrictions

The PublishFeatures action must come before [PublishProduct](#).

## ActionData Messages

Field	Description of action data
[1]	Identifier of advertised feature.

## Remarks

The PublishFeatures action publishes the composition of all features that are selected to be advertised or installed.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PublishProduct Action

The PublishProduct action manages the advertisement of the product information with the system. This action publishes the product if the product is in advertise mode or if any feature is being installed or reinstalled.

## Sequence Restrictions

The PublishProduct action must come after the [PublishFeatures](#) action.

## ActionData Messages

Field	Description of Action Data
[1]	Identifier of advertised product.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RegisterClassInfo Action

The RegisterClassInfo action manages the registration of COM class information with the system. It uses the [AppId table](#).

## Sequence Restrictions

The RegisterClassInfo action must come after the [InstallFiles](#) action and the [UnregisterClassInfo](#) action.

The sequencing of the actions in the following group is restricted. If any subset of these actions occur together in a sequence table, they must have the same relative sequence order as shown:

- [UnregisterClassInfo](#)
- [UnregisterExtensionInfo](#)
- [UnregisterProgIdInfo](#)
- [UnregisterMIMEInfo](#)
- RegisterClassInfo
- [RegisterExtensionInfo](#)
- [RegisterProgIdInfo](#)
- [RegisterMIMEInfo](#)

For example, RegisterClassInfo must come after UnregisterMIMEInfo in the sequence table.

## ActionData Messages

Field	Description of action data
[1]	GUID class identifier of the registered COM server.

## Remarks

If the system supports install-on-demand for OLE servers,

RegisterClassInfo registers all COM classes in the [Class table](#) associated with a feature selected to be installed or advertised. Otherwise this action only registers the COM classes associated with a feature selected for installation.

## See Also

[OLEAdvtSupport property](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RegisterComPlus Action

The RegisterComPlus action registers COM+ applications.

## Sequence Restrictions

The RegisterComPlus action must follow the [InstallFiles action](#) and the [UnregisterComPlus action](#).

## ActionData Messages

Field	Description of action data
[1]	Application ID of the COM+ application.

## See Also

[Complus table](#)

[UnregisterComPlus action](#)

[Installing a COM+ Application with the Windows Installer](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RegisterExtensionInfo Action

The RegisterExtensionInfo action manages the registration of extension related information with the system.

## Sequence Restrictions

The RegisterExtensionInfo action must come after the [InstallFiles](#) action and the [UnregisterExtensionInfo](#) action.

The sequencing of the actions in the following group is restricted. If any subset of these actions occur together in a sequence table, they must have the same relative sequence order as shown:

- [UnregisterClassInfo](#)
- [UnregisterExtensionInfo](#)
- [UnregisterProgIdInfo](#)
- [UnregisterMIMEInfo](#)
- [RegisterClassInfo](#)
- RegisterExtensionInfo
- [RegisterProgIdInfo](#)
- [RegisterMIMEInfo](#)

For example, RegisterExtensionInfo must come after UnregisterMIMEInfo in the sequence table.

## ActionData Messages

Field	Description of action data
[1]	Registered extension.

## Remarks

If the system supports installation-on-demand for extension servers,

RegisterExtensionInfo registers all extension servers in the [Extension table](#) associated with features set for installation or advertisement. Otherwise this action only registers extension servers associated with features set to installation.

## See Also

[ShellAdvtSupport property](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RegisterFonts Action

The RegisterFonts action registers installed fonts with the system. This action maps the font name in the FontTitle column of the [Font table](#) to the path of the font file installed.

## Sequence Restrictions

The [InstallFiles](#) action must be called before calling the RegisterFonts action.

## ActionData Messages

Field	Description of action data
[1]	Font file.

## Remarks

The RegisterFonts action is executed if the file specified in the File\_ column of the [Font table](#) belongs to a component being installed.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RegisterMIMEInfo Action

The RegisterMIMEInfo action registers MIME-related registry information with the system.

## Sequence Restrictions

The RegisterMIMEInfo action must come after the [InstallFiles](#) action, [UnregisterMIMEInfo](#) action, [RegisterClassInfo](#) action, and the [RegisterExtensionInfo](#) action.

The sequencing of the actions in the following group is restricted. If any subset of these actions occur together in a sequence table, they must have the same relative sequence order as shown:

- [UnregisterClassInfo](#)
- [UnregisterExtensionInfo](#)
- [UnregisterProgIdInfo](#)
- [UnregisterMIMEInfo](#)
- [RegisterClassInfo](#)
- [RegisterExtensionInfo](#)
- [RegisterProgIdInfo](#)
- [RegisterMIMEInfo](#)

For example, RegisterMIMEInfo must come after UnregisterMIMEInfo in the sequence table.

## ActionData Messages

Field	Description of action data
[1]	Identifier of registered MIME content type.
[2]	Extension associated with MIME content type.

## Remarks

The RegisterMIMEInfo action registers all MIME information for servers from the [MIME table](#) for which the corresponding class server or extension server has been selected to be installed.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RegisterProduct Action

The RegisterProduct action registers the product information with the installer and with Add/Remove Programs. Additionally, this action stores the Windows Installer database package on the local computer.

The RegisterProduct action configures the controls displayed by the Add/Remove Programs in Control Panel. For more information, see [Configuring Add/Remove Programs with Windows Installer](#).

## Sequence Restrictions

There are no sequence restrictions.

## ActionData Messages

Field	Description of action data
[1]	Information about registered product.

## Remarks

The RegisterProduct action is not performed during installation to an administrative installation point.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RegisterProgIdInfo Action

The RegisterProgIdInfo action manages the registration of OLE ProgId information with the system.

## Sequence Restrictions

The RegisterProgIdInfo action must come after the [InstallFiles](#) action, [UnregisterProgIdInfo](#) action, [RegisterClassInfo](#) action, and the [RegisterExtensionInfo](#) action.

The sequencing of the actions in the following group is restricted. If any subset of these actions occur together in a sequence table, they must have the same relative sequence order as shown:

- [UnregisterClassInfo](#)
- [UnregisterExtensionInfo](#)
- [UnregisterProgIdInfo](#)
- [UnregisterMIMEInfo](#)
- [RegisterClassInfo](#)
- [RegisterExtensionInfo](#)
- [RegisterProgIdInfo](#)
- [RegisterMIMEInfo](#)

For example, RegisterProgIdInfo must come after RegisterExtensionInfo in the sequence table.

## ActionData Messages

Field	Description of action data
[1]	Program identifier of registered program.

## Remarks

The RegisterProgIdInfo action registers all ProgId information for servers that are specified in the [ProgId table](#) and for which the corresponding class server or extension server has been selected to be installed.

## **See Also**

[Class table](#)

[Extension table](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RegisterTypeLibraries Action

The RegisterTypeLibraries action registers type libraries with the system. This action is applied to each file referred to in the [TypeLib table](#) that is scheduled for install.

## Sequence Restrictions

The RegisterTypeLibraries action must come after the [InstallFiles](#) action.

## ActionData Messages

Field	Description of action data
[1]	<a href="#">GUID</a> of registered type library.

## Remarks

The RegisterTypeLibraries action requires that the type library language be correctly authored in the Language field of the TypeLib table. Incorrect authoring of the Language field can cause the installer to fail to register an otherwise valid type library.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RegisterUser Action

The RegisterUser action registers the user information with the installer to identify the user of a product.

## Sequence Restrictions

There are no sequence restrictions.

## ActionData Messages

Field	Description of action data
[1]	Registered user information.

## Remarks

The RegisterUser action is not performed during an administrative installation. If the product identifier entered by the user has not been validated by the [ValidateProductID action](#), the **ProductID** property is not set and this action does nothing.

## See Also

[USERNAME](#)

[COMPANYNAME](#)

[ProductID](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RemoveDuplicateFiles Action

The RemoveDuplicateFiles action deletes files installed by the [DuplicateFiles](#) action.

## Sequence Restrictions

The RemoveDuplicateFiles action must occur after the [InstallValidate](#) action and before the [InstallFiles](#) action.

## ActionData Messages

Field	Description of action data
[1]	Identifier of removed file.
[9]	Identifier of directory holding removed file.

## Remarks

To delete a file duplicated with the [DuplicateFiles action](#) using the RemoveDuplicateFiles action, the component associated with the file's entry in the DuplicateFile table must be selected for removal.

Use the DestFolder column of the [DuplicateFile table](#) to specify the property name whose value identifies the destination folder.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RemoveEnvironmentStrings Action

The RemoveEnvironmentStrings action modifies the values of environment variables.

Note that environment variables do not change for the installation in progress when either the [WriteEnvironmentStrings action](#) or RemoveEnvironmentStrings action are run. On Windows 2000, this information is stored in the registry and a message is sent to notify the system of changes when the installation completes. A new process, or another process that checks for these messages, will use the new environment variables.

The installer runs the [WriteEnvironmentStrings action](#) only during the installation or reinstallation of a component, and runs the RemoveEnvironmentStrings action only during the removal of a component.

Values are written or removed based on the selection of primary actions and modifiers. These are described in the following [ActionData Messages](#) section. Note that depending upon the action specified, WriteEnvironmentStrings may remove variables, and RemoveEnvironmentStrings may add them based on the authoring of the [Environment table](#).

## Sequence Restrictions

The [InstallValidate action](#) must be executed before the RemoveEnvironmentStrings action. Because the WriteEnvironmentStrings action and RemoveEnvironmentStrings action are never both applied during the installation or removal of a component, their relative sequence is not restricted.

## ActionData Messages

Field	Description of action data
[1]	Name of the environment variable to modify.
[2]	The environment variable value.

[3]	This is a field of bit flags that specify the action to be performed. Include only one bit for a primary action. There may be more than one modifier bit included in this field. See the following bit flag descriptions.
-----	---

<b>Bit value</b>	<b>Description of primary actions</b>
0x1	Set. Sets the value of the environment variable in all cases. If this bit is combined with an Append or Prefix modifier bit, the action adds the value to any existing value in the variable.
0x2	Set. Sets the value if the variable is absent. If this bit is combined with an Append or Prefix modifier bit, the action adds the value to any existing value in the variable.
0x4	Remove. Removes the value from the variable. If this bit is combined with an Append or Prefix modifier bit, the value is removed from the existing string, if the value exists.

<b>Bit value</b>	<b>Description of modifier</b>
0x20000000	If this bit is set, actions are applied to the machine environment variables. If this bit is not set, actions are applied to the user's environment variables.
0x40000000	Append. This bit is optional. Do not set both the Append and Prefix modifiers.
0x80000000	Prefix. This bit is optional. Do not set both the Append and Prefix modifiers.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RemoveExistingProducts Action

The RemoveExistingProducts action goes through the product codes listed in the ActionProperty column of the [Upgrade table](#) and removes the products in sequence by invoking concurrent installations. For each concurrent installation the installer sets the **ProductCode** property to the product code and sets the **REMOVE** property to the value in the Remove field of the Upgrade table. If the Remove field is blank, its value defaults to ALL and the installer removes the entire product.

The installer only runs the RemoveExistingProducts action the first time it installs a product. It does not run the action during a [maintenance installation](#) or uninstallation.

## Sequence Restrictions

The RemoveExistingProducts action must be scheduled in the action sequence in one of the following locations.

- Between the [InstallValidate action](#) and the [InstallInitialize action](#). In this case, the installer removes the old applications entirely before installing the new applications. This is an inefficient placement for the action because all reused files have to be recopied.
- After the [InstallInitialize action](#) and before any actions that generate execution script.
- Between the [InstallExecute action](#), or the [InstallExecuteAgain action](#), and the [InstallFinalize action](#). Generally the last three actions are scheduled right after one another: InstallExecute, RemoveExistingProducts, and InstallFinalize. In this case the updated files are installed first and then the old files are removed. However, if the removal of the old application fails, then the installer rolls back both the removal of the old application and the install of the new application.
- After the [InstallFinalize action](#). This is the most efficient placement for the action. In this case, the installer updates files before removing

the old applications. Only the files being updated get installed during the installation. If the removal of the old application fails, then the installer only rolls back the uninstallation of the old application.

## ActionData Messages

Field	Description of action data
[1]	Removed product.

## Remarks

Windows Installer sets the **UPGRADINGPRODUCTCODE** Property when it runs this action.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RemoveFiles Action

The RemoveFiles action removes files previously installed by the [InstallFiles](#) action. Each of these files is gated by a link to an entry in the [Component](#) table. Only those files with components resolved to either the *msiInstallStateAbsent* state or the *msiInstallStateLocal* state if the component is installed locally, are removed.

## Sequence Restrictions

The [InstallValidate](#) action must be called before calling RemoveFiles. If an [InstallFiles](#) action is used, it must appear after RemoveFiles.

## ActionData Messages

Field	Description of action data
[1]	Identifier of removed file.
[9]	Identifier of directory holding removed file.

## Remarks

The [RemoveFile](#) table can be omitted from the installer database if there are no miscellaneous files to remove.

The RemoveFiles action can also remove author-specified files that are not installed by the InstallFiles action. These files are specified in the [RemoveFile](#) table. Each of these files is gated by a link to an entry in the [Component](#) table. Those files whose components are resolved to any active Action state (that is, not in the Off or Null state) are removed if the file exists in the specified directory. The removal of files specified in the RemoveFile table is attempted when the linked component is first installed, during a reinstall, and again when the linked component is removed.

The RemoveFiles action can also remove folders. An empty folder is removed if the value in the FileName column of the RemoveFile table is

null.

When removing previously installed files, the RemoveFiles action queries the same fields in the same tables as those queried by the [InstallFiles](#) action with the exception that the [Media table](#) is not used by the RemoveFiles action.

The target file name can be specified in localized text in the FileName column of the RemoveFile table.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RemoveFolders Action

The RemoveFolders action removes any folders linked to components set to be removed or run from source. These folders are removed only if they are empty. If a folder is removed, it is unregistered with the appropriate component identifier.

## Sequence Restrictions

The RemoveFolders action must come after the [RemoveFiles](#) action or any action that may remove files from folders.

## ActionData Messages

Field	Description of action data
[1]	Identifier of removed folder.

## Remarks

The installer does not automatically remove the folders created by the [CreateFolders action](#) during an uninstallation of the application. The installer must be instructed to remove these folders by authoring the RemoveFolders action into the action sequence.

To specify the name and location of the folder, use the Directory\_ column of the [CreateFolder table](#). Each folder name in the CreateFolder table is assumed to be a property defining the folder location.

To specify the component owning the folder, use the ComponentId column of the [Component table](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RemoveIniValues Action

The RemoveIniValues action removes .ini file information specified for removal in the [RemoveIniFile table](#) if the component is set to be installed locally or run-from-source. The RemoveIniValues action removes .ini file information that has been associated with a component in the [IniFile table](#). This action also removes .ini file information if the information was written by the [WriteIniValues action](#) and the component is scheduled to be uninstalled.

## Sequence Restrictions

The [InstallValidate](#) action must be called before the RemoveIniValues action. If a [WriteIniValues](#) action is used in the sequence, it must appear after RemoveIniValues.

## ActionData Messages

Field	Description of action data
[1]	Identifier of .ini file.
[2]	An .ini file key section.
[3]	Item removed from .ini file.
[4]	Value removed from .ini file.

## See Also

[RemoveIniFile table](#)

[IniFile table](#)

[WriteIniValues action](#)

[InstallValidate](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RemoveODBC Action

The RemoveODBC action removes the data sources, translators, and drivers listed for removal during the installation. This action queries the [ODBCDataSource table](#), [ODBCTranslator table](#), and [ODBCDriver table](#) for each data source, translator, or driver scheduled for removal.

## Sequence Restrictions

There are no sequence restrictions.

## ActionData Messages

For each driver installed.

Field	Description of action data
[1]	Driver description. The ODBC driver key.
[2]	ComponentId

For each translator installed.

Field	Description of action data
[1]	Driver description. The ODBC driver key.
[2]	ComponentId

For each data source installed.

Field	Description of action data
[1]	Driver description. The ODBC driver key.
[2]	ComponentId
[3]	Registration: SQL_REMOVE_DSN or SQL_REMOVE_SYS_DSN

## Remarks

If both the ODBC usage count and the file usage count become zero, the file is removed. Registration is dependent on ODBC usage counts and file removal is based on shared DLLs key reference counts.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RemoveRegistryValues Action

The RemoveRegistryValues action can only remove values from the system registry that have been authored into the [Registry table](#) or the [RemoveRegistry table](#). This action removes a registry value that has been authored into the Registry table if the associated component was installed locally or as run from source and is now set to be uninstalled. This action removes a registry value that has been authored into the RemoveRegistry table if the associated component is set to be installed locally or as run from source.

## Sequence Restrictions

The [InstallValidate](#) action must be called before calling RemoveRegistryValues. If a [WriteRegistryValues](#) action is used, it must come after RemoveRegistryValues. RemoveRegistryValues must come before [UnregisterMIMEInfo](#) or [UnregisterProgIDInfo](#).

A custom action can be used to add rows to the Registry table during an installation, uninstallation, or repair transaction. These rows do not persist in the Registry table and the information is only available during the current transaction. The custom action must therefore be run in every installation, uninstallation, or repair transaction that requires the information in these additional rows. The custom action must come before the RemoveRegistryValues and WriteRegistryValues actions in the action sequence.

## ActionData Messages

Field	Description of action data
[1]	Registry path to key of removed registry value.
[2]	Formatted string of name of removed registry value.

## Remarks

To remove a registry value, record the value in the Value column of the Registry table. If the WriteRegistryValues action has attached REG\_MULTI\_SZ strings to the value in the [Registry table](#), then the RemoveRegistryValues action removes only those strings from the registry value.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RemoveShortcuts Action

The RemoveShortcuts action manages the removal of an advertised shortcut whose feature is selected for uninstallation or a nonadvertised shortcut whose component is selected for uninstallation. For more information, see the [Shortcut Table](#).

## Sequence Restrictions

The RemoveShortcuts action must come before the [CreateShortcuts](#) action.

## ActionData Messages

Field	Description of action data
[1]	Identifier of removed shortcut.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ResolveSource Action

The ResolveSource action determines the location of the source and sets the **SourceDir** property if the source has not been resolved yet.

## Sequence Restrictions

The ResolveSource action must come after the [CostInitialize action](#).

## ActionData Messages

There are no ActionData messages.

## Remarks

The ResolveSource action must be called before using the **SourceDir** property in any expression. It must also be called before attempting to retrieve the value of the **SourceDir** property using [MsiGetProperty](#). The ResolveSource action should not be executed when the source is unavailable, as it may be when uninstalling the application.

## See Also

[Source Resiliency](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RMCCPSearch Action

The RMCCPSearch action uses file signatures to validate that qualifying products are installed on a system before an upgrade installation is performed.

## Sequence Restrictions

RMCCPSearch action should be authored into the [InstallUISequence table](#) and [InstallExecuteSequence table](#). The installer prevents RMCCPSearch from running in the InstallExecuteSequence sequence if the action has already run in InstallUISequence sequence.

## ActionData Messages

There are no ActionData messages.

## Remarks

The RMCCPSearch action requires the **CCP\_DRIVE** property to be set to the root path on the removable *volume* that has the installation for any of the qualifying products.

Each file signature in the CCPSearch table is searched for under the path referred to by the **CCP\_DRIVE** property using the [DrLocator](#) table. The absence of the signature from the [Signature](#) table denotes a directory. If any signature is determined to exist, the RMCCPSearch action terminates.

## See Also

[CCPSearch table](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ScheduleReboot Action

Insert the ScheduleReboot action into the action sequence to prompt the user for a restart of the system at the end of the installation. Use the [ForceReboot action](#) to prompt for a restart during installation.

If the installation has a user interface, the installer displays a message box and button at the end of installation asking whether the user wants to restart the system. The user must respond to this prompt before completing installation. If the installation has no user interface, the system automatically restarts at the end.

If the installer determines that a restart is necessary it automatically prompts the user to restart at the end of the installation, whether or not there are any ForceReboot or ScheduleReboot actions in the sequence. For example, the installer automatically prompts for a restart if it needs to replace any files that are in use during installation.

You can suppress certain prompts for restarts by setting the [REBOOT](#) property.

If the Windows Installer encounters the ForceReboot or ScheduleReboot action during a [multiple-package installation](#), the installer will stop and roll back the installation. No package of the multiple-package installation will be installed.

## Sequence Restrictions

There are no sequence restrictions.

## ActionData Messages

There are no ActionData messages.

## Remarks

For example, this action can be used to force the installer to prompt for a restart after installing drivers that require a restart. If the installer attempts to replace files that are in use, it automatically prompts the user to restart even if ScheduleReboot has not been used.

The ScheduleReboot action is typically placed at the end of the sequence, but it can be inserted at any point in the sequence.

## **See Also**

[System Reboots](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SelfRegModules Action

The SelfRegModules action processes all modules listed in the [SelfReg](#) table and registers all installed modules with the system. The installer does not self-register EXE files.

## Sequence Restrictions

The [InstallValidate](#) action must be called before calling the SelfRegModules action. If an [InstallFiles](#) action is used, it must appear before the SelfRegModules action in the sequence. Because the SelfRegModules action changes the system, SelfRegModules should come after the [InstallInitialize](#) action.

## ActionData Messages

Field	Description of action data
[1]	Identifier of registered module file.
[2]	Identifier of folder holding registered module file.

## Remarks

The SelfRegModules action attempts to call the **DllRegisterServer** function of the module scheduled to be registered. This action runs with elevated privileges when the installation is being run with elevated privileges, such as during a per-machine installation. During a per-user installation the installer runs this action with user privileges.

Note that you cannot specify the order in which the installer registers self-registering DLLs by using the [SelfUnRegModules](#) action.

## See Also

[Specifying the Order of Self Registration](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SelfUnregModules Action

The SelfUnregModules action unregisters all modules listed in the [SelfReg table](#) that are scheduled to be uninstalled. The installer does not self-register .EXE files.

## Sequence Restrictions

The [InstallValidate](#) action must appear before the SelfUnregModules action in the sequence. If a [SelfRegModules](#) action is used it must appear after the SelfUnregModules action in the sequence. If a [RemoveFiles action](#) is used it must appear after the SelfUnregModules action in the sequence.

## ActionData Messages

Field	Description of action data
[1]	Identifier of unregistered module file.
[2]	Identifier of folder holding unregistered module file.

## Remarks

The SelfUnregModules action attempts to call the **DllUnregisterServer** function of the module that is to be unregistered. This action runs with elevated privileges when the installation is being run with elevated privileges, such as during a per-machine installation. During a per-user installation, the installer runs this action with user privileges.

Note that you cannot specify the order in which the installer unregisters self-registering DLLs by using the SelfUnRegModules action.

## See Also

[Specifying the Order of Self Registration](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SEQUENCE Action

The SEQUENCE action sorts and runs in sequence the actions in a table.

The **SEQUENCE** property defines the table used by this action. The table must have the same schema as the [InstallExecuteSequence table](#). The table must have an Action, Condition, and Sequence column.

## Sequence Restrictions

There are no sequence restrictions.

## ActionData Messages

There are no ActionData messages.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SetODBCFolders Action

The SetODBCFolders action checks for existing ODBC drivers on the system and sets the target directory of each new driver to the location of an existing driver.

## Sequence Restrictions

The SetODBCFolders action must come after the [CostFinalize action](#) and before the [InstallValidate action](#).

## ActionData Messages

Field	Description of action data
[1]	Driver description. The ODBC driver key.
[2]	Original folder location of the new driver.
[3]	New folder location for the new driver. This is the location of an existing driver.

## Remarks

This action places a new driver into the same directory as the existing driver being replaced. The SetODBCFolders action uses the [Directory table](#) to set the locations of existing drivers that are to be replaced.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# StartServices Action

The StartServices action starts system services. This action queries the [ServiceControl table](#).

## Sequence Restrictions

The services actions must be used in the following sequence:

- [StopServices](#)
- [DeleteServices](#)

Any of the following actions:

- [InstallFiles](#)
- [RemoveFiles](#)
- [DuplicateFiles](#)
- [MoveFiles](#)
- [PatchFiles](#)
- [RemoveDuplicateFiles](#)
- [InstallServices](#)
- [StartServices](#)

## ActionData Messages

Field	Description of action data
[1]	Service display name.
[2]	Service name.

## Remarks

This action requires that the user be an administrator or have elevated

privileges with permission to control services or that the application be part of a managed installation.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# StopServices Action

The StopServices action stops system services. This action queries the [ServiceControl table](#).

## Sequence Restrictions

The services actions must be used in the following sequence:

- StopServices
- [DeleteServices](#)

Any of the following actions:

- [InstallFiles](#)
- [RemoveFiles](#)
- [DuplicateFiles](#)
- [MoveFiles](#)
- [PatchFiles](#)
- [RemoveDuplicateFiles](#)
- [InstallServices](#)
- [StartServices](#)

## ActionData Messages

Field	Description of action data
[1]	Service display name.
[2]	Service name.

## Remarks

This action requires that the user be an administrator or have elevated

privileges with permission to control services or that the application be part of a managed installation.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UnpublishComponents Action

The UnpublishComponents action manages the unadvertisement of components listed in the PublishComponent table. These are components that may be faulted in by other products. This action removes information about published components from the [PublishComponent table](#) for which the corresponding feature is currently selected to be uninstalled.

## Sequence Restrictions

There are no sequence restrictions.

## ActionData Messages

Field	Description of action data
[1]	GUID representing the component ID of an advertised feature.
[2]	Qualifier of the component ID.

## ActionData Messages

There are no ActionData messages.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UnpublishFeatures Action

When an entire product is uninstalled, the UnpublishFeatures action removes selection-state and feature-component mapping information from the system registry. This action queries the [FeatureComponents Table](#).

## Sequence Restrictions

There are no sequence restrictions.

## ActionData Messages

Field	Description of action data
[1]	Identifier of an unadvertised feature.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UnregisterClassInfo Action

The UnregisterClassInfo action manages the removal of COM class information from the system registry. It uses the [AppId table](#).

## Sequence Restrictions

The UnregisterClassInfo action must come after the [InstallInitialize](#) action and before the [RegisterClassInfo](#) action.

[RemoveRegistryValues](#) must come before UnregisterClassInfo in the sequence.

The sequencing of the actions in the following group is restricted. If any subset of these actions occur together in a sequence table, they must occur in the same relative sequence as shown in the following table:

- UnregisterClassInfo
- [UnregisterExtensionInfo](#)
- [UnregisterProgIdInfo](#)
- [UnregisterMIMEInfo](#)
- [RegisterClassInfo](#)
- [RegisterExtensionInfo](#)
- [RegisterProgIdInfo](#)
- [RegisterMIMEInfo](#)

For example, [RegisterExtensionInfo](#) must come before [UnregisterClassInfo](#) in the sequence table.

## ActionData Messages

Field	Description of action data
[1]	GUID of unregistered COM class.

## Remarks

The installer sets the [OLEAdvtSupport](#) property to true when the current user's system has been upgraded to work with install-on-demand through COM. If the system does not support install-on-demand through COM, UnregisterClassInfo removes all COM classes listed in the [Class table](#) associated with either uninstalled features or features installed as advertised from the system registry. Otherwise, this action removes only the COM classes associated with features selected to be uninstalled from the system registry.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UnregisterComPlus Action

The UnregisterComPlus action removes COM+ applications from the registry.

## Sequence Restrictions

The [RegisterComPlus action](#) must follow the [InstallFiles action](#) and the UnregisterComPlus action.

## ActionData Messages

Field	Description of action data
[1]	Application identifier of the COM+ application being removed.

## See Also

[RegisterComPlus action](#)

[Complus table](#)

[Installing a COM+ Application with the Windows Installer](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UnregisterExtensionInfo Action

The UnregisterExtensionInfo action manages the removal of extension-related information from the system registry.

## Sequence Restrictions

The UnregisterExtensionInfo action must come after the [InstallInitialize](#) action and before the [RegisterExtensionInfo](#) action.

[RemoveRegistryValues](#) must come before UnregisterExtensionInfo in the sequence.

The sequencing of the actions in the following group is restricted. If any subset of these actions occur together in a sequence table, they must have the same relative sequence order as shown:

- [UnregisterClassInfo](#)
- UnregisterExtensionInfo
- [UnregisterProgIdInfo](#)
- [UnregisterMIMEInfo](#)
- [RegisterClassInfo](#)
- [RegisterExtensionInfo](#)
- [RegisterProgIdInfo](#)
- [RegisterMIMEInfo](#)

For example, UnregisterExtensionInfo must come before UnregisterProgIdInfo in the sequence table.

## ActionData Messages

Field	Description of action data
[1]	Removed extension.

## Remarks

If the system does not support the install-on-demand of extension servers, UnregisterExtensionInfo removes all extension servers in the [Extension table](#) associated with either an uninstalled feature or a feature installed as advertised from the registry. Otherwise, this action removes the extension servers associated with a feature that is selected to be removed from the registry.

## See Also

### [ShellAdvtSupport Property](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UnregisterFonts Action

The UnregisterFonts action removes registration information about installed fonts from the system.

## Sequence Restrictions

The [RemoveFiles](#) action must be called after UnregisterFonts.

## ActionData Messages

Field	Description of action data
[1]	Font file.

## Remarks

The UnregisterFonts action is executed if the file specified in the File\_ column of the [Font table](#) belongs to a component being uninstalled.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UnregisterMIMEInfo Action

The UnregisterMIMEInfo action unregisters MIME-related registry information from the system. The action unregisters MIME information for servers from the [MIME table](#) for which the corresponding feature is currently selected to be uninstalled.

## Sequence Restrictions

The UnregisterMIMEInfo action must come after the [InstallInitialize](#) action, [UnregisterClassInfo](#) action, [UnregisterExtensionInfo](#) action, and before the [RegisterMIMEInfo](#) action.

[RemoveRegistryValues](#) must come before UnregisterMIMEInfo in the sequence.

The sequencing of the actions in the following group is restricted. If any subset of these actions occur together in a sequence table, they must have the same relative sequence order as shown:

- [UnregisterClassInfo](#)
- [UnregisterExtensionInfo](#)
- [UnregisterProgIdInfo](#)
- UnregisterMIMEInfo
- [RegisterClassInfo](#)
- [RegisterExtensionInfo](#)
- [RegisterProgIdInfo](#)
- [RegisterMIMEInfo](#)

For example, UnregisterMIMEInfo must come before RegisterExtensionInfo in the sequence table.

## ActionData Messages

Field	Description of action data
[1]	Identifier of unregistered MIME content type.

[2]

Extension associated with MIME content type.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UnregisterProgIdInfo Action

The UnregisterProgIdInfo action manages the unregistration of OLE ProgId information with the system.

## Sequence Restrictions

The UnregisterProgIdInfo action must come after the [InstallInitialize](#) action, [UnregisterClassInfo](#) action, [UnregisterExtensioninfo](#) action, and before the [RegisterProgIdInfo](#) action.

[RemoveRegistryValues](#) must come before UnregisterProgIdInfo in the sequence.

The sequencing of the actions in the following group is restricted. If any subset of these actions occur together in a sequence table, they must have the same relative sequence order as shown:

- [UnregisterClassInfo](#)
- [UnregisterExtensioninfo](#)
- UnregisterProgIdInfo
- [UnregisterMIMEInfo](#)
- [RegisterClassInfo](#)
- [RegisterExtensionInfo](#)
- [RegisterProgIdInfo](#)
- [RegisterMIMEInfo](#)

For example, UnregisterProgIdInfo must come before UnregisterMIMEInfo in the sequence table.

## ActionData Messages

Field	Description of action data
[1]	Program identifier of registered program.

## Remarks

The UnregisterProgIdInfo action removes ProgId information from the registry ([ProgId Table](#)) for features that are connected to the extension information ([Extension table](#)) or the Class information ([Class table](#)) and are currently selected to be uninstalled.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UnregisterTypeLibraries Action

The UnregisterTypeLibraries action unregisters type libraries from the system. This action is applied to each file listed in the [TypeLib](#) table that is triggered to be uninstalled.

## Sequence Restrictions

The UnregisterTypeLibraries action must come before the [RemoveFiles](#) action.

## ActionData Messages

Field	Description of action data
[1]	<a href="#">GUID</a> of an unregistered type library.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ValidateProductID Action

The ValidateProductID action sets the **ProductID** property to the full product identifier.

## Sequence Restrictions

This action must be sequenced before the user interface wizard in the [InstallUISequence table](#) and before the [RegisterUser action](#) in the [InstallExecuteSequence table](#).

## ActionData Messages

There are no ActionData messages.

## Remarks

The installer checks whether a product has validated successfully by checking the **ProductID** property. The installer sets the **ProductID** property to the full product identifier after a successful validation. The ValidateProductID action does nothing if the **ProductID** property has already been set by a successful validation or by another method.

The ValidateProductID action always returns a success, whether or not the product identifier is valid, so that the product identifier can be entered on the command line the first time the product is run.

The product identifier can be validated without having the user reenter this information by setting the **PIDKEY** property on the command line or by using a transform. The display of the dialog box requesting the user to enter the product identifier can then be made conditional upon the presence of the **ProductID** property, which is set when the **PIDKEY** property is validated.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# WriteEnvironmentStrings Action

The WriteEnvironmentStrings action modifies the values of environment variables.

Environment variables do not change for the installation in progress when the WriteEnvironmentStrings action or [RemoveEnvironmentStrings action](#) are run. On Windows 2000, Windows Server 2003, Windows XP, and Windows Vista this information is stored in the registry and a **WM\_SETTINGCHANGE** message is sent to notify the system of the changes when the installation completes. Another process can receive notification of the changes by handling these messages. No message is sent if a restart of the system is pending. A package can use the [MsiSystemRebootPending](#) property to check whether a system restart is pending.

The installer runs the WriteEnvironmentStrings action only during the installation or reinstallation of a component, and runs the [RemoveEnvironmentStrings action](#) only during the removal of a component.

Values are written or removed based on the selection of primary actions and modifiers. These are described in the following [ActionData Messages](#) section. Note that depending on the action specified, WriteEnvironmentStrings may remove variables, and RemoveEnvironmentStrings may add them based on the authoring of the [Environment table](#).

## Sequence Restrictions

The [InstallValidate action](#) must be executed before the RemoveEnvironmentStrings action. Because the WriteEnvironmentStrings action and RemoveEnvironmentStrings action are never both applied during an install or removal of a component, their relative sequence is not restricted.

## ActionData Messages

Field	Description of action data
-------	----------------------------

[1]	Name of the environment variable to modify.
[2]	The environment variable value.
[3]	This is a field of bit flags that specifies the action to be performed. Include only one bit for a primary action. There may be more than one modifier bit included in this field. See the following bit flag descriptions.

<b>Bit Value</b>	<b>Description of primary actions</b>
0x1	Set. Sets the value of the environment variable in all cases. If this bit is combined with an Append or Prefix modifier bit, the action adds the value to any existing value in the variable.
0x2	Set. Sets the value if the variable is absent. If this bit is combined with an Append or Prefix modifier bit, the action adds the value to any existing value in the variable.
0x4	Remove. Removes the value from the variable. If this bit is combined with an Append or Prefix modifier bit, the value is removed from the existing string, if the value exists.

<b>Bit Value</b>	<b>Description of modifier</b>
0x20000000	If this bit is set, actions are applied to the machine environment variables. If this bit is not set, actions are applied to the user's environment variables.
0x40000000	Append. This bit is optional. Do not set both the Append and Prefix modifiers.
0x80000000	Prefix. This bit is optional. Do not set both the Append and Prefix modifiers.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# WriteIniValues Action

The WriteIniValues action writes the .ini file information that the application needs written to its .ini files. The writing of this information is gated by the [Component table](#). An .ini value is written if the corresponding component has been set to be installed either locally or run from source.

## Sequence Restrictions

The InstallValidate action must come before the WriteIniValues action. If there is a [RemoveIniValues action](#) in the sequence, then it must come before the WriteIniValues action.

## ActionData Messages

Field	Description of action data
[1]	Identifier of .ini file.
[2]	.ini file key in the following section.
[3]	Item removed from .ini file.
[4]	Value removed from .ini file.

## See Also

[IniFile table](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# WriteRegistryValues Action

The WriteRegistryValues action sets up an application's registry information. The registry information is gated by the [Component table](#). A registry value is written to the registry if the corresponding component has been set to be installed either locally or as run from source. For more information, see [Registry table](#).

## Sequence Restrictions

The [InstallValidate action](#) must come before the WriteRegistryValues action. If there is a [RemoveRegistryValues action](#), then it must come before the WriteRegistryValues action.

A custom action can be used to add rows to the Registry table during an installation, uninstallation, or repair transaction. These rows do not persist in the Registry table and the information is only available during the current transaction. The custom action must therefore be run in every installation, uninstallation, or repair transaction that requires the information in these additional rows. The custom action must come before the RemoveRegistryValues and WriteRegistryValues actions in the action sequence.

## ActionData Messages

Field	Description of action data
[1]	Path to registry key of value written to registry.
[2]	Formatted text string name of value written to registry.
[3]	Formatted text string of value written to registry.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Actions

The Windows Installer provides many built-in actions for performing the installation process. For a list of these actions, see the [Standard Actions Reference](#).

Standard actions are sufficient to execute an installation in most cases. However, there are situations where the developer of an installation package finds it necessary to write a custom action. For example:

- You want to launch an executable during installation that is installed on the user's machine or that is being installed with the application.
- You want to call special functions during an installation that are defined in a dynamic-link library (DLL).
- You want to use functions written in the development languages Microsoft Visual Basic Scripting Edition or Microsoft JScript literal script text during an installation.
- You want to defer the execution of some actions until the time when the installation script is being executed.
- You want to add time and progress information to a ProgressBar control and a TimeRemaining Text control.

The following sections describe custom actions and how to incorporate them into an installation package:

- [About Custom Actions](#)
- [Using Custom Actions](#)
- [Custom Action Reference](#)
- [Summary List of All Custom Action Types](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# About Custom Actions

Although standard actions are sufficient to execute an installation in most cases, custom actions enable the author of an installation package to extend the capabilities of standard actions by including executables, dynamic-link libraries, and script. Custom actions cannot be used in the sequence tables used for advertisement, the [AdvtUISequence table](#) and the [AdvtExecuteSequence table](#).

The following sections provide more information about custom actions.

[Custom Action Sources](#)

[Synchronous and Asynchronous Custom Actions](#)

[Rollback Custom Actions](#)

[Commit Custom Actions](#)

[Deferred Execution Custom Actions](#)

[Executable Files](#)

[Dynamic-Link Libraries](#)

[Scripts](#)

[Formatted Text Custom Actions](#)

[Error Message Custom Actions](#)

[Custom Action Security](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Sources

A custom action source can exist in several different forms. The following table lists the types of custom actions that can have each form of source. See the topic for the basic custom action type for more information.

Custom action source	Associated custom action types
Custom action stored in the Binary table.	<a href="#">Custom Action Type 1</a> , <a href="#">Custom Action Type 2</a> , <a href="#">Custom Action Type 5</a> , <a href="#">Custom Action Type 6</a>
Custom action copied during installation.	<a href="#">Custom Action Type 17</a> , <a href="#">Custom Action Type 18</a> , <a href="#">Custom Action Type 21</a> , <a href="#">Custom Action Type 22</a>
Custom action referencing a directory.	<a href="#">Custom Action Type 34</a>
Custom action referencing a property.	<a href="#">Custom Action Type 50</a> , <a href="#">Custom Action Type 53</a> , <a href="#">Custom Action Type 54</a>
Custom action stored as literal script code in database table.	<a href="#">Custom Action Type 37</a> , <a href="#">Custom Action Type 38</a>
Custom action displaying an error message.	<a href="#">Custom Action Type 19</a>

See [Summary List of All Custom Action Types](#) for a summary of all types of custom actions and how sources are specified in the [CustomAction table](#).

## See Also

[About Custom Actions](#)  
[Using Custom Actions](#)  
[Custom Action Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Synchronous and Asynchronous Custom Actions

The Windows Installer processes custom actions as a separate thread from the main installation. During synchronous execution of a custom action, the installer waits for the thread of the custom action to complete before continuing the main installation. During asynchronous execution, the installer runs the custom action simultaneously as the current installation continues. Authors of custom actions must therefore be aware of any asynchronous threads that may be making changes to the installation database between function calls.

In particular, calls to [MsiGetTargetPath](#) and [MsiSetTargetPath](#) should be avoided in asynchronous custom actions. Instead use [MsiGetProperty](#) to obtain a target path. Bulk database operations such as import, export, and transform operations should be avoided in any type of custom action.

Option flags can be set in the Type field of the [CustomAction table](#) to specify that the main and custom action threads run synchronously or asynchronously. See [Custom Action Return Processing Options](#).

The installer can only execute [Rollback Custom Actions](#) and [Concurrent Installation](#) actions as synchronous custom actions.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Rollback Custom Actions

When the installer processes the installation script, it simultaneously generates a rollback script. In addition to the rollback script, the installer saves a copy of every file it deletes during the installation. These files are kept in a hidden system directory. Once the installation is complete, the rollback script and the saved files are deleted. If an installation is unsuccessful, the installer attempts to rollback the changes made during the installation and restore the original state of the computer.

Although custom actions that schedule system operations by inserting rows into database table are reversed by a rollback of the installation, custom actions that change the system directly, or that issue commands to other system services, cannot always be reversed by a rollback. A rollback custom action is an action that the installer executes only during an installation rollback, and its purpose is to reverse a custom action that has made changes to the system.

A rollback custom action is a type of [deferred execution custom action](#), because its execution is deferred when it is invoked during the installation sequence. It differs from a regular deferred custom action in that it is only executed during a rollback. A rollback custom action must always precede the deferred custom action it rolls back in the action sequence. A rollback custom action should also handle the case where the deferred custom action is interrupted in the middle of execution. For example, if the user were to press the Cancel button while the custom action was executing.

Note that Rollback Custom Actions cannot run asynchronously. See [Synchronous and Asynchronous Custom Actions](#).

The complement to a rollback custom action is a [commit custom action](#). The installer executes a commit custom action during the installation sequence, copies the custom action into the rollback script, but does not execute the action during rollback.

Note that a rollback custom action may not be able to remove all of the changes made by commit custom actions. Although the installer writes both rollback and commit custom actions into the rollback script, commit custom actions only run after the installer has successfully processed the installation script. Commit custom actions are the first actions to run in

the rollback script. If a commit custom action fails, the installer initiates rollback but can only rollback those operations already written to the rollback script. This means that depending on the commit custom action, a rollback may not be able to undo the changes made by the action. You can ignore failures in commit custom actions by authoring the custom action to ignore return codes.

When the installer runs a rollback custom action, the only mode parameter it will set is `MSIRUNMODE_ROLLBACK`. See [MsiGetMode](#) for a description of the run mode parameters.

A rollback custom action can be specified by adding an option flag to the `Type` field of the [CustomAction table](#). See [Custom Action In-Script Execution Options](#) for the option flag designating a rollback custom action.

Rollback and commit custom actions do not run when rollback is disabled. If a package author requires these types of custom actions for proper installation, they should use the [RollbackDisabled](#) property in a condition that prevents the installation from continuing when rollback is disabled. For information on how to disable rollback see [Rollback Installation \(Windows Installer\)](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Commit Custom Actions

Commit Custom actions are executed upon successful completion of the installation script. If the [InstallFinalize action](#) is successful, the installer will then run any existing Commit Custom actions. The only mode parameter the installer sets in this case is MSIRUNMODE\_COMMIT. See [MsiGetMode](#) for a description of the run mode parameters.

A commit custom action can be specified by adding an option flag to the Type field of the [CustomAction table](#). See [Custom Action In-Script Execution Options](#) for the option flag designating a commit custom action.

A commit custom action is the complement to a [rollback custom action](#) and can be used with rollback custom actions to reverse custom actions that make changes directly to the system.

Note that a rollback custom action may not be able to remove all of the changes made by commit custom actions. Although the installer writes both rollback and commit custom actions into the rollback script, commit custom actions only run after the installer has successfully processed the installation script. Commit custom actions are the first actions to run in the rollback script. If a commit custom action fails, the installer initiates rollback but can only rollback those operations already written to the rollback script. This means that depending on the commit custom action, a rollback may not be able to undo the changes made by the action. You can ignore failures in commit custom actions by authoring the custom action to ignore return codes.

Rollback and commit custom actions do not run when rollback is disabled. If a package author requires these types of custom actions for proper installation, they should use the [RollbackDisabled](#) Property in a condition that prevents the installation from continuing when rollback is disabled.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Deferred Execution Custom Actions

The purpose of a deferred execution custom action is to delay the execution of a system change to the time when the installation script is executed. This differs from a regular custom action, or a standard action, in which the installer executes the action immediately upon encountering it in a sequence table or in a call to [MsiDoAction](#). A deferred execution custom action enables a package author to specify system operations at a particular point within the execution of the installation script.

The installer does not execute a deferred execution custom action at the time the installation sequence is processed. Instead the installer writes the custom action into the installation script. The only mode parameter the installer sets in this case is `MSIRUNMODE_SCHEDULED`. See [MsiGetMode](#) for a description of the run mode parameters.

A deferred execution custom action must be scheduled in the execute sequence table within the section that performs script generation. Deferred execution custom actions must come after [InstallInitialize](#) and come before [InstallFinalize](#) in the action sequence.

Custom actions that set properties, feature states, component states, or target directories, or that schedule system operations by inserting rows into sequence tables, can in many cases use immediate execution safely. However, custom actions that change the system directly, or call another system service, must be deferred to the time when the installation script is executed. See [Synchronous and Asynchronous Custom Actions](#) for more information about potential clashes between their custom actions and the main installation thread.

Because the installation script can be executed outside of the installation session in which it was written, the session may no longer exist during execution of the installation script. This means that the original session handle and property data set during the installation sequence is not available to a deferred execution custom action. Deferred custom actions that call dynamic-link libraries (DLLs) pass a handle which can only be used to obtain a very limited amount of information, as described in [Obtaining Context Information for Deferred Execution Custom Actions](#).

Note that deferred custom actions, including [rollback custom actions](#) and [commit custom actions](#), are the only types of actions that can run outside

the users security context.

## **See Also**

[Custom Action In-Script Execution Options](#)  
[Custom Action Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Executable Files

A custom action can launch an executable file that is stored on the user's machine or contained inside the installation package.

The following types of custom action launch an executable file.

<b>Custom action type</b>	<b>Description</b>
<a href="#">Custom Action Type 2</a>	EXE file stored in a Binary table stream.
<a href="#">Custom Action Type 18</a>	EXE file installed with a product.
<a href="#">Custom Action Type 50</a>	EXE file with a path specified by a property value.
<a href="#">Custom Action Type 34</a>	EXE file with a path referencing a directory.

See the section [Summary List of All Custom Action Types](#) for a summary of all types of custom actions and how they are encoded into the [CustomAction table](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Dynamic-Link Libraries

A custom action can call a function defined in a dynamic-link library (DLL) written in C or C++. The DLL can exist as a file installed during the current installation or as a temporary binary stream originating from the [Binary table](#) of the installation database.

Note that any called functions, including custom actions in DLLs, must specify the `__stdcall` calling convention. For example, to call `CustomAction` use the following.

```
#include <windows.h>
#include <msi.h>
#include <Msiquery.h>
#pragma comment(lib, "msi.lib")

UINT __stdcall CustomAction(MSIHANDLE hInstall)
```

For more information see, [Accessing the Current Installer Session from Inside a Custom Action](#)

The following types of custom actions call a dynamic-link library.

Custom action type	Description
<a href="#">Custom Action Type 1</a>	DLL file stored in a Binary table stream.
<a href="#">Custom Action Type 17</a>	DLL file installed with a product.

**Note** To use COM you need to call **CoInitializeEx** in the custom action. Do not quit if you find that the thread has already been initialized. For example, the thread is initialized in a per-machine installation but not in a per-user installation.

See [Summary List of All Custom Action Types](#) for a summary of all types of custom actions and how they are encoded into the [CustomAction table](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Scripts

A custom action can call functions that are written in VBScript or JScript. Windows Installer does not provide the script engine. Authors wishing to make use of a scripting language during installation must therefore ensure that the appropriate scripting engine is available.

The installer does not support JScript version 1.0.

A 64-bit custom action based on scripts must be explicitly marked as a 64-bit custom action by adding the `msidbCustomActionType64BitScript` bit to the custom actions numeric type in the `Type` column of the `CustomAction` table. For information see [64-Bit Custom Actions](#).

The following basic custom action types call functions written in script.

<b>Custom action type</b>	<b>Description</b>
<a href="#">Custom Action Type 5</a>	JScript file stored in a Binary table stream.
<a href="#">Custom Action Type 21</a>	JScript file that is installed with a product.
<a href="#">Custom Action Type 53</a>	JScript text specified by a property value.
<a href="#">Custom Action Type 37</a>	JScript text stored in the Target column of the CustomAction table.
<a href="#">Custom Action Type 6</a>	VBScript file stored in a <a href="#">Binary</a> table stream.
<a href="#">Custom Action Type 22</a>	VBScript file that is installed with a product.
<a href="#">Custom Action Type 54</a>	VBScript text specified by a property value.
<a href="#">Custom Action Type 38</a>	VBScript text stored in the Target column of the CustomAction table.

**Note** The installer runs script custom actions directly and does not use the Windows Script Host. The **WScript** object cannot be used inside a script custom action because this object is provided by the Windows Script Host. Objects in the Windows Script Host object model can only be used in custom actions if Windows Script Host is installed on the computer by creating new instances of the object, with a call to `CreateObject`, and providing the ProgId of the object (for example "WScript.Shell"). Depending on the type of script custom action, access to some objects and methods of the Windows Script Host object model may be denied for security reasons.

For more information, see [Summary List of All Custom Action Types](#) for a summary of all types of custom actions and how they are encoded into the [CustomAction](#) table.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Formatted Text Custom Actions

This type of custom action sets a property or an install directory from a formatted text string. The parameters in this string may be properties, environment variables, file paths, or the directory paths of components. This is useful for passing information to subsequent custom actions.

The following basic custom action types set a property or directory. For more information, see the particular Custom Action Type.

<b>Custom action type</b>	<b>Description</b>
<a href="#">Custom Action Type 51</a>	Sets a property with a formatted text string.
<a href="#">Custom Action Type 35</a>	Sets a directory with a formatted text string.

See the section [Summary List of All Custom Action Types](#) for a summary of all types of custom actions and how they are encoded into the [CustomAction table](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Error Message Custom Actions

This type of custom action displays a specified error message and returns failure, terminating the installation. The error message to be displayed can be supplied as a string or as an index into the [Error table](#).

Custom action type	Description
<a href="#">Custom Action Type 19</a>	Terminates the installation and displays an error message.

See the section [Summary List of All Custom Action Types](#) for a summary of how to encode each of the custom action types into the [CustomAction table](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Security

The installer runs custom actions with user privileges by default in order to limit the access of custom actions to the system. The installer may run custom actions with elevated privileges if a managed application is being installed or if the system policy has been specified for elevated privileges.

You should use the [MsiHiddenProperties](#) property and **msidbCustomActionTypeHideTarget** to prevent logging of sensitive information used by the custom action. For more information about **msidbCustomActionTypeHideTarget** see [Custom Action Hidden Target Option](#).

Clear the CustomActionData property after setting it to ensure that sensitive data is no longer available. Example code below is a snippet used by an immediate DLL custom action that is setting up data for use by a deferred custom action called "MyDeferredCA":

```
#include <windows.h>
#include <Msiquery.h>
#pragma comment(lib, "msi.lib")

UINT __stdcall MyImmediateCA(MSIHANDLE hInstall)
{
    // set up information for deferred custom action called
    const TCHAR szValue[] = TEXT("data");
    UINT uiStat = ERROR_INSTALL_FAILURE;
    if (ERROR_SUCCESS == Msi SetProperty(hInstall, TEXT("MyDeferredCA"), szValue))
    {
        uiStat = MsiDoAction(hInstall, TEXT("MyDeferredCA"));
    }

    // clear CustomActionData property
    if (ERROR_SUCCESS != Msi SetProperty(hInstall, TEXT("CustomActionData"), TEXT("")))
        return ERROR_INSTALL_FAILURE;
}

return (uiStat == ERROR_SUCCESS) ? uiStat : ERROR_INSTALL_FAILURE;
}
```

Note that only [deferred execution custom actions](#) can use the **msidbCustomActionTypeNoImpersonate** attribute. For more information see [Custom Action In-Script Execution Options](#).

If the **msidbCustomActionTypeNoImpersonate** bit is not set for a custom action, the installer runs the custom action with user-level privileges. For more information, see [Custom Action In-Script Execution Options](#).

If the **msidbCustomActionTypeNoImpersonate** bit is set and a managed application is being installed with administrator permission, the installer may run the custom action with elevated privileges. However, if a user attempts to install the managed application without administrator permission, the installer runs the application with user level privileges regardless of whether **msidbCustomActionTypeNoImpersonate** is set.

Note that a custom action may run with system privileges even when the **msidbCustomActionTypeNoImpersonate** bit is not set. This occurs if an administrator installs the application for all users on a server running the Terminal Server role service using Windows 2000 and the action is not marked with **msidbCustomActionTypeTSAware**. A custom action may also run with system privileges if the installation is invoked when there is no user context. For example, if there is no currently logged-on user during an installation invoked by Windows 2000 Application Deployment.

When creating your own custom actions, you should always author the custom action using secure methods. For more information, see [Guidelines for Securing Custom Actions](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Custom Actions

The following sections describe using custom actions.

- [Invoking Custom Actions](#)
- [Sequencing Custom Actions](#)
- [Obtaining Context Information for Deferred Execution Custom Actions](#)
- [Adding Custom Actions to the ProgressBar](#)
- [Debugging Custom Actions](#)
- [Determining UI Level from a Custom Action](#)
- [Uninstalling Custom Actions](#)
- [Returning Error Messages from Custom Actions](#)
- [Setting a restore point from a Custom Action](#)
- [Functions Not for Use in Custom Actions](#)
- [Changing the System State Using a Custom Action](#)
- [Accessing the Current Installer Session from Inside a Custom Action](#)
- [Accessing a Database or Session from Inside a Custom Action](#)
- [Using a Custom Action to Launch an Installed File at the End of the Installation](#)
- [Using a Custom Action to Create User Accounts on a Local Computer](#)
- [Using 64-bit Custom Actions](#)

For an overview of custom actions, see [About Custom Actions](#).

For more information about encoding custom actions into the `CustomAction` table, see [Custom Action Reference](#).

For a summary of custom actions and how they are encoded into the `CustomAction` table, see [Summary List of All Custom Action Types](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Invoking Custom Actions

Custom actions are invoked in the same manner as standard actions, either by explicit invocation or during the execution of a sequence table. There are two methods for calling actions:

- You call the specified action directly with the **MsiDoAction** function.
- A top-level action calls the sequence table containing the custom action. For more information about scheduling a custom action in a sequence table, see [Sequencing Custom Actions](#).

When the installer obtains an action name from the **MsiDoAction** function, or from a sequence table, it first searches for a standard action of that name. If it cannot find the standard action, then the installer queries the [CustomAction table](#) to check if the specified action is a custom action. If the specified action is not a custom action, then the installer queries the [Dialog table](#) for a dialog box.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Sequencing Custom Actions

Custom actions are scheduled in sequence tables in the same way as standard actions.

## ► To schedule a custom action in a sequence table

1. Enter the custom action name (which is the primary key of the [CustomAction](#)) table into the Action column of the [Sequence](#) table.
2. Enter the custom action's sequence relative to the other actions in the table into the Sequence column of the Sequence table. For more information about sequence tables, see [Using a Sequence Table](#).
3. To conditionally skip the action, enter a conditional expression into the Condition column of the [Sequence](#) table. The installer skips this action if the expression evaluates to FALSE.

As in the case of standard actions, custom actions that are scheduled in the [InstallUISequence](#) or [AdminUISequence](#) run only if the internal user interface is set to the full level. The UI level is set by using the [MsiSetInternalUI](#) function.

Standard and custom actions scheduled in the [InstallExecuteSequence](#), [AdminExecuteSequence](#), or [AdvtExecuteSequence](#) tables do not make system changes. Instead the installer queues up execution records in a script for subsequent execution during the install service. If there is no install service, then the actions scheduled in these tables are run in the same context as the UI sequence.

If the installer server is not registered, the custom actions are executed on the client side. If the server is registered and using the full UI mode, then the custom actions are run on the server side.

If using full UI with the server, the initial actions prior to the [InstallValidate](#) action are run on the client to allow full interaction. Execution is then switched to the server which repeats those actions and runs the script execution actions. This is followed by a return to the client for the final

actions.

Note that if a product is removed by setting its top feature to absent, the **REMOVE** property may not equal ALL until after the InstallValidate action. This means that any custom action that depends on REMOVE=ALL must be sequenced after the InstallValidate action. A custom action may check REMOVE to determine whether a product has been set to be completely uninstalled.

Custom actions that reference an installed file as their source, such as Custom Action Type 17 (DLL), Custom Action Type 18 (EXE), Custom Action Type 21 (JScript), and Custom Action Type 22 (VBScript), must adhere to the following sequencing restrictions.

- The custom action must be sequenced after the **CostFinalize** action so that the path to the referenced file can be resolved.
- If the source file is not already installed on the computer, deferred (in-script) custom actions must be sequenced after the **InstallFiles**.
- If the source file is not already installed on the computer, nondeferred custom actions must be sequenced after the **InstallInitialize** action.

The following sequencing restrictions apply to custom actions that change or update a Windows Installer package.

- If the custom action changes the package, such as by adding rows to a table, the action must be sequenced before the InstallInitialize action.
- If the custom action makes changes that would affect costing, then it should be sequenced before the CostInitialize action.
- If the custom action changes the installation state of features or components, it must be sequenced before the InstallValidate action.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Obtaining Context Information for Deferred Execution Custom Actions

Because installation script can be executed outside of the installation session in which it was written, the session may no longer exist during execution of the installation script. In this case, the original session handle and properties set during the installation sequence are not available to a deferred execution custom action. Any functions that require a session handle are restricted to a few methods that can retrieve context information, or else properties that are needed during script execution must be written into the installation script. For example, deferred custom actions that call dynamic-link libraries (DLLs) pass a handle which can only be used to obtain a very limited amount of information. Functions that do not require a session handle can be accessed from a deferred custom action.

Deferred execution custom actions are restricted to calling only the following functions requiring a handle.

Function	Description
<b>MsiGetProperty</b>	Supports a limited set of properties when used with <a href="#">deferred execution custom actions</a> : the CustomActionData property, <b>ProductCode</b> property, and <b>UserSID</b> property. <a href="#">Commit custom actions</a> cannot use the <b>MsiGetProperty</b> function to obtain the <b>ProductCode</b> property. Commit custom actions can use the CustomActionData property to obtain the product code.
<b>MsiFormatRecord</b>	Supports a limited set of properties when used with <a href="#">deferred execution custom actions</a> : the CustomActionData and ProductCode properties.
<b>MsiGetMode</b>	When called from <a href="#">deferred execution custom actions</a> , <a href="#">commit custom actions</a> , or <a href="#">rollback custom actions</a> , <b>MsiGetMode</b> returns True or False when requested to check the mode parameters

	MSIRUNMODE_SCHEDULED, MSIRUNMODE_COMMIT, or MSIRUNMODE_ROLLBACK. Requests to check any other run mode parameters from a deferred, commit, or rollback custom action returns False.
<b>MsiGetLanguage</b>	The numeric language ID for the current product. Commit custom actions cannot use the <b>MsiGetLanguage</b> function. Commit custom actions can use the CustomActionData property to get the numeric language ID.
<b>MsiProcessMessage</b>	Processes error or progress messages from the custom action.

A custom action that is written in JScript or VBScript requires the install **Session** object. This is of the type **Session Object** and the installer attaches it to the script with the name "Session". Because the **Session** object may not exist during an installation rollback, a deferred custom action written in script must use one of the following methods or properties of the **Session** object to retrieve its context.

Name	Description
<b>Mode Property</b>	Returns True for MSIRUNMODE_SCHEDULED only.
<b>Property Property (Session Object)</b>	Returns the CustomActionData property, <b>ProductCode</b> property, or <b>UserSID</b> property.
<b>Language Property (Session Object)</b>	Returns numeric language ID of the installation session.
<b>Message Method</b>	Called to handle errors and progress.
<b>Installer</b>	Returns the parent object, which is used for non-session

**Property**

functions such as registry access and installer configuration management.

Property values that are set at the time the installation sequence is processed into script may be unavailable at the time of script execution. Only the following limited set of properties is always accessible to custom actions during script execution.

<b>Property name</b>	<b>Description</b>
CustomActionData	Value at time custom action is processed in sequence table. The CustomActionData property is only available to deferred execution custom actions. Immediate custom actions do not have access to this property.
<b>ProductCode</b>	Unique code for the product, a <a href="#">GUID</a> string.
<b>UserSID</b>	Set by the installer to the user's security identifier (SID).

If other property data is required by the deferred execution custom action, then their values must be stored in the installation script. This can be done by using a second custom action.

► **To write the value of a property into the installation script for use during a deferred execution custom action**

1. Insert a small custom action into the installation sequence that sets the property of interest to a property having the same name as the deferred execution custom action. For example, if the primary key for the deferred execution custom action is "MyAction" set a property named "MyAction" to the property X which you need to retrieve. You must set the "MyAction" property in the installation sequence before the "MyAction" custom action. Although any type of custom action can set the context data, the simplest method is to use a property assignment custom action (for example [Custom Action Type 51](#)).
2. At the time when the installation sequence is processed, the

installer will write the value of property X into the execution script as the value of the property CustomActionData.

## **See Also**

[About Properties](#)

[Using Properties](#)

[Property Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Adding Custom Actions to the ProgressBar

[Custom Actions](#) can add time and progress information to a [ProgressBar](#) control. For more information about creating an action display dialog box having a ProgressBar, see [Authoring a ProgressBar Control](#).

Note that two custom actions must be added to the Windows Installer package to accurately report time and progress information to the ProgressBar. One custom action must be a deferred custom action. This custom action should complete your custom installation and send the amounts of individual increments to the ProgressBar control when the installer runs the installation script. The second custom action must be an immediate execution custom action that informs the ProgressBar how many ticks to add to the total count during the acquisition and script generation phase of the installation.

## ► To add a custom action to the ProgressBar

1. Decide how the custom action will describe its progress. For example, a custom action that installs registry keys could display a progress message and update the ProgressBar each time the installer writes one registry key.
2. Each update by the custom action changes the length of the ProgressBar by a constant increment. Specify or calculate the number of ticks in each increment. Typically a change in ProgressBar length of one tick corresponds to the installation of one byte. For example, if the installer installs approximately 10000 bytes when it writes one registry key, you can specify that there are 10000 ticks in an increment.
3. Specify or calculate the total number of ticks the custom action adds to the length of the ProgressBar. The number of ticks added by the custom action is usually calculated as: (tick increment) x (number of items). For example, if the custom action writes 10

registry keys, the installer installs approximately 100000 bytes and the installer therefore must increase the estimate of the final total length of the ProgressBar by 100000 ticks.

**Note** To calculate this dynamically, the custom action must contain a section that is immediately executed during script generation. The amount of ticks reported by your deferred execution custom action must be equal to the number of ticks added to the total tick count by the immediate execution action. If this is not the case, the time remaining as reported by the TimeRemaining text control will be inaccurate.

4. Separate your custom action into two sections of code: a section that runs during the script generation phase and a section that runs during the execution phase of the installation. You can do this using two files or you can use one file by conditioning on the run mode of the installer. The following sample uses one file and checks the installation state. Sections of the sample are conditioned to run depending on whether the installer is in the execution or script generation phase of the installation.
5. The section that runs during script generation should increase the estimate of the final total length of the [ProgressBar](#) by the total number of ticks in the custom action. This is done by sending a **ProgressAddition** progress message.
6. The section that runs during the execution phase of installation should set up message text and templates to inform the user about what the custom action is doing and to direct the installer on updating the ProgressBar control. For example, inform the installer to move the ProgressBar forward one increment and send an explicit progress message with each update. There is usually a loop in this section if the custom action is installing something. With each pass through this loop, the installer can install one reference item such as a registry key and update the

## ProgressBar control

7. Add an immediate execution custom action to your Windows Installer package. This custom action informs the ProgressBar how much to advance during the acquisition and script generation phases of the installation. For the following sample, the source is the DLL created by compiling the sample code and the target is the entry point, CAProgress.
8. Add a deferred execution custom action to your Windows Installer package. This custom action completes the steps of the actual installation and informs the ProgressBar how much to advance the bar at the time when the installer runs the installation script. For the following sample, the source is the DLL created by compiling the sample code and the target is the entry point, CAProgress.
9. Schedule both custom actions between [InstallInitialize](#) and [InstallFinalize](#) in the [InstallExecuteSequence](#) table. The deferred custom action should be scheduled immediately after the immediate execution custom action. The installer will not run the deferred custom action until the script is executed.

The following sample shows how a custom action can be added to the ProgressBar. The source of both custom actions is the DLL created by compiling the sample code and the target of both custom actions is the entry point, CAProgress. This sample does not make any actual changes to the system, but operates the ProgressBar as if installing 10 reference items that are each approximately 10,000 bytes in size. The installer updates the message and ProgressBar each time it installs a reference item.

```
#include <windows.h>
#include <msiquery.h>
#pragma comment(lib, "msi.lib")

// Specify or calculate the number of ticks in an increment
// to the ProgressBar
```

```

const UINT iTickIncrement = 10000;

// Specify or calculate the total number of ticks the custom
// action adds to the length of the ProgressBar
const UINT iNumberItems = 10;
const UINT iTotalTicks = iTickIncrement * iNumberItems;

UINT __stdcall CAProgress(MSIHANDLE hInstall)
{
    // Tell the installer to check the installation state and
    // the code needed during the rollback, acquisition, or
    // execution phases of the installation.

    if (MsiGetMode(hInstall,MSIRUNMODE_SCHEDULED) == TRUE)
    {
        PMSIHANDLE hActionRec = MsiCreateRecord(3);
        PMSIHANDLE hProgressRec = MsiCreateRecord(3);

        // Installer is executing the installation script.
        // record specifying appropriate templates and text
        // messages that will inform the user about what the
        // action is doing. Tell the installer to use this
        // text in progress messages.

        MsiRecordSetString(hActionRec, 1, TEXT("MyCustomActi
        MsiRecordSetString(hActionRec, 2, TEXT("Incrementir
        MsiRecordSetString(hActionRec, 3, TEXT("Incrementir
        UINT iResult = MsiProcessMessage(hInstall, INSTALLM
            if ((iResult == IDCANCEL))
                return ERROR_INSTALL_USEREXIT;

        // Tell the installer to use explicit progress mess
        MsiRecordSetInteger(hProgressRec, 1, 1);
        MsiRecordSetInteger(hProgressRec, 2, 1);
        MsiRecordSetInteger(hProgressRec, 3, 0);
        iResult = MsiProcessMessage(hInstall, INSTALLMESSAG
            if ((iResult == IDCANCEL))
                return ERROR_INSTALL_USEREXIT;

        //Specify that an update of the progress bar's posi
        //this case means to move it forward by one increme
        MsiRecordSetInteger(hProgressRec, 1, 2);

```

```

MsiRecordSetInteger(hProgressRec, 2, iTickIncrement);
MsiRecordSetInteger(hProgressRec, 3, 0);

// The following loop sets up the record needed by
// messages and tells the installer to send a message
// the progress bar.

MsiRecordSetInteger(hActionRec, 2, iTotalTicks);

for( int i = 0; i < iTotalTicks; i+=iTickIncrement)
{
    MsiRecordSetInteger(hActionRec, 1,
        i);
    iResult = MsiProcessMessage(hInstall, INSTALLMESSAGE_CUSTOM);
    if ((iResult == IDCANCEL))
        return ERROR_INSTALL_USEREXIT;

    iResult = MsiProcessMessage(hInstall, INSTALLMESSAGE_CUSTOM);
    if ((iResult == IDCANCEL))
        return ERROR_INSTALL_USEREXIT;

    //A real custom action would have done something
    //of the installation. For this sample, we will
    //10 registry keys.
    Sleep(1000);
}

return ERROR_SUCCESS;
}
else
{
    // Installer is generating the installation
    // custom action.

    // Tell the installer to increase the value
    // length of the progress bar by the total
    // the custom action.
    PMSIHANDLE hProgressRec = MsiCreateRecord(2);

    MsiRecordSetInteger(hProgressRec, 1, 3);
    MsiRecordSetInteger(hProgressRec, 2, 0);
    UINT iResult = MsiProcessMessage(hInstall,

```

```
        if ((iResult == IDCANCEL))
            return ERROR_INSTALL_USEREXIT;
        return ERROR_SUCCESS;
    }
}
```

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Debugging Custom Actions

You can debug custom actions that are based on [dynamic-link libraries](#) by using [Debugging Tools for Windows](#). It is not possible to use dynamic debugging with custom actions based on [executable files](#) or [scripts](#).

The techniques described in this section can help you debug Windows Installer custom actions. See the Driver Development Tools section of the Windows Driver Kit for information about [Debugging Tools for Windows](#).

Windows Installer uses the MsiBreak environment variable to determine which custom action is to be debugged. If you have access to the custom action's source code, you may be able to use debugging without MsiBreak. To start debugging without MsiBreak, put a temporary message box at the beginning of the action's code. When the message box appears during the installation, attach the debugger to the process owning the message box. You can then set any necessary breakpoints and dismiss the message box to resume execution. It is not possible to debug the earlier portions of the custom action by this method.

To use the MsiBreak environment variable to debug the custom action, set MsiBreak to the custom action's name in the [CustomAction table](#). MsiBreak can be either a system or a user environment variable. If the variable is set as a system variable, a restart of the system may be needed when the value is changed to detect the new value.

To use the MsiBreak environment variable to debug an embedded user interface, set the value of MsiBreak to MsiEmbeddedUI.

Windows Installer only checks the MsiBreak environment variable if the user is an Administrator. The installer ignores the value of MsiBreak if the user is not an Administrator, even if this is a [managed application](#).

If you are debugging a custom action that runs with elevated (system) privileges in the execution sequence, attach the debugger to the Windows Installer service. When debugging a custom action that runs with impersonated privileges in the execution sequence, the system prompts with a dialog box that indicates which process should be debugged. The user is prompted with a dialog box indicating which process to debug. For more information about elevated custom actions, see [Custom Action Security](#).

Once the debugger has been attached to the correct process, the installer triggers a debugger breakpoint immediately before calling the entry point of the DLL. At the breakpoint, your DLL is already loaded into the process and the entry point address determined. If your custom action DLL could not be loaded or the custom action entry point did not exist, no breakpoint is triggered. Because the breakpoint is triggered before calling the DLL function, once the breakpoint has been triggered you should use your debugger to step forward until your custom action entry point is called. Alternately, you can set a breakpoint anywhere in your custom action and resume normal execution.

The Windows Installer executes DLLs not stored in the [Binary table](#) directly from the DLL location. The installer does not know the original name of a DLL stored in the Binary table and runs the DLL custom action under a temporary file name. The form of the temporary file name is MSI?????.TMP. On Windows 2000 or Windows XP this temporary file is stored in a secure location, commonly <WindowFolder>\Installer.

Note that many DLLs created for debugging contain the name and path of the corresponding PDB file as part of the DLL itself. When debugging this type of DLL on a system where the PDB can be found at the location stored in the DLL, symbols may be loaded automatically by the debugger tool. In situations where the PDB cannot be found at the stored location, where the debugger does not support loading symbols from the stored location, or where the DLL was not built with debugging information, you may need to place your symbol files in the folder with the temporary DLL file.

The installer adds debugging information for custom action scripts to the installation log file.

```
There is a problem with this Windows Installer package. A script  
required for this install to complete could not be run. Contact  
support personnel or package vendor. {Custom action [2] script  
[3], [4]: [5] Line [6], Column [7], [8] }
```

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Determining UI Level from a Custom Action

A custom action in a UI sequence table or an external executable file may need the current user interface level of the installation. For example, a custom action that has a dialog box should only display the dialog when the user interface level is Full UI or Reduced UI, it should not display the dialog if the user interface level is Basic UI or None. You should use the **UILevel** property to determine the current user interface level. You cannot call **MsiSetInternalUI** from a custom action and it is not possible to change the UI level property from within a custom action.

It is recommended that custom actions not use the UI level as a condition for sending error messages to the installer because this can interfere with logging and external messages.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Uninstalling Custom Actions

Developers of installation packages that make use of custom actions must determine in each case how to condition the custom action to work correctly when the package is uninstalled. It is up to the package author to decide whether to condition the custom action to run on uninstall, not to run on uninstall, or to run an alternate custom action on uninstall.

## See Also

[Conditional Statement Syntax](#)

[Using Properties in Conditional Statements](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Returning Error Messages from Custom Actions

This section describes how to send messages from custom actions that actually perform a part of the installation by calling a dynamic link library or script. Note that [Custom Action Type 19](#) only sends a specified error message, returns failure, and then terminates the installation. Custom Action Type 19 does not perform any part of the installation.

To send an error message from a custom action that uses a [dynamic-link library](#) (DLL), have the custom action call **MsiProcessMessage**. Note that custom actions launched by a [DoAction ControlEvent](#) can send messages with the **Message** method but cannot send a message with **MsiProcessMessage**. On systems earlier than Windows Server 2003, custom actions launched by a DoAction ControlEvent cannot send messages with **MsiProcessMessage** or **Message** method. For more information, see [Sending Messages to Windows Installer Using MsiProcessMessage](#).

## ► To display an error message from within a custom action using a DLL

1. The custom action should call **MsiProcessMessage** and pass in the parameters *hInstall*, *eMessageType*, and *hRecord*. The handle to the installation, Custom Action Type 19, may be provided to the custom action as described in [Accessing the Current Installer Session from Inside a Custom Action](#) or from **MsiOpenProduct** or **MsiOpenPackage**.
2. The parameter *eMessageType* should specify one of the message types as listed in **MsiProcessMessage**.
3. The *hRecord* parameter of the **MsiProcessMessage** function depends upon the message type. See [Sending Messages to Windows Installer Using MsiProcessMessage](#). If the message contains formatted data, enter the message into the **Error** table using the formatting described in [Formatted](#).

To send an error message from a custom action that uses [Scripts](#), the custom action may call the [Message](#) method of the [Session](#) object.

► **To display an error message from within a custom action using script**

1. The custom action should call the [Message](#) method of the [Session](#) object and pass in the parameters *kind* and *record*.
2. The parameter *kind* should specify one of the message types listed in the [Message](#) method.
3. The *record* parameter of the [Message](#) method depends upon the message type. If the message contains formatted data, enter the message into the [Error](#) table using the formatting described in [Formatted](#).

Custom actions using [Executable Files](#) cannot send a message by calling [MsiProcessMessage](#) or the [Message](#) method because they cannot get a handle to the installation.

## See Also

[Custom Action Return Values](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Setting a Restore Point from a Custom Action

Custom actions must not call the **SRSetRestorePoint** function because this may result in a restore entry point being written into the middle of a Windows Installer installation.

For more information, see [System Restore Points and the Windows Installer](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Functions Not for Use in Custom Actions

The following [Database Functions](#) must never be called from a custom action.

- [MsiConfigureProduct](#)
- [MsiConfigureProductEx](#)
- [MsiCreateTransformSummaryInfo](#)
- [MsiDatabaseApplyTransform](#)
- [MsiDatabaseCommit](#)
- [MsiDatabaseExport](#)
- [MsiDatabaseGenerateTransform](#)
- [MsiDatabaseImport](#)
- [MsiDatabaseMerge](#)
- [MsiEnableLog](#)
- [MsiEnableUIPreview](#)
- [MsiGetDatabaseState](#)
- [MsiOpenDatabase](#)
- [MsiPreviewBillboard](#)
- [MsiPreviewDialog](#)
- [MsiReinstallProduct](#)
- [MsiSetExternalUI](#)
- [MsiSetExternalUIRecord](#)
- [MsiSetInternalUI](#)

The following [Installer Functions](#) must never be called from a custom action.

- [MsiApplyPatch](#)
- [MsiCollectUserInfo](#)

- **MsiConfigureFeature**
- **MsiConfigureProduct**
- **MsiConfigureProductEx**
- **MsiEnableLog**
- **MsiGetFeatureInfo**
- **MsiGetProductCode**
- **MsiGetProductProperty**
- **MsiInstallMissingComponent**
- **MsiInstallMissingFile**
- **MsiInstallProduct**
- **MsiOpenPackage**
- **MsiOpenProduct**
- **MsiReinstallFeature**
- **MsiReinstallProduct**
- **MsiSetExternalUI**
- **MsiSetInternalUI**
- **MsiUseFeature**
- **MsiUseFeatureEx**
- **MsiVerifyPackage**

The following **Installer Functions** must never be called from a custom action if doing so starts another installation. They may be called from a custom action that does not start another installation.

- **MsiProvideComponent**
- **MsiProvideQualifiedComponent**
- **MsiProvideQualifiedComponentEx**

A custom action should never spawn a new thread that uses Windows Installer functions to change the feature state, component state, or to send messages from a Control Event. Attempting to do this causes the installation to fail.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Changing the System State Using a Custom Action

Custom actions intended to change the system state must be deferred execution custom actions. Custom actions that set properties, feature states, component states, or target directories, or schedule system operations by inserting rows into sequence tables can use immediate execution safely. However, custom actions that change the system directly or call another system service must be deferred to the time when the installation script is executed. For more information, see [Deferred Execution Custom Actions](#).

You should not attempt to use an immediate execution custom action to change the system state, because every custom action that changes the state needs to have a corresponding rollback custom action to undo the system state change on an installation rollback. All rollback custom actions are also deferred custom actions and must precede the action they undo. For more information, see [Rollback Custom Actions](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Accessing the Current Installer Session from Inside a Custom Action

Nondeferred custom actions that call [dynamic-link libraries](#) or [scripts](#) may access a running installation to query or modify the attributes of the current installation session. Only one **Session** object can exist for each process, and custom action scripts must not attempt to create another session.

Custom actions can only add, modify, or remove temporary rows, columns, or tables from a database. Custom actions cannot modify persistent data in a database, for example, data that is a part of the database stored on disk.

## Dynamic-Link Libraries

To access a running installation, custom actions that call dynamic-link libraries (DLL) are passed a handle of the type `MSIHANDLE` for the current session as the only argument to the DLL entry point named in the `Target` column of the [CustomAction Table](#). Because the installer provides this handle, the custom action should not close it, for example, to receive the handle *hInstall* from the installer, the custom action function is declared as follows.

```
UINT __stdcall CustomAction(MSIHANDLE hInstall)
```

For read-only access to the current database obtain the database handle by calling [MsiGetActiveDatabase](#). For more information, see [Obtaining a Database Handle](#).

## Scripts

Custom actions written in VBScript or JScript can access the current installation session by using the [Session Object](#). The installer creates a **Session** object named "Session" that references the current installation. For read-only access to the current database use the [Database](#) property of the **Session** object.

Because a script is run from the context of the **Session** object, it is not always necessary to fully qualify properties and methods. In the following example, when using VBScript, the Me reference can replace the **Session** object, for example, the following three lines are equivalent.

```
Session.SetInstallLevel 1
```

```
Me.SetInstallLevel 1
```

```
SetInstallLevel 1
```

### [Executable Files](#)

You cannot access the current installer session from custom actions that call executable files launched with a command-line, for example, [Custom Action Type 2](#) and [Custom Action Type 18](#).

### [Deferred Execution Custom Actions](#)

You cannot access the current installer session or all property data from a deferred execution custom action. For more information, see [Obtaining Context Information for Deferred Execution Custom Actions](#).

## **See Also**

[Accessing a Database or Session from Inside a Custom Action](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Accessing a Database or Session from Inside a Custom Action

You cannot access an installer session from a custom action that is not the current installation session. Custom actions are limited to working with only the active database of the session and no other databases. The following Windows Installer [Database Functions](#) must not be called from a custom action, because they require a handle to a database that is not the database of the current installation session:

[MsiDatabaseMergeMsiCreateTransformSummaryInfo](#)

[MsiDatabaseApplyTransform](#)

[MsiDatabaseCommit](#)

[MsiDatabaseExport](#)

[MsiDatabaseGenerateTransform](#)

[MsiDatabaseImport](#)

[MsiEnableUIPreview](#)

[MsiGetDatabaseState](#)

## See Also

[Accessing the Current Installer Session from Inside a Custom Action](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using a Custom Action to Launch an Installed File at the End of the Installation

The following example illustrates how to launch an HTML file at the end of an installation. The Installer installs the component that contains the file, and then publishes a control event at the end of the installation to run a custom action that opens the file. This approach may be used to launch a help tutorial at the end of the first installation of an application.

The sample must meet the following specifications.

- The Installer executes the custom action only if the *full UI* level is used to install an application.
- The Installer executes the custom action only if the component that contains the HTML file is installed to run locally on the computer.
- The custom action only runs on the first installation of the application.
- The installation does not fail if the custom action fails.

The sample includes a hypothetical component named Tutorial that controls at least one resource, a file named tutorial.htm. The identifier for this file in the File column of the File table is Tutorial. The following discussion assumes that you have already created the resources required by Tutorial, and have made all the necessary entries in the [Feature](#), [Component](#), [File](#), [Directory](#), and [FeatureComponents](#) tables to install this component. For more information, see [An Installation Example](#).

The following topics contain information about how to create required custom actions and add them to an installation package.

- [Authoring the Launch Custom Action](#)
- [Adding Launch to the CustomAction and Binary Tables](#)
- [Adding a Control Event at the End of the Installation to Run Launch](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring the Launch Custom Action

The source code for a sample custom action named Launch, which meets the sample specifications, is provided by the Windows Installer SDK as the file Tutorial.cpp. This custom action makes use of [MsiFormatRecord](#) to format a string containing properties. The property [#FileKey] resolves to the full path of the HTML file. Use the source file to create the file Tutorial.dll. The entry point to this DLL is LaunchTutorial.

The sample custom action Launch calls a DLL written in C++ and is generated from a temporary binary stream. Custom actions of this type include the base type constants `msidbCustomActionTypeDll` and `msidbCustomActionTypeBinaryData`, which give a base numeric type equal to 1. See [Custom Action Type 1](#). Because the specifications require that the installation continue if the custom action fails, Launch also includes the optional constant `msidbCustomActionTypeContinue`, which is 64. See [Custom Action Return Processing Options](#). The total numeric type of Launch is 65.

Continue to [Adding Launch to the CustomAction and Binary Tables](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Adding Launch to the CustomAction and Binary Tables

Add a record to the [CustomAction table](#) for the Launch custom action. Enter the custom action's name in the Action column of the CustomAction table. Enter the total numeric type for Launch, 65, into the Type column of the Custom action table. The Source column of the CustomAction table specifies a key into the record of the [Binary Table](#) that contains the binary data for the DLL. Enter Tutorial.dll into the Source column of the CustomAction table. The entry point specified in the Target field of the CustomAction table must match that exported from the DLL. Enter LaunchTutorial into the Target column of the CustomAction table.

## CustomAction Table

Action	Type	Source	Target
Launch	65	Tutorial.dll	LaunchTutorial

Add the Tutorial.dll you created from Tutorial.cpp as a binary stream to the Binary table.

## Binary Table

Name	Data
Tutorial.dll	{binary data added for DLL}

Continue to [Adding a Control Event at the End of the Installation to Run Launch](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Adding a Control Event at the End of the Installation to Run Launch

The installer runs the sample's installation wizard sequence only if the *full UI* level is used to install the application. The last dialog box of the sample dialog sequence is an [Exit Dialog](#) named ExitDialog. When a user interacts with the OK button on ExitDialog, this first publishes an [EndDialog ControlEvent](#) that returns control to the installer. The control then publishes a [DoAction ControlEvent](#) that runs the Launch custom action. Each control event requires a record in the [ControlEvent table](#). See [ControlEvent Overview](#).

## ControlEvent Table

Dialog	Control_	Event	Argument	Condition	Ordering
ExitDialog	OK	EndDialog	Return	1	1
ExitDialog	OK	DoAction	Launch	NOT Installed AND \$Tutorial=3	2

The condition on the DoAction control ensures the custom action only runs during the first installation of the application and that it is being installed locally. The phrase \$Tutorial=3 means the action state of the Tutorial component is set to local. See [Conditional Statement Syntax](#).

This completes the sample.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using a Custom Action to Create User Accounts on a Local Computer

This sample demonstrates how to use custom actions to create user accounts on a local computer when installing a component. Removal of a component removes the local user accounts created by the custom action. Several custom actions are demonstrated including [Deferred Execution Custom Actions](#) and [Rollback Custom Actions](#).

The sample meets the following specifications.

- The installation creates user accounts only if running Windows 2000.
- The installation creates user accounts only if the component is being installed to run locally. This precludes creating user accounts during the repair or reinstallation of the component.
- The Installer removes the accounts when the component is removed.
- User account information is read from a custom table in the installation database and is not hard-coded into the custom action code.
- Because the creation or removal of user accounts requires elevated privileges, some of the custom actions must be capable of making changes to the system that require elevated privileges. These custom actions must be deferred custom actions that run when in the execution script.
- Each account has a rollback custom action to ensure the account is removed on rollback of the component installation. This does not include the rollback of an account deletion during the removal of a component.
- Custom actions send `ActionData` messages for each account that is created or removed. This does not include providing progress messages for the `ProgressBar`.

- Custom actions report an error if an account cannot be created.
- The password for the account is obtained through the user interaction with the user interface, or in the case of an installation at the Basic UI or None [User Interface Levels](#), as a property passed on the command line.
- Sensitive data is hidden from the log file.

The sample includes a hypothetical component named TestAccount. The discussion in the following sections assumes that you have already created the resources required by TestAccount and have authored the [Feature](#), [Component](#), [File](#), [Directory](#), and [FeatureComponents](#) tables in the sample database required to install this component. For more information, see [An Installation Example](#).

The following topics contain information about how to create required custom actions and add them to an installation package.

- [Authoring the Custom Actions](#)
- [Adding a Custom CustomUserAccounts Table](#)
- [Authoring the CustomAction Table](#)
- [Authoring the ActionText and Error Tables](#)
- [Authoring the InstallExecuteSequence Table](#)
- [Authoring the User Interface for Password Input](#)
- [Securing the installation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring the Custom Actions

The table below lists the five custom actions used to meet the sample specifications: ProcessAccounts, UninstallAccounts, CreateAccounts, RemoveAccounts, and RollbackAccounts. All of these custom actions are in [dynamic-link libraries](#) stored in the [Binary Table](#). The C++ source code for the dynamic-link libraries containing the sample custom actions are provided in the Windows Installer SDK. ProcessAccounts and UninstallAccounts are in the file Process.cpp. CreateAccount is in the file Create.cpp. RemoveAccount and RollbackAccount are in the file Remove.cpp. These source files can be used to create the files Process.dll, Create.dll, and Remove.dll.

Because the creation or removal of a user account requires elevated privileges, [deferred execution custom actions](#) that run in the context of the system must be used to create, remove, or roll back user accounts. The immediate execution custom actions, ProcessAccounts and UninstallAccounts, generate the deferred custom actions that create, remove, or rollback user accounts: CreateAccount, RemoveAccount, and RollbackAccount.

Because deferred custom actions cannot read information in database tables, ProcessAccounts and UninstallUserAccounts must set a CustomActionData property to pass the information in the UserAccounts table to the deferred custom actions as described in [Obtaining Context Information for Deferred Execution Custom Actions](#). When the installer runs the execution script, the deferred custom actions handle user accounts according to the information in the CustomActionData property.

Because all the custom actions are in dynamic-link libraries stored in the Binary table, they all include the constants msidbCustomActionTypeDll and msidbCustomActionTypeBinaryData in their base numeric type. ProcessAccounts and UninstallAccounts are examples of pure [Custom Action Type 1](#). For information on other custom action types see the [Summary List of All Custom Action Types](#).

CreateAccount and RemoveAccount are [deferred execution custom actions](#) that do not allow services to impersonate particular users. These custom actions include the constants msidbCustomActionTypeInScript and msidbCustomActionTypeNoImpersonate to specify these [custom](#)

[action in-script execution options](#).

RollbackAccount is a [rollback custom action](#) that only removes user accounts during a [rollback installation](#). RollbackAccount includes the constants msidbCustomActionTypeInScript and msidbCustomActionTypeRollback to specify these [custom action in-script execution options](#).

These custom actions may handle sensitive data such as user passwords, which should not be written to the log file. The deferred custom actions should therefore include msidbCustomActionTypeHideTarget in the custom action type. The names of the deferred custom actions also need to be added to the [MsiHiddenProperties](#) property list in the [Property table](#) because of the way immediate custom actions pass data to deferred custom actions using the CustomActionData property.

Custom action	DLL entry point	Custom action type
ProcessAccounts	ProcessUserAccounts in Process.dll.	msidbCustomActionTypeDll + msidbCustomActionTypeBinaryDa
UninstallAccounts	UninstallUserAccounts in Process.dll.	msidbCustomActionTypeDll + msidbCustomActionTypeBinaryDa
CreateAccount	CreateUserAccount in Create.dll.	msidbCustomActionTypeDll + msidbCustomActionTypeBinaryDa msidbCustomActionTypeInScript + msidbCustomActionTypeNoImpers + msidbCustomActionTypeHideTa 11265.
RemoveAccount	RemoveUserAccount in Remove.dll.	msidbCustomActionTypeDll + msidbCustomActionTypeBinaryDa msidbCustomActionTypeInScript + msidbCustomActionTypeNoImpers + msidbCustomActionTypeHideTa 11265.
RollbackAccount	RemoveUserAccount in Remove.dll.	msidbCustomActionTypeDll + msidbCustomActionTypeBinaryDa msidbCustomActionTypeInScript + msidbCustomActionTypeRollback

	msidbCustomActionTypeHideTarg 9473.
--	--

---

Continue to [Authoring the CustomAction Table](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring the CustomAction Table

Enter records for the five sample custom actions created in the previous section to the [CustomAction Table](#). For more information on how to populate the CustomAction table for this type of custom action see [Custom Action Type 1](#).

## CustomAction Table

Action	Type	Source	Target
ProcessAccounts	1	Process.dll	ProcessUserAccounts
UninstallAccounts	1	Process.dll	UninstallUserAccounts
CreateAccount	11265	Create.dll	CreateUserAccount
RemoveAccount	11265	Remove.dll	RemoveUserAccount
RollbackAccount	9473	Remove.dll	RemoveUserAccount

The C++ source code for the dynamic-link libraries are provided in the Windows Installer SDK. Use Process.cpp to create the file Process.dll. Use Create.cpp to create the file Create.dll. Use Remove.cpp to create Remove.dll. Add these dynamic-link library files to the Binary table.

## Binary Table

Name	Data
Process.dll	{binary data}
Create.dll	{binary data}
Remove.dll	{binary data}

Continue to [Adding a Custom CustomUserAccounts Table](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Adding a Custom CustomUserAccounts Table

A specification of the sample is that user account information be read from a custom table in the installation database and not hard-coded into the custom action.

Add a custom table to the sample installation database named CustomUserAccounts to hold user account information. See [Examples of Database Queries Using SQL and Script](#) for an example of how to add a custom table. Use the following schema for the CustomUserAccounts table. See [Column Definition Format](#) for an explanation of the column types.

Column	Type	Key	Nullable	Description
UserName	s72	Y	N	Name of user account being created.
Password	s72		N	Name of property containing the password for the account. This is a <a href="#">public property</a> set on the command line or through an <a href="#">edit control</a> in the user interface. This edit control should have the <a href="#">Password Control Attribute</a> .
Attributes	i4		Y	Attributes for account. These are defined as the <b>DWORD</b> values for the usri1_flags member of the USER_INFO_1 structure.

After the CustomUserAccounts table has been added to the database you may edit this table using Orca, a table editor provided with the Windows Installer SDK, or another editor. Enter the following record in the CustomUserAccounts table to create a password secured user account for a user called TestUser. Note that 512 is the numeric value for UF\_NORMAL\_ACCOUNT.

## CustomUserAccounts Table

UserName	Password	Attributes

TestUser	TESTUSERPASSWORD	512
----------	------------------	-----

Add the following records to the \_Validation table for the custom table.

**[\\_Validation Table](#)**

Table	Column	Nullable	MinValue	MaxValue	KeyTab
CustomUserAccounts	UserName	N			
CustomUserAccounts	Password	N			
CustomUserAccounts	Attributes	Y	0	2147483647	

Continue to [Authoring the ActionText and Error Tables](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring the ActionText and Error Tables

The sample specifications include sending ActionData messages when a custom action creates or removes a user account, and reporting an error if an account cannot be created.

Enter the following entries into the ActionText table to provide information used by ActionText messages.

## ActionText Table

Action	Description	Template
ProcessAccounts	Generating actions to create user accounts on the local computer.	Account: [1], Attributes: [2]
CreateAccount	Creating user account on the local computer.	Account: [1]
RemoveAccount	Removing user account from the local computer.	Account: [1]
RollbackAccount	Rolling back the creation of user accounts on the local computer.	Account: [1]
UninstallAccounts	Generating actions to remove user accounts on the local computer.	Account: [1]

Enter the following entries into the Error table to provide information used by error reporting.

## Error Table

Error	Message
25001	Unable to create user account '[2]' on the local machine. Error Code: [3].
25002	User account '[2]' already exists on the local machine.
25003	Unable to remove user account '[2]' on the local machine. Error Code: [3].

25004	User account '[2]' does not exist on the local machine.
-------	---

Continue to [Authoring the InstallExecuteSequence Table](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring the InstallExecuteSequence Table

The custom actions ProcessAccounts and UninstallAccounts generate the deferred custom actions that create, remove, or rollback user accounts. The custom actions ProcessAccounts and UninstallAccounts must be entered into the [InstallExecuteSequence table](#) to be executed. Add the following entries to the [InstallExecuteSequence table](#). Because these custom actions must be a part of the script generation, both custom actions must be sequenced after the [InstallInitialize action](#).

The condition on ProcessAccounts ensures the following. See [Conditional Statement Syntax](#).

- ProcessAccounts runs only on Windows 2000.
- ProcessAccounts runs only if the component TestAccount is being installed locally on the computer.
- The component Test Account is currently not installed or is installed to run from the source.

The condition on UninstallAccount ensures the following:

- UninstallAccounts runs only on Windows 2000.
- UninstallAccounts runs only if the component TestAccount is installed locally on the computer.
- The component Test Account is being removed or being installed to run from the source.

## InstallExecuteSequence Table

Action	Condition	Sequence
ProcessAccounts	VersionNT AND (?TestAccount=2 OR ?TestAccount=4) AND \$TestAccount=3	1550
UninstallAccounts	VersionNT AND ?TestAccount=3 AND (\$TestAccount=4 OR \$TestAccount=2)	1560

Continue to [Authoring the user interface for password input](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring the User Interface for Password Input

For each password that must be entered by the user, add an edit control on a dialog that stores the value of the password into a property. This [edit control](#) should have the [Password Control Attribute](#). This specifies that the property entered is a password and prevents the installer from writing the property to the log file.

The [control attributes](#) for the password edit control are `msidbControlAttributesVisible`, `msidbControlAttributesEnabled`, and `msidbControlAttributesPasswordInput (1 + 2 + 2097152)`. The X, Y, Width, Height, and `Control_Next` depend on the layout of the control on the dialog.

## Control Table

Dialog_	Control_	Type	X	Y	Width	Height	Attributes
MyDialog	TestUserPasswordEdit	Edit	25	120	300	20	2097155

Continue to [Securing the installation](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Securing the Installation

Add all of the Password properties from the CustomUserAccounts table to the **MsiHiddenProperties** property in the [Property table](#). Add the names of the deferred custom actions to the **MsiHiddenProperties** property as well. This will prevent the installer from writing the sensitive property values (the passwords) into the log and will ensure these values are not logged when the deferred custom actions use the values as the CustomActionData property.

For the user's password to be available in the Windows Installer service, the password property must be added to the **SecureCustomProperties** property.

## Property Table

Property	Value
<b>MsiHiddenProperties</b>	TESTUSERPASSWORD;CreateAccount;RemoveA
<b>SecureCustomProperties</b>	TESTUSERPASSWORD

This concludes the sample.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using 64-bit Custom Actions

On 64-bit operating systems, Windows Installer can call custom actions that are compiled for 32-bit or 64-bit systems. A 64-bit custom action based on [Scripts](#) must be explicitly marked as a 64-bit custom action by adding the `msidbCustomActionType64BitScript` bit to the custom action numeric type in the `Type` column of the [CustomAction Table](#). Custom actions based on [Executable files](#) or [Dynamic-Link Libraries](#) that are compiled for 64-bit operating systems do not require including this additional bit in the `Type` column of the `CustomAction Table`.

For more information, see [64-Bit Custom Actions](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Reference

The following sections contain information on custom actions:

- [Custom Action Types](#)
- [Custom Action Execution Scheduling Options](#)
- [Custom Action Hidden Target Option](#)
- [Custom Action In-Script Execution Options](#)
- [Custom Action Patch Uninstall Option](#)
- [Custom Action Return Processing Options](#)
- [Custom Action Return Values](#)
- [Return Values of JScript and VBScript Custom Actions](#)

## See Also

[About Custom Actions](#)

[Using Custom Actions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Types

The following table identifies the basic types of custom actions and shows the values that are in the Type, Source, and Target fields of the [CustomAction](#) table for each type. The basic custom actions can be modified by including optional flag bits in the Type column. For descriptions of the options and the values, see the following:

- [Custom Action Return Processing Options](#)
- [Custom Action Execution Scheduling Options](#)
- [Custom Action In-Script Execution Options](#)
- [Custom Action Patch Uninstall Option](#)

Use the links to the Basic Custom Action Type for a description and the options available for each type.

<b>Basic custom action type</b>	<b>Type</b>	<b>Source</b>	<b>Target</b>
<a href="#">Custom Action Type 1</a> DLL file stored in a Binary table stream.	1	Key to <a href="#">Binary</a> table.	DLL entry point.
<a href="#">Custom Action Type 2</a> EXE file stored in a Binary table stream.	2	Key to <a href="#">Binary</a> table.	Command-line string.
<a href="#">Custom Action Type 5</a> JScript file stored in a Binary table stream.	5	Key to <a href="#">Binary</a> table.	An optional JScript function that can be called.
<a href="#">Custom Action</a>	6	Key to <a href="#">Binary</a> table.	An optional VBScript

Type 6 VBScript file stored in a Binary table stream.			function that can be called.
Custom Action Type 17 DLL file that is installed with a product.	17	Key to File table.	DLL entry point.
Custom Action Type 18 EXE file that is installed with a product.	18	Key to File table.	Command-line string.
Custom Action Type 19 Displays a specified error message and returns failure, terminating the installation.	19	Blank	Formatted text string. The literal message or an index into the Error table.
Custom Action Type 21 JScript file that is installed with a product.	21	Key to File table.	An optional JScript function that can be called.
Custom Action Type 22 VBScript file that is installed with a product.	22	Key to File table.	An optional VBScript function that can be called.
Custom Action	34	Key to Directory	The Target column is

<p><b>Type 34</b> EXE file having a path referencing a directory.</p>		<p>table. This is the working directory for execution.</p>	<p>formatted and contains the full path and name of the executable file followed by optional arguments.</p>
<p><b>Custom Action Type 35</b> Directory set with formatted text.</p>	35	<p>A key to the <b>Directory</b> table. The designated directory is set by the formatted string in the Target field.</p>	<p>A formatted text string.</p>
<p><b>Custom Action Type 37</b> JScript text stored in this sequence table.</p>	37	<p>Null</p>	<p>A string of JScript code.</p>
<p><b>Custom Action Type 38</b> VBScript text stored in this sequence table.</p>	38	<p>Null</p>	<p>A string of VBScript code.</p>
<p><b>Custom Action Type 50</b> EXE file having a path specified by a property value.</p>	50	<p>Property name or key to <b>Property</b> table.</p>	<p>Command-line string.</p>
<p><b>Custom Action Type 51</b> Property set with formatted text.</p>	51	<p>Property name or key to the <b>Property</b> table. This property is set by the formatted string in the Target field.</p>	<p>A formatted text string.</p>
<p><b>Custom Action Type 53</b> JScript text specified by a</p>	53	<p>Property name or key to <b>Property</b> table.</p>	<p>An optional JScript function that can be called.</p>

property value.			
<a href="#">Custom Action Type 54</a> VBScript text specified by a property value.	54	Property name or key to <a href="#">Property</a> table.	An optional VBScript function that can be called.

In addition, the following custom action types are used with concurrent installations:

- [Custom Action Type 7](#)
- [Custom Action Type 23](#)
- [Custom Action Type 39](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Type 1

This custom action calls a dynamic link library (DLL) written in C or C++.

## Source

The DLL is generated from a temporary binary stream. The Source field of the [CustomAction table](#) contains a key to the [Binary table](#).

The Data column in the Binary table contains the stream data. A separate stream is allocated for each row. New binary data can be inserted from a file by using [MsiRecordSetStream](#) followed by [MsiViewModify](#) to insert the record into the table. When the custom action is invoked, the stream data is copied to a temporary file, which is then processed depending upon the type of custom action.

## Type Value

Include the following flag bits in the Type column of the CustomAction table to specify the basic numeric type.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeDll + msidbCustomActionTypeBinaryData	0x001	1

## Target

The DLL is called through the entry point named in the Target field of the CustomAction table, passing a single argument that is the handle to the current install session. The entry point name specified in the table must match that exported from the DLL. Note that if the entry function is not specified by a .DEF file or by a /EXPORT: linker specification, the name may have a leading underscore and a "@4" suffix. The called function must specify the `__stdcall` calling convention.

## Return Processing Options

Include optional flag bits in the Type column of the CustomAction table to specify return processing options. For a description of the options and the values, see [Custom Action Return Processing Options](#).

## Execution Scheduling Options

Include optional flag bits in the Type column of the CustomAction table to specify execution scheduling options. These options control the multiple execution of custom actions. For a description of the options, see [Custom Action Execution Scheduling Options](#).

## In-Script Execution Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify an in-script execution option. These options copy the action code into the execution, rollback, or commit script. For a description of the options, see [Custom Action In-Script Execution Options](#).

## Return Values

See [Custom Action Return Values](#).

## Remarks

A custom action that calls a dynamic-link library (DLL) requires a handle to the installation session. If this is also a deferred execution custom action, the session may no longer exist during execution of the installation script. For information on how a custom action of this type can obtain context information, see [Obtaining Context Information for Deferred Execution Custom Actions](#).

When a database table is exported, each stream is written as a separate file in the subfolder named after the table, using the primary key as the file name (Name column for the Binary table), with a default extension of ".ibd". The name should use the 8.3 format if the file system or version control system does not support long file names. The persistent archive file replaces the stream data with the file name used, so that the data can be located when the table is imported.

## See Also

[Custom\\_Actions](#)

[Dynamic-Link Libraries](#)

[Obtaining Context Information for Deferred Execution Custom Actions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Type 2

This custom action calls an executable launched with a command line.

## Source

The executable is generated from a temporary binary stream. The Source field of the [CustomAction table](#) contains a key to the [Binary table](#). The Data column in the Binary table contains the stream data. A separate stream is allocated for each row.

New binary data can be inserted from a file by using [MsiRecordSetStream](#) followed by [MsiViewModify](#) to insert the record into the table. When the custom action is invoked, the stream data is copied to a temporary file, which is then processed depending upon the type of custom action.

## Type Value

Include the following value in the Type column of the [CustomAction table](#) to specify the basic numeric type.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeExe + msidbCustomActionTypeBinaryData	0x002	2

## Target

The Target column of the [CustomAction table](#) contains the command line string for the executable named in the Source column.

## Return Processing Options

Include optional flag bits in the Type column of the CustomAction table to specify return processing options. For a description of the options and the values, see [Custom Action Return Processing Options](#).

## Execution Scheduling Options

Include optional flag bits in the Type column of the CustomAction table to specify execution scheduling options. These options control the multiple execution of custom actions. For a description of the options, see [Custom Action Execution Scheduling Options](#).

## In-Script Execution Options

Include optional flag bits in the Type column of the CustomAction table to specify an in-script execution option. These options copy the action code into the execution, rollback, or commit script. For a description of the options, see [Custom Action In-Script Execution Options](#).

## Return Values

Custom actions that are [executable files](#) must return a value of 0 for success. The installer interprets any other return value as failure. To ignore return values, set the msidbCustomActionTypeContinue bit flag in the Type field of the CustomAction table.

## Remarks

A custom action that launches an executable takes a command line, which commonly contains properties that are designated dynamically. If this is also a [deferred execution custom action](#), the installer uses **CreateProcessAsUser** or **CreateProcess** to create the process when the custom action is invoked from the installation script.

When a database table is exported, each stream is written as a separate file in the subfolder named after the table, using the primary key as the file name (Name column for the Binary table), with a default extension of ".ibd". The name should use the 8.3 format if the file system or version control system does not support long file names. The persistent archive file replaces the stream data with the file name used, so that the data can be located when the table is imported.

## See Also

[Custom\\_Actions](#)  
[Executable Files](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Type 5

This custom action is written in JScript, such as ECMA 262. Windows Installer does not support JScript 1.0. For more information, see [Scripts](#).

## Source

The script is generated from a temporary binary stream. The Source field of the [CustomAction table](#) contains a key to the [Binary table](#). The Data column in the Binary table contains the stream data. A separate stream is allocated for each row.

New binary data can be inserted from a file by using [MsiRecordSetStream](#) followed by [MsiViewModify](#) to insert the record into the table. When the custom action is invoked, the stream data is copied to a temporary file, which is then processed according to the type of custom action.

## Type Value

Include the following value in the Type column of the [CustomAction table](#) to specify the basic numeric type of 32-bit custom action.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeJScript + msidbCustomActionTypeBinaryData	0x05	5

Windows Installer may use 64-bit custom actions on 64-bit operating systems. A 64-bit custom action based on scripts must include the `msidbCustomActionType64BitScript` bit in its numeric type. For information see [64-bit Custom Actions](#). Include the following value in the Type column of the [CustomAction table](#) to specify the basic numeric type of a 64-bit custom action.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeJScript + msidbCustomActionTypeBinaryData +	0x0001005	4101

msidbCustomActionType64BitScript		
----------------------------------	--	--

## Target

The Target field of the CustomAction table contains an optional script function. Processing first sends the script for parsing and then calls the optional script function.

## Return Processing Options

Include optional flag bits in the Type column of the CustomAction table to specify return processing options. For a description of the options and the values, see [Custom Action Return Processing Options](#).

## Execution Scheduling Options

Include optional flag bits in the Type column of the CustomAction table to specify execution scheduling options. These options control the multiple execution of custom actions. For a description of the options, see [Custom Action Execution Scheduling Options](#).

## In-Script Execution Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify an in-script execution option. These options copy the action code into the execution, rollback, or commit script. For a description of the options, see [Custom Action In-Script Execution Options](#).

## Return Values

Optional functions written in script must return one of the values described in [Return Values of JScript and VBScript Custom Actions](#).

## Remarks

A custom action that is written in JScript or VBScript requires the

installation of **Session Object**. The installer attaches the **Session** object to the script with the name *Session*. Because the **Session** object may not exist during an installation rollback, a deferred custom action written in script must use one of the methods or properties of the **Session** object described in the section [Obtaining Context Information for Deferred Execution Custom Actions](#) to retrieve its context.

When a database table is exported, each stream is written as a separate file in the subfolder named after the table, using the primary key as the file name (Name column for the Binary table), with a default extension of ".ibd". The name should use the 8.3 file name format if the file system or version control system does not support long file names. The persistent archive file replaces the stream data with the file name used, so that the data can be located when the table is imported.

## See Also

[Custom\\_Actions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Type 6

This custom action is written in VBScript. For more information, see [Scripts](#).

## Source

The script is generated from a temporary binary stream. The Source field of the [CustomAction table](#) contains a key to the [Binary table](#). The Data column in the Binary table contains the stream data. A separate stream is allocated for each row.

New binary data can be inserted from a file by using [MsiRecordSetStream](#) followed by [MsiViewModify](#) to insert the record into the table. When the custom action is invoked, the stream data is copied to a temporary file, which is then processed depending upon the type of custom action.

## Type Value

Include the following value in the Type column of the CustomAction table to specify the basic numeric type of a 32-bit custom action.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeVBScript + msidbCustomActionTypeBinaryData	0x006	6

Windows Installer may use 64-bit custom actions on 64-bit operating systems. A 64-bit custom action based on scripts must include the msidbCustomActionType64BitScript bit in its numeric type. For information see [64-bit Custom Actions](#). Include the following value in the Type column of the CustomAction table to specify the basic numeric type of a 64-bit custom action.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeVBScript + msidbCustomActionTypeBinaryData +	0x0001006	4102

msidbCustomActionType64BitScript		
----------------------------------	--	--

## Target

The Target field of the CustomAction table contains an optional script function. Processing first sends the script for parsing and then calls the optional script function.

## Return Processing Options

Include optional flag bits in the Type column of the CustomAction table to specify return processing options. For a description of the options and the values, see [Custom Action Return Processing Options](#).

## Execution Scheduling Options

Include optional flag bits in the Type column of the CustomAction table to specify execution scheduling options. These options control the multiple execution of custom actions. For a description of the options, see [Custom Action Execution Scheduling Options](#).

## In-Script Execution Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify an in-script execution option. These options copy the action code into the execution, rollback, or commit script. For a description of the options, see [Custom Action In-Script Execution Options](#).

## Return Values

Optional functions written in script must return one of the values described in [Return Values of JScript and VBScript Custom Actions](#).

## Remarks

A custom action that is written in JScript or VBScript requires the

installation of the **Session Object**. The installer attaches the **Session** object to the script with the name *Session*. Because the **Session** object may not exist during an installation rollback, a deferred custom action written in script must use one of the methods or properties of the **Session** object described in the section [Obtaining Context Information for Deferred Execution Custom Actions](#) to retrieve its context.

When a database table is exported, each stream is written as a separate file in the subfolder named after the table, using the primary key as the file name (Name column for the Binary table), with a default extension of ".ibd". The name should use the 8.3 file name format if the file system or version control system does not support long file names. The persistent archive file replaces the stream data with the file name used, so that the data can be located when the table is imported.

## See Also

[Custom\\_Actions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Type 17

This custom action calls a dynamic link library (DLL) written in C or C++.

## Source

The DLL is installed with the application during the current session. The Source field of the [CustomAction table](#) contains a key to the [File table](#). The location of the custom action code is determined by the resolution of the target path for this file; therefore this custom action must be called after that file has been installed and before it is removed.

## Type Value

Include the following value in the Type column of the [CustomAction table](#) to specify the basic numeric type.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeDll + msidbCustomActionTypeSourceFile	0x011	17

## Target

The DLL is called through the entry point named in the Target field of the [CustomAction table](#), passing a single argument that is the handle to the current install session. The entry point name specified in the table must match that exported from the DLL. Note that if the entry function is not specified by a .DEF file or by a /EXPORT: linker specification, the name may have a leading underscore and a "@4" suffix. The called function must specify the `__stdcall` calling convention.

## Return Processing Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify return processing options. For a description of the options and the values, see [Custom Action Return Processing Options](#).

## Execution Scheduling Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify execution scheduling options. These options control the multiple execution of custom actions. For a description of the options, see [Custom Action Execution Scheduling Options](#).

## In-Script Execution Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify an in-script execution option. These options copy the action code into the execution, rollback, or commit script. For a description of the options, see [Custom Action In-Script Execution Options](#).

## Return Values

See [Custom Action Return Values](#).

## Remarks

A custom action that calls a dynamic-link library (DLL) requires a handle to the installation session. If this is also a deferred execution custom action, the session may no longer exist during execution of the installation script. For information on how a custom action of this type can obtain context information, see [Obtaining Context Information for Deferred Execution Custom Actions](#).

Custom actions execute in a separate thread, and may have limited access to the system. Custom actions that run asynchronously block the main thread at the termination of either the current sequence or the install session until they return.

Custom actions that reference an installed file as their source, such as Custom Action Type 17 (DLL), must adhere to the following sequencing restrictions:

- The custom action must be sequenced after the [CostFinalize action](#). This is so that the custom action can resolve the path needed to locate the DLL.

- If the source file is not already installed on the computer, deferred (in-script) custom actions of this type must be sequenced after the [InstallFiles action](#).
- If the source file is not already installed on the computer, non-deferred custom actions of this type must be sequenced after the [InstallFinalize action](#).

## See Also

[Custom\\_Actions](#)

[Deferred Execution Custom Actions](#)

[Dynamic-Link Libraries](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Type 18

This custom action calls an executable launched with a command line.

## Source

The executable is generated from a file installed with the application. The Source field of the [CustomAction table](#) contains a key to the [File table](#). The location of the custom action code is determined by the resolution of the target path for this file; therefore this custom action must be called after the file has been installed and before it is removed.

## Type Value

Include the following value in the Type column of the [CustomAction table](#) to specify the basic numeric type.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeExe + msidbCustomActionTypeSourceFile	0x012	18

## Target

The Target column of the [CustomAction table](#) contains the command line string for the executable identified in the Source column.

## Return Processing Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify return processing options. For a description of the options and the values, see [Custom Action Return Processing Options](#).

## Execution Scheduling Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify execution scheduling options. These options control the multiple

execution of custom actions. For a description of the options, see [Custom Action Execution Scheduling Options](#).

## In-Script Execution Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify an in-script execution option. These options copy the action code into the execution, rollback, or commit script. For a description of the options, see [Custom Action In-Script Execution Options](#).

## Return Values

Custom actions that are [executable files](#) must return a value of 0 for success. The installer interprets any other return value as failure. To ignore return values, set the `msidbCustomActionTypeContinue` bit flag in the Type field of the CustomAction table.

## Remarks

A custom action that launches an executable takes a command line, which commonly contains properties that are designated dynamically. If this is also a [deferred execution custom action](#), the installer uses **CreateProcessAsUser** or **CreateProcess** to create the process when the custom action is invoked from the installation script.

Custom actions that reference an installed file as their source, such as Custom Action Type 18 (EXE), must adhere to the following sequencing restrictions:

- The custom action must be sequenced after the [CostFinalize action](#). This is so that the custom action can resolve the path needed to locate the EXE.
- If the source file is not already installed on the computer, deferred (in-script) custom actions of this type must be sequenced after the [InstallFiles action](#).
- If the source file is not already installed on the computer, non-deferred custom actions of this type must be sequenced after the

[InstallFinalize action.](#)

## **See Also**

[Custom\\_Actions](#)  
[Executable Files](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Type 19

This custom action displays a specified error message, returns failure, and then terminates the installation. The error message displayed can be supplied as a string or as an index into the [Error table](#).

## Source

Leave the Source column of the [CustomAction table](#) blank.

## Type Value

Include the following value in the Type column of the CustomAction table to specify the basic numeric type.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeTextData + msidbCustomActionTypeSourceFile	0x013	19

## Target

The Target column of the [CustomAction table](#) contains a text string formatted using the functionality specified in [MsiFormatRecord](#) (without the numeric field specifiers). Parameters to be replaced are enclosed in square brackets, [...], and may be properties, environment variables (% prefix), file paths (# prefix), or component directory paths (\$) prefix). If after formatting the string evaluates to an integer, that integer is used as an index into the [Error table](#) to retrieve the message to display. If after formatting the string contains non-numeric characters, the string itself is displayed as the message.

## Return Processing Options

The custom action does not use any options.

## Execution Scheduling Options

The custom action does not use any options.

## In-Script Execution Options

The custom action does not use any options.

## Return Values

See [Custom Action Return Values](#).

## Remarks

For example, the custom actions CAError1, CAError2, CAError3, and CAError4 return these messages.

### CustomAction Table

Action	Type	Source	Target
CAError1	19		[Prop1]
CAError2	19		Installation failure due to Error2.
CAError3	19		25000
CAError4	19		[Prop2]

### Property Table

Property	Value
Prop1	"Installation failure due to Error1."
Prop2	"25100"

### Error Table

Code	Message
25000	Installation failure due to Error3.
25100	Installation failure due to Error4.

These custom actions return the following error messages:

<b>Custom action</b>	<b>Returned message string</b>
CAError1	Installation failure due to Error1.
CAError2	Installation failure due to Error2.
CAError3	Installation failure due to Error3.
CAError4	Installation failure due to Error4.

Note that because the order of evaluation of launch conditions cannot be guaranteed by authoring the [LaunchCondition table](#), you should use Custom Action Type 19 custom actions in your installation to evaluate conditions in a specific order.

## See Also

[Custom\\_Actions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Type 21

This custom action is written in JScript, such as ECMA 262. Windows Installer does not support JScript 1.0. For more information, see [Scripts](#).

## Source

The script is installed with the application during the current session. The Source field of the [CustomAction table](#) contains a key to the [File table](#). The location of the custom action code is determined by the resolution of the target path for this file; therefore this custom action must be called after the file has been installed and before it is removed.

## Type Value

Include the following value in the Type column of the [CustomAction table](#) to specify the basic numeric type of a 32-bit custom action.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeJScript + msidbCustomActionTypeSourceFile	0x015	21

Windows Installer may use [64-bit Custom Actions](#) on 64-bit operating systems. A 64-bit custom action based on scripts must include the msidbCustomActionType64BitScript bit in its numeric type. For information see [64-bit Custom Actions](#). Include the following value in the Type column of the [CustomAction table](#) to specify the basic numeric type of a 64-bit custom action.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeJScript + msidbCustomActionTypeSourceFile + msidbCustomActionType64BitScript	0x0001015	4117

## Target

The Target field of the [CustomAction table](#) contains an optional script function. Processing first sends the script for parsing and then calls the optional script function.

## Return Processing Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify return processing options. For a description of the options and the values, see [Custom Action Return Processing Options](#).

## Execution Scheduling Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify execution scheduling options. These options control the multiple execution of custom actions. For a description of the options, see [Custom Action Execution Scheduling Options](#).

## In-Script Execution Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify an in-script execution option. These options copy the action code into the execution, rollback, or commit script. For a description of the options, see [Custom Action In-Script Execution Options](#).

## Return Values

Optional functions written in script must return one of the values described in [Return Values of JScript and VBScript Custom Actions](#).

## Remarks

A custom action that is written in JScript or VBScript requires the installation [Session](#) object. The installer attaches the **Session Object** to the script with the name "Session". Because the **Session** object may not exist during an installation rollback, a deferred custom action written in script must use one of the methods or properties of the **Session** object described in the section [Obtaining Context Information for Deferred Execution Custom Actions](#) to retrieve its context.

Custom actions that reference an installed file as their source, such as Custom Action Type 21 (JScript), must adhere to the following sequencing restrictions:

- The custom action must be sequenced after the [CostFinalize action](#). This is so that the custom action can resolve the path needed to locate the source file containing the JScript.
- If the source file is not already installed on the computer, deferred (in-script) custom actions of this type must be sequenced after the [InstallFiles action](#).
- If the source file is not already installed on the computer, non-deferred custom actions of this type must be sequenced after the [InstallFinalize action](#).

## See Also

[Custom\\_Actions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Type 22

This custom action is written in VBScript. See also [Scripts](#).

## Source

The script is installed with the application during the current session. The Source field of the [CustomAction table](#) contains a key to the [File table](#). The location of the custom action code is determined by the resolution of the target path for this file; therefore this custom action must be called after the file has been installed and before it is removed.

## Type Value

Include the following value in the Type column of the [CustomAction table](#) to specify the basic numeric type of a 32-bit custom action.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeVBScript + msidbCustomActionTypeSourceFile	0x016	22

Windows Installer may use 64-bit custom actions on 64-bit operating systems. A 64-bit custom action based on scripts must include the msidbCustomActionType64BitScript bit in its numeric type. For information see [64-bit Custom Actions](#). Include the following value in the Type column of the [CustomAction table](#) to specify the basic numeric type of a 64-bit custom action.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeVBScript + msidbCustomActionTypeSourceFile + msidbCustomActionType64BitScript	0x0001016	4118

## Target

The Target field of the [CustomAction table](#) contains an optional script function. Processing first sends the script for parsing and then calls the optional script function.

## Return Processing Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify return processing options. For a description of the options and the values, see [Custom Action Return Processing Options](#).

## Execution Scheduling Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify execution scheduling options. These options control the multiple execution of custom actions. For a description of the options, see [Custom Action Execution Scheduling Options](#).

## In-Script Execution Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify an in-script execution option. These options copy the action code into the execution, rollback, or commit script. For a description of the options, see [Custom Action In-Script Execution Options](#).

## Return Values

Optional functions written in script must return one of the values described in [Return Values of JScript and VBScript Custom Actions](#).

## Remarks

A custom action that is written in JScript or VBScript requires the install [Session Object](#). This is of the type **Session Object** and the installer attaches it to the script with the name "Session". Because the **Session** object may not exist during an installation rollback, a deferred custom action written in script must use one of the methods or properties of the **Session** object described in the section [Obtaining Context Information for Deferred Execution Custom Actions](#) to retrieve its context.

Custom actions that reference an installed file as their source, such as Custom Action Type 22 (VBScript), must adhere to the following sequencing restrictions:

- The custom action must be sequenced after the [CostFinalize action](#). This is so that the custom action can resolve the path needed to locate the source file containing the VBScript.
- If the source file is not already installed on the computer, deferred (in-script) custom actions of this type must be sequenced after the [InstallFiles action](#).
- If the source file is not already installed on the computer, non-deferred custom actions of this type must be sequenced after the [InstallFinalize action](#).

## See Also

[Custom\\_Actions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Type 34

This custom action calls an executable launched with a command line. For more information, see [Executable Files](#).

## Source

The executable is generated from a file. The Source field of the [CustomAction](#) table contains a key into the [Directory](#) table. The referenced Directory table entry is used to resolve the full path to a working directory. This is not required to be the path to the directory containing the executable.

## Type Value

Include the following value in the Type column of the [CustomAction](#) table to specify the basic numeric type.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeExe + msidbCustomActionTypeDirectory	0x022	34

## Target

The Target column of the [CustomAction](#) table contains the full path and name of the executable file followed by optional arguments to the executable. The full path and name to the executable file is required. Quotation marks must be used around long file names or paths. The value is treated as [formatted](#) text and may contain references to properties, files, directories, or other formatted text attributes.

## Return Processing Options

Include optional flag bits in the Type column of the [CustomAction](#) table to specify return processing options. For a description of the options and the values, see [Custom Action Return Processing Options](#).

## Execution Scheduling Options

Include optional flag bits in the Type column of the [CustomAction](#) table to specify execution scheduling options. These options control the multiple execution of custom actions. For a description of the options, see [Custom Action Execution Scheduling Options](#).

## In-Script Execution Options

Include optional flag bits in the Type column of the [CustomAction](#) table to specify an in-script execution option. These options copy the action code into the execution, rollback, or commit script. For a description of the options, see [Custom Action In-Script Execution Options](#).

## Return Values

Custom actions that are [executable files](#) must return a value of 0 for success. The installer interprets any other return value as failure. To ignore return values set the `msidbCustomActionTypeContinue` bit flag in the Type field of the [CustomAction](#) table.

## Remarks

A custom action that launches an executable takes a command line, which commonly contains properties that are designated dynamically. If this is also a [deferred execution custom action](#), the installer uses **CreateProcessAsUser** or **CreateProcess** to create the process when the custom action is invoked from the installation script.

## See Also

[Custom\\_Actions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Type 35

This custom action sets the install directory from a formatted text string. For more information, see [Changing the Target Location for a Directory](#)

## Source

The Source field of the [CustomAction table](#) contains a key to the [Directory table](#). The designated directory is set by the formatted string in the Target field using [MsiSetTargetPath](#). This sets the target path and associated property to the expanded value of the formatted text string in the Target field. Do not attempt to change the location of a target directory during a [maintenance installation](#). Do not attempt to change the target directory path if some components using that path are already installed for any user.

## Type Value

Include the following value in the Type column of the [CustomAction table](#) to specify the basic numeric type.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeTextData + msidbCustomActionTypeDirectory	0x023	35

## Target

The Target column of the [CustomAction table](#) contains a text string formatted using the functionality specified in [MsiFormatRecord](#) (without the numeric field specifiers). Parameters to be replaced are enclosed in square brackets [...], and may be properties, environment variables (% prefix), file paths (# prefix), or component directory paths (\$ prefix). Note that directory paths always end with a directory separator.

## Return Processing Options

The custom action does not use these options.

## Execution Scheduling Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify execution scheduling options. These options control the multiple execution of custom actions. For a description of the options, see [Custom Action Execution Scheduling Options](#).

## In-Script Execution Options

The custom action does not use these options.

## Return Values

See [Custom Action Return Values](#).

## Remarks

If you set a [private property](#) in the UI sequence by authoring a custom action in one of the user interface sequence tables, that property is not set in the execution sequence. To set the property in the execution sequence you must also put a custom action in an execution sequence table. Alternatively, you can make the property a [public property](#) and include it in the [SecureCustomProperties property](#).

## See Also

[Custom\\_Actions](#)

[Formatted Text Custom Actions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Type 37

This custom action is written in JScript, such as ECMA 262. Windows Installer does not support JScript 1.0. For more information, see [Scripts](#).

## Source

The Source field of the [CustomAction table](#) contains the null value. The script code for the custom action is stored as a string of literal script text in the Target field.

## Type Value

Include the following value in the Type column of the [CustomAction table](#) to specify the basic numeric type of a 32-bit custom action.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeJScript + msidbCustomActionTypeDirectory	0x025	37

Windows Installer may use 64-bit custom actions on 64-bit operating systems. A 64-bit custom action based on scripts must include the msidbCustomActionType64BitScript bit in its numeric type. For information see [64-bit Custom Actions](#). Include the following value in the Type column of the [CustomAction table](#) to specify the basic numeric type of a 64-bit custom action.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeJScript + msidbCustomActionTypeDirectory + msidbCustomActionType64BitScript	0x0001025	4133

## Target

The Target field of the [CustomAction table](#) contains the script code for the

custom action as a string of literal script text.

## Return Processing Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify return processing options. For a description of the options and the values, see [Custom Action Return Processing Options](#).

## Execution Scheduling Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify execution scheduling options. These options control the multiple execution of custom actions. For a description of the options, see [Custom Action Execution Scheduling Options](#).

## In-Script Execution Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify an in-script execution option. These options copy the action code into the execution, rollback, or commit script. For a description of the options, see [Custom Action In-Script Execution Options](#).

## Return Values

This custom action type always returns success.

## Remarks

A custom action that is written in JScript or VBScript requires the install [Session](#) object. The installer attaches the **Session Object** to the script with the name "Session". Because the **Session** object may not exist during an installation rollback, a deferred custom action written in script must use one of the methods or properties of the **Session** object described in the section [Obtaining Context Information for Deferred Execution Custom Actions](#) to retrieve its context.

## See Also

[Custom\\_Actions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Type 38

This custom action is written in VBScript. See also [Scripts](#).

## Source

The Source field of the [CustomAction table](#) contains the null value. The script code for the custom action is stored as a string of literal script text in the Target field.

## Type Value

Include the following value in the Type column of the [CustomAction table](#) to specify the basic numeric type of a 32-bit custom action.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeVBScript + msidbCustomActionTypeDirectory	0x026	38

Windows Installer may use 64-bit custom actions on 64-bit operating systems. A 64-bit custom action based on scripts must include the msidbCustomActionType64BitScript bit in its numeric type. For information see [64-bit Custom Actions](#). Include the following value in the Type column of the [CustomAction table](#) to specify the basic numeric type of a 64-bit custom action.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeVBScript + msidbCustomActionTypeDirectory + msidbCustomActionType64BitScript	0x0001026	4134

## Target

The Target field of the [CustomAction table](#) contains the script code for the custom action as a string of literal script text.

## Return Processing Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify return processing options. For a description of the options and the values, see [Custom Action Return Processing Options](#).

## Execution Scheduling Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify execution scheduling options. These options control the multiple execution of custom actions. For a description of the options, see [Custom Action Execution Scheduling Options](#).

## In-Script Execution Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify an in-script execution option. These options copy the action code into the execution, rollback, or commit script. For a description of the options, see [Custom Action In-Script Execution Options](#).

## Return Values

This custom action type always returns success.

## Remarks

A custom action that is written in JScript or VBScript requires the install [Session](#) object. The installer attaches the **Session Object** to the script with the name "Session". Because the **Session** object may not exist during an installation rollback, a deferred custom action written in script must use one of the methods or properties of the **Session** object described in the section [Obtaining Context Information for Deferred Execution Custom Actions](#) to retrieve its context.

## See Also

[Custom\\_Actions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Type 50

This custom action calls an executable launched with a command line.

See also [Executable Files](#).

## Source

The executable is generated from an existing file. The Source field of the [CustomAction table](#) contains a key to the [Property table](#) for a property that contains the full path to the executable file.

## Type Value

Include the following value in the Type column of the [CustomAction table](#) to specify the basic numeric type.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeExe + msidbCustomActionTypeProperty	0x032	50

## Target

The Target column of the [CustomAction table](#) contains the command line string for the executable identified in the Source column.

## Return Processing Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify return processing options. For a description of the options and the values, see [Custom Action Return Processing Options](#).

## Execution Scheduling Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify execution scheduling options. These options control the multiple

execution of custom actions. For a description of the options, see [Custom Action Execution Scheduling Options](#).

## In-Script Execution Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify an in-script execution option. These options copy the action code into the execution, rollback, or commit script. For a description of the options, see [Custom Action In-Script Execution Options](#).

## Return Values

Custom actions that are [executable files](#) must return a value of 0 for success. The installer interprets any other return value as failure. To ignore return values set the msidbCustomActionTypeContinue bit flag in the Type field of the CustomAction table.

## Remarks

A custom action that launches an executable takes a command line, which commonly contains properties that are designated dynamically. If this is also a [deferred execution custom action](#), the installer uses CreateProcessAsUser or CreateProcess to create the process when the custom action is invoked from the installation script.

## See Also

[Custom\\_Actions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Type 51

This custom action sets a property from a formatted text string.

To affect a property used in a condition on a component or feature, the custom action must come before the [CostFinalize action](#) in the action sequence.

## Source

The Source field of the [CustomAction table](#) can contain either the name of a property or a key to the [Property table](#). This property is set by the formatted string in the Target field using [Msi SetProperty](#).

## Type Value

Include the following value in the Type column of the [CustomAction table](#) to specify the basic numeric type.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeTextData + msidbCustomActionTypeProperty	0x033	51

## Target

The Target column of the [CustomAction table](#) contains a text string formatted using the functionality specified in [MsiFormatRecord](#) (without the numeric field specifiers). Parameters to be replaced are enclosed in square brackets, [...], and may be properties, environment variables (% prefix), file paths (# prefix), or component directory paths (\$) prefix).

## Return Processing Options

The custom action does not use these options.

## Execution Scheduling Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify execution scheduling options. These options control the multiple execution of custom actions. For a description of the options, see [Custom Action Execution Scheduling Options](#).

## In-Script Execution Options

The custom action does not use these options.

## Return Values

See [Custom Action Return Values](#).

## Remarks

If you set a [private property](#) in the UI sequence by authoring a custom action in one of the user interface sequence tables, that property is not set in the execution sequence. To set the property in the execution sequence, you must also put a custom action in an execution sequence table. Alternatively, you can make the property a [public property](#) and include it in the [SecureCustomProperties property](#).

## See Also

[Custom\\_Actions](#)

[Formatted Text Custom Actions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Type 53

This custom action is written in JScript, such as ECMA 262. Windows Installer does not support JScript 1.0. For more information, see [Scripts](#).

## Source

The Source field of the [CustomAction table](#) contains a property name or a key to the [Property table](#) for a property containing the script text.

## Type Value

Include the following value in the Type column of the [CustomAction table](#) to specify the basic numeric type of a 32-bit custom action.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeJScript + msidbCustomActionTypeProperty	0x035	53

Windows Installer may use 64-bit custom actions on 64-bit operating systems. A 64-bit custom action based on scripts must include the msidbCustomActionType64BitScript bit in its numeric type. For information see [64-bit Custom Actions](#). Include the following value in the Type column of the [CustomAction table](#) to specify the basic numeric type of a 64-bit custom action.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeJScript + msidbCustomActionTypeProperty + msidbCustomActionType64BitScript	0x0001035	4149

## Target

The Target field of the [CustomAction table](#) contains an optional script function. Processing first sends the script for parsing and then calls the

optional script function.

## Return Processing Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify return processing options. For a description of the options and the values, see [Custom Action Return Processing Options](#).

## Execution Scheduling Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify execution scheduling options. These options control the multiple execution of custom actions. For a description of the options, see [Custom Action Execution Scheduling Options](#).

## In-Script Execution Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify an in-script execution option. These options copy the action code into the execution, rollback, or commit script. For a description of the options, see [Custom Action In-Script Execution Options](#).

## Return Values

Optional functions written in script must return one of the values described in [Return Values of JScript and VBScript Custom Actions](#).

## Remarks

A custom action that is written in JScript requires the installation [Session](#) object. Because the **Session** object may not exist during an installation rollback, a deferred custom action written in script uses one of the methods described in [Obtaining Context Information for Deferred Execution Custom Actions](#).

## See Also

[Custom\\_Actions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Type 54

This custom action is written in VBScript. See also [Scripts](#).

## Source

The Source field of the [CustomAction table](#) contains a property name or a key to the [Property table](#) for a property containing the script text.

## Type Value

Include the following value in the Type column of the [CustomAction table](#) to specify the basic numeric type of a 32-bit custom action.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeVBScript + msidbCustomActionTypeProperty	0x036	54

Windows Installer may use 64-bit custom actions on 64-bit operating systems. A 64-bit custom action based on scripts must include the msidbCustomActionType64BitScript bit in its numeric type. For information see [64-bit Custom Actions](#). Include the following value in the Type column of the [CustomAction table](#) to specify the basic numeric type of a 64-bit custom action.

Constants	Hexadecimal	Decimal
msidbCustomActionTypeVBScript + msidbCustomActionTypeProperty + msidbCustomActionType64BitScript	0x0001036	4150

## Target

The Target field of the [CustomAction table](#) contains an optional script function. Processing first sends the script for parsing and then calls the optional script function.

## Return Processing Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify return processing options. For a description of the options and the values, see [Custom Action Return Processing Options](#).

## Execution Scheduling Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify execution scheduling options. These options control the multiple execution of custom actions. For a description of the options, see [Custom Action Execution Scheduling Options](#).

## In-Script Execution Options

Include optional flag bits in the Type column of the [CustomAction table](#) to specify an in-script execution option. These options copy the action code into the execution, rollback, or commit script. For a description of the options, see [Custom Action In-Script Execution Options](#).

## Return Values

Optional functions written in script must return one of the values described in [Return Values of JScript and VBScript Custom Actions](#).

## Remarks

A custom action that is written in JScript or VBScript requires the install [Session](#) object. The installer attaches the **Session Object** to the script with the name *Session*. Because the **Session** object may not exist during an installation rollback, a deferred custom action written in script must use one of the methods or properties of the **Session** object described in the section [Obtaining Context Information for Deferred Execution Custom Actions](#) to retrieve its context.

## See Also

[Custom\\_Actions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Execution Scheduling Options

Because a custom action can be scheduled in both the UI and execute sequence tables, and can be executed either in the service or client process, a custom action can potentially execute multiple times.

Note that the installer:

- Executes actions in a sequence table immediately by default.
- Does not execute an action if the conditional expression field of the sequence table evaluates to False.
- Processes the UI sequence table in the client process if the internal user's interface level is set to the full UI mode (see [MsiSetInternalUI](#) for a description of UI levels).
- Is a service registered by default when using Windows 2000 and, in this case, the execute sequence table is processed in the installer service.

You can use the following option flags to control multiple immediate execution of custom actions. To set an option, add the value in this table to the value in the Type field of the [CustomAction](#) table. None of the following flags should be used with [deferred execution custom actions](#).

(default)

Hexadecimal: 0x00000000

Decimal: 0

Always execute. Action may run twice if present in both sequence tables.

msidbCustomActionTypeFirstSequence

Hexadecimal: 0x00000100

Decimal: 256

Execute no more than once if present in both sequence tables.

Always skips action in execute sequence if UI sequence has run. No

effect in UI sequence. The action is not required to be present or run in the UI sequence to be skipped in the execute sequence. Not affected by install service registration.

msidbCustomActionTypeOncePerProcess

Hexadecimal: 0x00000200

Decimal: 512

Execute once per process if in both sequence tables. Skips action in execute sequence if UI sequence has been run in same process, for example both run in the client process. Used to prevent actions that modify the session state, such as property and database data, from running twice.

msidbCustomActionTypeClientRepeat

Hexadecimal: 0x00000300

Decimal: 768

Execute only if running on client after UI sequence has run. The action runs only if the execute sequence is run on the client following UI sequence. May be used to provide either/or logic, or to suppress the UI-related processing if already done for the client session.

Note that to run a custom action during two different run modes, author two entries into the [CustomAction table](#) . For example, to have a custom action that calls a C/C++ dynamic link library (DLL) ( [Custom Action Type 1](#)) both when the mode is MSIRUNMODE\_SCHEDULED and MSIRUNMODE\_ROLLBACK, put two entries in the CustomAction table that call the same DLL but that have different numeric types. Include code that calls [MsiGetMode](#) to determine when to run which custom action.

## See Also

[Custom Action Reference](#)

[About Custom Actions](#)

[Using Custom Actions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Hidden Target Option

Use the following option flags to specify that the installer not write the value entered into the Target field of the [CustomAction table](#) into the log. To set the option add the value in this table to the value in the Type field of the CustomAction table.

Constant	Hexadecimal	Decimal	Description
(none)	0x0000	0	The installer may write the value in the Target column of the CustomAction table into the log file.
msidbCustomActionTypeHideTarget	0x2000	8192	<p>The installer is prevented from writing the value in the Target column of the <a href="#">CustomAction table</a> into the log file. The CustomActionDescription property is also not logged when the installer executes a custom action.</p> <p>Because the installer sets the value of CustomActionDescription from a property with the same name as a custom action, the property must be in the <a href="#">MsiHiddenProperty</a> Property to prevent the value from appearing in the log.</p>

---

For more information, see [Preventing Confidential Information from Being Written into the Log File](#)

## **See Also**

[Custom Action Reference](#)

[About Custom Actions](#)

[Using Custom Actions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action In-Script Execution Options

You can use the following option flags to specify the in-script execution of custom actions. These options copy the action code into the execution, rollback, or commit script. To set an option add the value in this table to the value in the Type field of the [CustomAction table](#).

Note that the `msidbCustomActionTypeInScript` must be included with each of these options.

Term	Description
(none)	Hexadecimal: 0x00000000 Decimal: 0 Immediate execution.
<code>msidbCustomActionTypeInScript</code>	Hexadecimal: 0x00000400 Decimal: 1024 Queues for execution at scheduled point within script. This flag designates that this is a <a href="#">deferred execution custom action</a> .
<code>msidbCustomActionTypeInScript + msidbCustomActionTypeRollback</code>	Hexadecimal: 0x00000400 + 0x00000100 Decimal: 1280 Queues for execution at scheduled point within script. Executes only upon an installation rollback. This flag designates that this is a <a href="#">rollback custom action</a> .
<code>msidbCustomActionTypeInScript + msidbCustomActionTypeCommit</code>	Hexadecimal: 0x00000400 + 0x00000200 Decimal: 1536

	<p>Queues for execution at scheduled point within script. Executes only upon instance commit. This flag designates that this is a <a href="#">commit custom action</a>.</p>
<p>msidbCustomActionTypeInScript + msidbCustomActionTypeNoImpersonate</p>	<p>Hexadecimal: 0x00000400 + 0x00000800</p> <p>Decimal: 3072</p> <p>Queues for execution at scheduled point within script. Executes with no user impersonation. Runs in system context.</p>
<p>msidbCustomActionTypeInScript + msidbCustomActionTypeNoImpersonate + msidbCustomActionTypeRollback</p>	<p>Hexadecimal: 0x00000400 + 0x00000800 + 0x00000100</p> <p>Decimal: 3328</p> <p>Queues for execution at scheduled point within script. Executes with no user impersonation. Runs in system context. This flag combination designates that this is a <a href="#">rollback custom action</a>.</p>
<p>msidbCustomActionTypeInScript + msidbCustomActionTypeNoImpersonate + msidbCustomActionTypeCommit</p>	<p>Hexadecimal: 0x00000400 + 0x00000800 + 0x00000200</p> <p>Decimal: 3584</p> <p>Queues for execution at scheduled point within script. Executes with no user impersonation. Runs in system context. This flag combination designates that this is a <a href="#">commit custom action</a>.</p>
<p>msidbCustomActionTypeTSAware + msidbCustomActionTypeInScript</p>	<p>Hexadecimal: 0x00000400 + 0x00004000</p> <p>Decimal: 17408</p> <p>Queues for execution at the scheduled point within script. Executes with user</p>

	<p>impersonation. Runs with user impersonation during per-machine installs on a server running the Terminal Server role service. Normal deferred execution custom actions, without this attribute, run with no user impersonation on a terminal server during per-machine installations. This attribute has no effect if the action also has the <code>msidbCustomActionTypeNoImpersonation</code> attribute.</p> <p><b>Windows 2000 Server:</b> This flag is not available.</p>
<p><code>msidbCustomActionTypeTSAware</code> + <code>msidbCustomActionTypeInScript</code> + <code>msidbCustomActionTypeRollback</code></p>	<p>Hexadecimal: 0x00000400 + 0x00004000 + 0x00000100</p> <p>Decimal: 17664</p> <p>Queues for execution at the scheduled point within script. Run only upon an installation rollback. Execute with user impersonation. Runs with user impersonation during per-machine installs on a terminal server.</p> <p><b>Windows 2000 Server:</b> This flag is not available.</p>
<p><code>msidbCustomActionTypeTSAware</code> + <code>msidbCustomActionTypeInScript</code> + <code>msidbCustomActionTypeCommit</code></p>	<p>Hexadecimal: 0x00000400 + 0x00004000 + 0x00000200</p> <p>Decimal: 17920</p> <p>Queues for execution at the scheduled point within script. Runs only upon an install commit. Executes with user impersonation. Runs with user impersonation during per-machine</p>

installs on a terminal server.

**Windows 2000 Server:** This flag is not available.

For information about custom actions that run only when a patch is being uninstalled, see the [Custom Action Patch Uninstall Option](#).

## See Also

[Custom Action Reference](#)  
[About Custom Actions](#)  
[Using Custom Actions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Patch Uninstall Option

Use the following option flag to specify that the installer run the custom action only when a patch is being uninstalled. To set the option, add the value in this table to the value in the ExtendedType field of the [CustomAction table](#).

**Windows Installer 4.0 and earlier:** Not supported. This option is available beginning with Windows Installer 4.5.

Constant	Hexadecimal	Decimal	Description
msidbCustomActionTypePatchUninstall	0x8000	32768	The custom action runs only when a patch is being uninstalled.

## Remarks

This attribute can be added to a custom action by authoring it in the Windows Installer package (.msi file). A new custom action with this attribute can be added by a patch. A custom action having this attribute can be updated by a patch. This attribute cannot be added or removed by a patch to an existing custom action.

If a patch adds or updates a custom action with this attribute, Windows Installer runs the new or updated custom action when the patch is uninstalled. Windows Installer makes the updates within the patch being uninstalled available to the patch uninstall custom action. The patch must include a [MsiTransformView<PatchGUID>](#) table to provide this information to Windows Installer.

When a package that contains a custom action with the msidbCustomActionTypePatchUninstall attribute is installed using an

installer version earlier than Windows Installer 4.0, the installer does not call the custom action when the patch is uninstalled. The install can run the custom action during the installation, repair, or update of the package.

Custom actions with the `msidbCustomActionTypePatchUninstall` attribute should be conditioned using the **MSIPATCHREMOVE** property to prevent the custom action from running when installing, repairing, or updating using a system with Windows Installer 4.0 or earlier. When Windows Installer 4.5 and later is installed, all the patches on the system having custom actions marked with the `msidbCustomActionTypePatchUninstall` attribute run the custom action during patch uninstallation. If Windows Installer 4.5 or later is removed from the system, patches lose the custom action patch uninstall functionality.

For information about running a custom action during the uninstallation of a patch using a version earlier than Windows Installer 4.5, see [Patch Uninstall Custom Actions](#).

## See Also

[Custom Action In-Script Execution Options](#)

[Custom Action Reference](#)

[About Custom Actions](#)

[Using Custom Actions](#)

[MsiTransformView<PatchGUID>](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiTransformView

This temporary table enables the [Custom Action Patch Uninstall Option](#) for custom actions added or updated by a patch.

If a patch adds or updates a custom action having the `msidbCustomActionTypePatchUninstall` attribute, Windows Installer runs the new or updated custom action when the patch is uninstalled. Windows Installer makes the updates within the patch being uninstalled available to the patch uninstall custom action. The patch must include a [MsiTransformView<PatchGUID>](#) table to provide this information to Windows Installer. The information in this table is available to any immediate custom action, and is unavailable to deferred custom actions.

**Windows Installer 4.0 and earlier:** Not supported. The Custom Action Patch Uninstall Option is available beginning with Windows Installer 4.5.

This table should be named [MsiTransformView<PatchGUID> Table](#), where `<PatchGUID>` is the GUID that uniquely identifies the patch. The [MsiTransformView<PatchGUID> Table](#) has the following columns.

Column	Type	Key	Nullable
Table	Identifier	Y	N
Column	Text	Y	N
Row	Text	Y	Y
Data	Text	N	Y
Current	Text	N	Y

## Column

Table

Name of an altered database table.

Column

Name of an altered table column or INSERT, DELETE, CREATE, or

DROP.

Row

A list of the primary key values separated by tabs. Null primary key values are represented by a single space character. A Null value in this column indicates a schema change.

Data

Data, name of a data stream, or a column definition.

Current

Current value from reference database, or column a number.

## Remarks

Patch uninstall custom actions run when the patch is uninstalled. They do not run when the product is uninstalled. Use the Custom Action Patch Uninstall Option and this table to run a custom only when the patch is being uninstalled.

A patch can update a custom action provided in the original package (.msi file.) To run the updated version of the custom action when the patch is uninstalled, mark the custom action with the msidbCustomActionTypePatchUninstall attribute in the original package.

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Return Processing Options

This topic identifies the option flags that you can use to control the processing of the custom action thread. The flags are used to specify that the main and custom action threads run synchronously (Windows Installer waits for the custom action thread to complete before resuming the main installation thread), or asynchronously (Windows Installer runs the custom action simultaneously while the main installation continues).

To enable the option flags, add the value that is identified in the following table to the value in the Type field of the [CustomAction Table](#).

Constant	Hexadecimal	Decimal	Description
(none)	0x00000000	+0	A synchronous exit if the exit code is 1. If the flag <code>msidbCustomActionTypeContinue</code> is not set, then the custom action must return one of the values that is described in <a href="#">Action Return Values</a> .
<code>msidbCustomActionTypeContinue</code>	0x00000040	+64	A synchronous exit that ignores exit code 1.
<code>msidbCustomActionTypeAsync</code>	0x00000080	+128	An asynchronous custom action that does not wait for exit code 1. This option cannot be used with <a href="#">Concurrent Installers</a> , <a href="#">Custom Actions</a> , or <a href="#">Custom Actions</a> .
<code>msidbCustomActionTypeAsync + msidbCustomActionTypeContinue</code>	0x00000040 + 0x00000080	+192	An asynchronous custom action that does not wait for exit code 1.

		<p>Execution continu Windows Installer</p> <p>This option can on the EXE type cust that is, <a href="#">executable</a></p> <p>All other types of can be asynchron the install sessi for the installation</p> <p>This option cannot <a href="#">Concurrent Install:</a></p>
--	--	--

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Custom Action Return Values

If the `msidbCustomActionTypeContinue` return processing option is not set, the custom action must return an integer status code as shown in the following table.

Return value	Description
<code>ERROR_FUNCTION_NOT_CALLED</code>	Action not executed.
<code>ERROR_SUCCESS</code>	Completed actions successfully.
<code>ERROR_INSTALL_USEREXIT</code>	User terminated prematurely.
<code>ERROR_INSTALL_FAILURE</code>	Unrecoverable error occurred.
<code>ERROR_NO_MORE_ITEMS</code>	Skip remaining actions, not an error.

Note that custom actions that are [executable files](#) must return a value of 0 for success. The installer interprets any other return value as failure. To ignore return values, set the `msidbCustomActionTypeContinue` bit flag in the `Type` field of the [CustomAction](#) table.

For more information about the `msidbCustomActionTypeContinue` option and other return processing options, see [Custom Action Return Processing Options](#).

Note that Windows Installer translates the return values from all actions when it writes the return value into the log file. For example, if the action return value appears as 1 in the log file, this means that the action returned `ERROR_SUCCESS`. For more information about this translation see [Logging of Action Return Values](#).

## See Also

[Error Codes](#)

[Logging of Action Return Values](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Return Values of JScript and VBScript Custom Actions

Custom actions written in JScript or VBScript can call an optional function. These functions must return one of the values shown in the following table.

Return value	Value	Description
msiDoActionStatusNoAction	0	Action not executed.
msiDoActionStatusSuccess	IDOK = 1	Action completed successfully.
msiDoActionStatusUserExit	IDCANCEL=2	Premature termination by user.
msiDoActionStatusFailure	IDABORT = 3	Unrecoverable error. Returned if there is an error during parsing or execution of the Jscript or VBScript.
msiDoActionStatusSuspend	IDRETRY = 4	Suspended sequence to be resumed later.
msiDoActionStatusFinished	IDIGNORE=5	Skip remaining actions. Not an error.

Note that Windows Installer translates the return values from all actions when it writes the return value into the log file. For example, if the action return value appears as 1 (one) in the log file, this means that the action returned `msiDoActionStatusSuccess`. For more information about this translation see [Logging of Action Return Values](#).

To return a value other than success from a script custom action, you must use a function target for the custom action. The target function is specified in the Target column of the [CustomAction Table](#).

The following script example shows you how to return success or failure from a VBScript custom action.



```
Function MyVBScriptCA()  
  
    If Session.Property("CustomErrorStatus") <> "0" Then  
        'return error  
        MyVBScriptCA = 3  
        Exit Function  
    End If  
  
    ' return success  
    MyVBScriptCA = 1  
    Exit Function  
  
End Function
```

If this VBScript were embedded in the [Binary table](#) of the installation package as MyCA.vbs, the CustomAction Table entry for the script would be the following:

Action	Type	Source	Target
MyCustomAction	6	MyCA.vbs	MyVBScriptCA

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Properties

Properties are global variables that Windows Installer uses during an installation. The following sections describe the properties used by the installer:

- [About Properties](#)
- [Using Properties](#)
- [Property Reference](#)

The term "property" also refers to an attribute of an automation object. See [Automation Interface](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# About Properties

Windows Installer can configure software installation by using the values of variables defined in an installation package or by the user.

Windows Installer uses three categories of global variables during an installation:

- **Private properties:** The installer uses private properties internally and their values must be authored into the installation database or set to values determined by the operating environment.
- **Public properties:** Public properties can be authored into the database and changed by a user or system administrator on the command line, by applying a transform, or by interacting with an authored user interface.
- **Restricted public properties:** For security purposes, the author of an installation package can restrict the public properties a user can change.

Not all properties need to be defined in every package, there is a small set of required properties that must be defined in every package. The installer sets the values of properties in a particular order of precedence.

[Private Properties](#)

[Public Properties](#)

[Restricted Public Properties](#)

[Required Properties](#)

[Order of Property Precedence](#)

## See Also

[Using Properties](#)

[Property Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Private Properties

Private properties are used internally by the installer and their values must be entered into the database by the author of the installation package or set by the installer during the installation to values determined by the operating environment. The only way a user can interact with private properties is through [Control Events](#) in the package's authored user interface. Private property names must include lowercase letters. See [Restrictions on Property Names](#).

Private properties commonly describe the operating environment. For example, if the installation is run on a Windows platform, the installer sets the [WindowsFolder](#) property to the value specified in the Property table.

Private property values cannot be overridden at a command line. To clear a private property from an installation, leave it out of the [Property table](#). On Windows XP, and Windows 2000 you cannot set a private property in the user interface phase of the installation and then pass the value to the execution phase.

For a list of all the standard private properties used by the installer, see [Property Reference](#). You can define a custom private property by entering the property's name and initial value in the Property table. Private property names must always include lowercase letters.

## See Also

[Public Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Public Properties

Public properties can be authored into the installation database in the same way as [private properties](#). In addition, the values of public properties can be changed by a user or system administrator by setting the property on the command line, by applying a transform, or by interacting with an authored user interface. Public property names cannot contain lowercase letters. See [Restrictions on Property Names](#).

Public properties are commonly set by users during the installation. For example, the public property **INSTALLLEVEL** property can be specified at the command line used to launch the installation or chosen by using an authored user interface.

Public property values can be overridden either at a command line, by using a [standard](#) or [custom](#) action, by applying a transform, or by having the user interact with an authored user interface. To clear a public property in the property table, leave it out of the table. Properties that are to be set by the user interface during the installation and then passed to the execution phase of the installation must be public.

For a list of the standard public properties used by the installer see [Property Reference](#). An author can also define a custom public property by entering the property's name and an initial value into the [Property table](#). All public properties can be overridden by all users if any of the following conditions are true.

- The user is a system administrator.
- The per-machine EnableUserControl policy is set to 1. See [System Policy](#).
- The [EnableUserControl](#) property is set to 1.
- This is an unmanaged installation that is not being done with elevated privileges.

If none of the above conditions are true, the installer defaults to limiting which public properties can be overridden by a user that is not a system administrator. See [Restricted Public Properties](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Restricted Public Properties

In the case of a managed installation, the package author may need to limit which [public properties](#) are passed to the server side and can be changed by a user that is not a system administrator. Some restrictions are commonly necessary to maintain a secure environment when the installation requires the installer to use elevated privileges. If all of the following conditions are met, a user that is not a system administrator can only override an approved list of restricted public properties:

- The system is Windows 2000.
- The user is not a system administrator.
- The application or product is being installed with elevated privileges.

If all of the above conditions are true, the installer defaults to the following list of restricted public properties that can be changed by any user:

- **ACTION**
- **AFTERREBOOT**
- **ALLUSERS**
- **EXECUTEACTION**
- **EXECUTEMODE**
- **FILEADDDEFAULT**
- **FILEADDLOCAL**
- **FILEADDSOURCE**
- **INSTALLLEVEL**
- **LIMITUI**
- **LOGACTION**
- **NOCOMPANYNAME**
- **NOUSERNAME**
- **MSIENFORCEUPGRADECOMPONENTRULES**
- **MSIINSTANCEGUID**

- **MSINEWINSTANCE**
- **MSIPATCHREMOVE**
- **PATCH**
- **PRIMARYFOLDER**
- **PROMPTROLLBACKCOST**
- **REBOOT**
- **REINSTALL**
- **REINSTALLMODE**
- **RESUME**
- **SEQUENCE**
- **SHORTFILENAME**
- **TRANSFORMS**
- **TRANSFORMSATSOURCE**

The author of an installation package can extend this default list to include additional public properties by using the **SecureCustomProperties** property.

Setting the **EnableUserControl** property or the **EnableUserControl system policy** extends the list to all public properties. All users can then change any public property.

The installer sets the **RestrictedUserControl** property whenever the list of public properties passed to the server by non-administrator users is restricted.

## See Also

[About Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Required Properties

There are five properties that are required for every installation.

- [ProductCode](#)
- [ProductLanguage](#)
- [Manufacturer](#)
- [ProductVersion](#)
- [ProductName](#)

## See Also

[About Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Order of Property Precedence

The installer sets properties using the following order of precedence. A property value in this list can override a value that comes after it and be overridden by a value coming before it in the list.

1. Properties specified by the operating environment.
2. [Public properties](#) set on the command line.
3. Public properties listed by the [AdminProperties](#) propertyset during an [administrative installation](#) .
4. Public or private properties set during the application of a [transform](#).
5. Public or private property that set by authoring the [Property table](#) of the .msi file.

## See Also

[About Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Properties

The following sections describe using the built-in installer properties and additional properties defined by the package author. Properties can be either [public properties](#) or [private properties](#). Every property that needs to be defined upon initialization of the installer must have its name and initial value listed in the [Property table](#). Properties having a Null value are not listed in the Property table. You can get or set properties from programs and custom actions and set public properties from the command line. The installation process can be modified by using properties in conditional statements.

[Restrictions on Property Names](#)

[Initialization of Property Values](#)

[Getting and Setting Properties](#)

[Using Properties in Conditional Statements](#)

[Using a Directory Property in a Path](#)

**Note** Do not use properties for passwords or other information that must remain secure. The installer may write the value of a property authored into the Property table or created at runtime into a log or the system registry.

## See Also

[About Properties](#)

[Property Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Restrictions on Property Names

All property names must follow these restrictions.

- A property name is an [Identifier](#), which is a text string that must begin with either a letter or an underscore. Identifiers and property names may contain letters, numerals, underscores, or periods; but cannot begin with a numeral or period.
- [Public property](#) names cannot contain lowercase letters.
- [Private property](#) names must contain some lowercase letters.
- Property names prefixed with % represent system and user environment variables. These are never entered into the [Property table](#). The permanent settings of environment variables can only be modified using the [Environment Table](#).

## See Also

[Using Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Initialization of Property Values

It is recommended that all properties to be used by the installation be entered into the [Property table](#) with an initial value. The installer sets the properties to these values at the launch of the installation. Properties for which a blank is an acceptable value and properties built into the installer do not need to be initialized.

## See Also

[Using Properties](#)

[Getting and Setting Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Getting and Setting Properties

To use [properties](#) in your installation, you can get and set property values from programs using [MsiGetProperty](#) and [MsiSetProperty](#) and include as part of conditional statements in the installation database.

- To obtain a current property, call the **MsiGetProperty** function.
- To obtain the current installation state, call the [MsiGetMode](#) function.
- To obtain the language for the current installation, call the [MsiGetLanguage](#) function.
- To set a property, call the **MsiSetProperty** function.
- To set the installation state, call the [MsiSetMode](#) function.
- To clear a property (setting it to Null), set the property's value to an empty string: "".

## See Also

[Using Properties](#)

[Setting Public Property Values on the Command Line](#)

[Clearing an Installer Property](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Setting Public Property Values on the Command Line

To set a public property to a literal string value, include the string between quotation marks.

```
PROPERTY = "string"
```

The quotation marks are only required if the string contains spaces. To clear a public property at a command line (setting it to Null), set the property's value to an empty string. In this case, quotation marks are required.

```
PROPERTY=""
```

## See Also

[Using Properties](#)

[Getting and Setting Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Clearing an Installer Property

To clear a property (setting it to Null), set the property's value to an empty string: "".

## See Also

[Using Properties](#)

[Getting and Setting Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Properties in Conditional Statements

The logical value of a property that has been set is True. To determine whether a property is set without actually getting its value, test the logical expression "MyProperty" or "Not MyProperty". When the property MyProperty is set, the former evaluates as True and the latter as False.

One or more properties can be combined with operators to form logical expressions used in a conditional statements. For more information about the operators that can be used in conditional statements, see [Conditional Statement Syntax](#).

A conditional statement using properties can be entered into the Condition column of the [Condition table](#) to modify the selection state of any entry in the [Feature table](#).

Conditional statements with one or more properties are commonly used in the Condition column of database tables.

The following tables each have a column for conditional expressions:

- [Condition table](#)
- [ControlEvent table](#)
- [LaunchCondition table](#)
- [InstallUISequence table](#)
- [InstallExecuteSequence table](#)
- [ControlCondition table](#)
- [AdminExecuteSequence table](#)
- [AdvtExecuteSequence table](#)
- [AdminUISequence table](#)

Note that the six action sequence tables have fields for a condition. If the conditional expression in this field evaluates to False, the installer skips that action.

If you set a [private property](#) in the UI sequence by authoring a custom

action in one of the user interface sequence tables, that property is not set in the execution sequence. To set the property in the execution sequence you must also put a custom action in an execution sequence table. Alternatively, you can make the property a [public property](#) and include it in the [SecureCustomProperties](#) property.

For more information, see [Using a Sequence Table](#) or [Using Properties](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using a Directory Property in a Path

The directories in the [Directory table](#) specify the layout of an installation. When the Windows Installer resolves these directories during the [CostFinalize action](#), the keys in the Directory table become [properties](#) set to directory paths. The installer also always sets a number of standard [System Folder Properties](#) to system folder paths.

The values of the [System Folder Properties](#) are guaranteed to end in a directory separator. The values of all other properties entered in the Directory table are only guaranteed to end in a directory separator after the installer has run the CostFinalize action. Before costing has completed, the values of properties entered in the Directory table which are not [System Folder Properties](#) may not end in a directory separator. Therefore, if your installation sets directory properties using [custom actions](#) in the package, the values on reference might not end with a directory separator.

Directory properties ending with a directory separator can therefore be used in a path string without explicitly including the directory separator. For example, if the value of DirectoryProperty ends with a directory separator, the following string correctly specifies the path to *file* in *subdirectory*

```
[DirectoryProperty]subdirectory\file
```

and the following path string is incorrect.

```
[DirectoryProperty]\subdirectory\file
```

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Property Reference

This section lists the properties defined by Windows Installer:

- [Component Location Properties](#)
- [Configuration Properties](#)
- [Date, Time Properties](#)
- [Feature Installation Options Properties](#)
- [Hardware Properties](#)
- [Installation Status Properties](#)
- [Operating System Properties](#)
- [Product Information Properties](#)
- [Summary Information Update Properties](#)
- [System Folder Properties](#)
- [User Information Properties](#)

Additional properties can be specified by authored data or custom actions. Properties with names containing no lowercase letters are public properties and can be specified on the command line.

For information about values of the **Uninstall** registry key that are provided by installer properties, see [Uninstall Registry Key](#).

## Component Location Properties

The following list provides links to more information about the component location properties.

Property	Description
<a href="#">OriginalDatabase</a>	The installer sets this property to the launched-from database, the database on the source, or the cached database.
<a href="#">ParentOriginalDatabase</a>	The installer sets this property for installations run by a <a href="#">Concurrent Installation</a> action.

<b>SourceDir</b>	Root directory that contains the source files.
<b>TARGETDIR</b>	Specifies the root destination directory for the installation. During an <a href="#">administrative installation</a> this property is the location to copy the installation package.

## Configuration Properties

The following list provides links to more information about other configurable properties.

<b>Property</b>	<b>Description</b>
<b><a href="#">ACTION</a></b>	Initial action called after installer is initialized.
<b><a href="#">ALLUSERS</a></b>	Determines where configuration information is stored.
<b><a href="#">ARPAUTHORIZEDCDFPREFIX</a></b>	URL of the update channel application.
<b><a href="#">ARPCOMMENTS</a></b>	Provides Comments for <b>or Remove Programs in Panel</b> .
<b><a href="#">ARPCONTACT</a></b>	Provides Contact for the <b>Remove Programs in C Panel</b> .
<b><a href="#">ARPINSTALLLOCATION</a></b>	Fully qualified path to the folder of an application.
<b><a href="#">ARPNOMODIFY</a></b>	Disables functionality that modifies a product.
<b><a href="#">ARPNOREMOVE</a></b>	Disables functionality that removes a product.
<b><a href="#">ARPNOREPAIR</a></b>	Disables the <b>Repair</b> but Programs wizard.

<b>ARPPRODUCTICON</b>	Specifies the primary icon installation package.
<b>ARPREADME</b>	Provides a <b>ReadMe</b> for <b>Add or Remove Programs</b> in <b>Add or Remove Programs Panel</b> .
<b>ARPSIZE</b>	Estimated size of an application in kilobytes.
<b>ARPSYSTEMCOMPONENT</b>	Prevents display of an application in the <b>Add or Remove Programs</b> list.
<b>ARPURLINFOABOUT</b>	URL for the home page of the application.
<b>ARPURLUPDATEINFO</b>	URL for application update information.
<b>AVAILABLEFREEREG</b>	Registry space (in kilobytes) that an application requires. <a href="#">AllocateRegistrySpace</a> and <a href="#">FreeRegistrySpace</a> are used to manage this space.
<b>CCP_DRIVE</b>	The root path for qualified products for CCP.
<b>DefaultUIFont</b>	Default font style used for controls.
<b>DISABLEADVTSHORTCUTS</b>	Set to disable the general application-specific shortcuts that support <a href="#">installation-on-demand</a> .
<b>DISABLEMEDIA</b>	Prevents the installer from registering media sources such as CD-ROMs, as valid sources for the product.
<b>DISABLEROLLBACK</b>	Disables rollback for the configuration.
<b>EXECUTEACTION</b>	Top-level action that <code>ExecuteAction</code> initiates.

<b>EXECUTEMODE</b>	Mode of execution that the installer performs.
<b>FASTOEM</b>	Improves installation performance under specific OEM scenarios.
<b>INSTALLLEVEL</b>	Initial level where features are installed.
<b>LIMITUI</b>	UI level capped as Basic.
<b>LOGACTION</b>	List of action names to be logged.
<b>MEDIAPACKAGEPATH</b>	This property must be set to a relative path if the installation package is not located at the root of the CD-ROM.
<b>MSIARPSETTINGSIDENTIFIER</b>	This optional property contains a semi-colon delimited list of registry locations where the application stores a user's settings and preferences. Available in Windows Installer 4.0.
<b>MSIDISABLEEEEUI</b>	Disable the embedded user interface for the installation.  <b>Windows Installer 4.0 and earlier:</b> Not supported.
<b>MSIFASTINSTALL</b>	Reduce the time required to install a large Windows Installer package.  <b>Windows Installer 4.0 and earlier:</b> Not supported.
<b>MSIINSTALLPERUSER</b>	Requests that the Windows Installer install the package for the current user.  <b>Windows Installer 4.0 and earlier:</b> Not supported.

	<b>earlier:</b> Not supported
<b>MSINODISABLEMEDIA</b>	Set this property to prevent the installer from setting the <b>DISABLEMEDIA</b> property.
<b>MSIENFORCEUPGRADECOMPONENTRULES</b>	Set this property to 1 (or 0) on the command line or in the <b>Table</b> to apply the upgrade component rules during updates and minor upgrades for a specific product. Available beginning with Windows 3.0.
<b>MSIUNINSTALLSUPERSEDEDCOMPONENTS</b>	When this property has a value of 1, the installer can unregister and uninstall redundant components to prevent leaving behind orphaned components on the computer.  <b>Windows Installer 4.0 and earlier:</b> Not supported
<b>PRIMARYFOLDER</b>	Allows the author to designate a primary folder for an installation. Used to determine the volume for the <b>PrimaryVolumePath</b> , <b>PrimaryVolumeSpace</b> , <b>PrimaryVolumeSpaceId</b> , and <b>PrimaryVolumeSpaceId</b> properties.
<b>Privileged</b>	Runs an installation with elevated privileges.
<b>PROMPTROLLBACKCOST</b>	Action if there is insufficient space for the installation.
<b>REBOOT</b>	Forces or suppresses a reboot.

<b>REBOOTPROMPT</b>	Suppresses the display of a message for restarts to the user. All actions that are needed happen automatically.
<b>ROOTDRIVE</b>	Default drive for an installation.
<b>SEQUENCE</b>	A table that has the sequence and schema.
<b>SHORTFILENAMES</b>	Causes short file names.
<b>TRANSFORMS</b>	List of transforms to be applied to a database.
<b>TRANSFORMSATSOURCE</b>	Informs the installer that the transforms for a product are from the source.
<b>TRANSFORMSSECURE</b>	Setting the <b>TRANSFORMSECURE</b> property to 1 (one) informs the installer that transforms are cached locally on the user's computer in a location where the user does not have write access.
<b>MsiLogFileLocation</b>	The installer sets the value of the <b>MsiLogFileLocation</b> property to the full path to the log file, when logging has been enabled. This property is available starting with Windows Installer 4.0.
<b>MsiLogging</b>	Sets the default logging level for the Windows Installer package. This property is available starting with Windows Installer 4.0.
<b>MSIUSEREALADMINDETECTION</b>	Set this property to 1 to inform the installer that the user is an administrator so that the installer use actual user information when setting the <b>AdminUser</b> property. This property is available starting with Windows Installer 4.0.

property is available starting with Windows Installer 4.0.

## Date, Time Properties

The **Date** and **Time** properties are live properties that the installer sets when data is extracted.

Property	Description
<b>Date</b>	The current date.
<b>Time</b>	The current time.

## Feature Installation Options Properties

The following list provides links to more information about the feature installation options properties.

Property	Description
<b>ADDDEFAULT</b>	List of features to be installed in the default configuration.
<b>ADDLOCAL</b>	List of features to be installed locally.
<b>ADDSOURCE</b>	List of features to be run from source.
<b>ADVERTISE</b>	List of features to be advertised.
<b>COMPADDDEFAULT</b>	List of components to be installed in the default configuration.
<b>COMPADDLOCAL</b>	List of component IDs to be installed locally.
<b>COMPADDSOURCE</b>	List of component IDs to run from source media.

<b>FILEADDDEFAULT</b>	List of file keys for files to be installed in the default configuration.
<b>FILEADDLOCAL</b>	List of file keys for files to be run locally.
<b>FILEADDSOURCE</b>	List of file keys to be run from the source media.
<b>MSIDISABLELUAPATCHING</b>	Setting this property prevents Least Privileged User (LUA) patching of an application.
<b>MsiPatchRemovalList</b>	List of patches to be removed during the installation.
<b>MSIRESTARTMANAGERCONTROL</b>	Specifies whether the package uses the Restart Manager or FilesInUse functionality.
<b>MSIDISABLELRMRESTART</b>	Specifies how applications or services that are currently using files affected by an update should be shutdown and restarted to enable the installation of the update.
<b>MSIRMSHUTDOWN</b>	Specifies how applications or services that are currently using files affected by an update should be shutdown to enable the installation of the update.
<b>MSIPATCHREMOVE</b>	Setting this property removes patches.
<b>PATCH</b>	Setting this property applies a patch.
<b>REINSTALL</b>	List of features to be reinstalled.
<b>REINSTALLMODE</b>	A string that contains letters that specify the type of reinstall to

	perform.
<b>REMOVE</b>	List of features to be removed.

## Hardware Properties

The following list identifies the hardware properties that the Windows Installer sets at startup.

<b>Property</b>	<b>Description</b>
<b>Alpha</b>	The numeric processor level when running on an Alpha processor. <b>Note</b> This property is obsolete, the Alpha platform is not supported by Windows Installer.
<b>BorderSide</b>	The width of the window borders, in pixels.
<b>BorderTop</b>	The height of the window borders, in pixels.
<b>CaptionHeight</b>	The height of normal caption area, in pixels.
<b>ColorBits</b>	The number of adjacent color bits for each pixel.
<b>Intel</b>	The numeric processor level when running on an Intel processor.
<b>Intel64</b>	The numeric processor level when running on an Itanium processor.
<b>Msix64</b>	The numeric processor level when running on an x64 processor.
<b>PhysicalMemory</b>	The size of the installed RAM, in megabytes.
<b>ScreenX</b>	The width of the screen, in pixels.
<b>ScreenY</b>	The height of the screen, in pixels.
<b>TextHeight</b>	The height of characters, in logical units.
<b>VirtualMemory</b>	The amount of available page file space, in megabytes.

## Installation Status Properties

The following list provides links to more information about status properties that are updated by the installer during installation.

Property	Description
<a href="#">AFTERREBOOT</a>	Indicates current installation follows a reboot that the <a href="#">ForceReboot action</a> invokes.
<a href="#">CostingComplete</a>	Indicates whether disk space costing is complete.
<a href="#">Installed</a>	Indicates that a product is already installed.
<a href="#">MSICHECKCRCS</a>	The Installer does a CRC on files only if the <a href="#">MSICHECKCRCS</a> property is set.
<a href="#">MsiRestartManagerSessionKey</a>	The Installer sets this property to the session key for the Restart Manager session.
<a href="#">MsiRunningElevated</a>	The Installer sets the value of this property to 1 when the installer is running with <i>elevated</i> privileges.
<a href="#">MsiSystemRebootPending</a>	The Installer sets this property to 1 if a restart of the operating system is currently pending.
<a href="#">MsiUIHideCancel</a>	The Installer sets <a href="#">MsiUIHideCancel</a> to 1 when the internal install level includes <code>INSTALLUILEVEL_HIDECANCEL</code> .
<a href="#">MsiUIProgressOnly</a>	The Installer sets <a href="#">MsiUIProgressOnly</a> to 1 when the internal install level includes <code>INSTALLUILEVEL_PROGRESSONLY</code> .
<a href="#">MsiUISourceResOnly</a>	<a href="#">MsiUISourceResOnly</a> to 1 (one) when the internal install level includes <code>INSTALLUILEVEL_SOURCERESONLY</code> .
<a href="#">NOCOMPANYNAME</a>	Suppresses the automatic setting of the <a href="#">COMPANYNAME</a> property.

<b>NOUSERNAME</b>	Suppresses the automatic setting of the <b>USERNAME</b> property.
<b>OutOfDiskSpace</b>	Insufficient disk space to accommodate the installation.
<b>OutOfNoRbDiskSpace</b>	Insufficient disk space with rollback turned off.
<b>Preselected</b>	Features are already selected.
<b>PrimaryVolumePath</b>	The Installer sets the value of this property to the path of the volume that the <b>PRIMARYFOLDER</b> property designates.
<b>PrimaryVolumeSpaceAvailable</b>	The Installer sets the value of this property to a string that represents the total number of bytes available on the volume that the <b>PrimaryVolumePath</b> property references.
<b>PrimaryVolumeSpaceRemaining</b>	The Installer sets the value of this property to a string that represents the total number of bytes remaining on the volume that the <b>PrimaryVolumePath</b> property references if all the currently selected features are installed.
<b>PrimaryVolumeSpaceRequired</b>	The Installer sets the value of this property to a string that represents the total number of bytes required by all currently selected features on the volume that the <b>PrimaryVolumePath</b> property references.
<b>ProductLanguage</b>	Numeric language identifier (LANGID) for the database. (REQUIRED)
<b>ReplacedInUseFiles</b>	Set if the installer installs over a file that is being held in use.
<b>RESUME</b>	Resumed installation.
<b>RollbackDisabled</b>	The installer sets this property when rollback is disabled.

<b>UILevel</b>	Indicates the user interface level.
<b>UpdateStarted</b>	Set when changes to the system have begun for this installation.
<b>UPGRADINGPRODUCTCODE</b>	Set by the installer when an upgrade removes an application.
<b>VersionMsi</b>	The installer sets this property to the version of Windows Installer that is run during the installation.

## Operating System Properties

The following list provides links to more information about operating system properties that the Installer sets at startup.

<b>Property Name</b>	<b>Brief Description</b>
<b>AdminUser</b>	Set on Windows 2000 if the user has administrator privileges.
<b>ComputerName</b>	Computer name of the current system.
<b>MsiNetAssemblySupport</b>	On systems that support common language runtime assemblies, the Installer sets the value of this property to the file version of fusion.dll. The Installer does not set this property if the operating system does not support common language runtime assemblies.
<b>MsiNTProductType</b>	Indicates the Windows product type.
<b>MsiNTSuiteBackOffice</b>	On Windows 2000 and later operating systems, the Installer sets this property to 1 (one) only if Microsoft BackOffice components are installed.

<b>MsiINTSuiteDataCenter</b>	On Windows 2000 and later operating systems, the Installer sets this property to 1 (one) only if Windows 2000 Datacenter Server is installed.
<b>MsiINTSuiteEnterprise</b>	On Windows 2000 and later operating systems, the Installer sets this property to 1 (one) only if Windows 2000 Advanced Server is installed.
<b>MsiINTSuitePersonal</b>	On Windows XP and later operating systems, the Installer sets this property to 1 (one) only if the operating system is Workstation Personal (not Professional).
<b>MsiINTSuiteSmallBusiness</b>	On Windows 2000 and later operating systems, the Installer sets this property to 1 (one) only if Microsoft Small Business Server is installed.
<b>MsiINTSuiteSmallBusinessRestricted</b>	On Windows 2000 and later operating systems, the Installer sets this property to 1 (one) only if Microsoft Small Business Server is installed with the restrictive client license.
<b>MsiINTSuiteWebServer</b>	On Windows 2000 and later operating systems, the Installer sets the <b>MsiINTSuiteWebServer</b> property to 1 (one) if the web edition of the Windows Server 2003 is installed. Only available with the Windows Server 2003 release of the Windows Installer.
<b>MsiTabletPC</b>	The installer sets this property to a nonzero value if the current

	operating system is Windows XP Tablet PC Edition.
<b>MsiWin32AssemblySupport</b>	On systems that support Win32 assemblies, the Installer sets the value of this property to the file version of sxs.dll. The Installer does not set this property if the operating system does not support Win32 assemblies.
<b>OLEAdvtSupport</b>	Set if OLE supports the Windows Installer.
<b>RedirectedDllSupport</b>	The Installer sets the <b>RedirectedDllSupport</b> property if the system performing the installation supports <a href="#">Isolated Components</a> .
<b>RemoteAdminTS</b>	The Installer sets the <b>RemoteAdminTS</b> property when the system is a remote administration server running the Terminal Server role service.
<b>ServicePackLevel</b>	The version number of the operating system service pack.
<b>ServicePackLevelMinor</b>	The minor version number of the operating system service pack.
<b>SharedWindows</b>	Set when the system is operating as Shared Windows.
<b>ShellAdvtSupport</b>	Set if the shell supports feature advertising.
<b>SystemLanguageID</b>	Default language identifier for the system.
<b>TerminalServer</b>	Set when the system is a server running the Terminal Server role service.

<b>TTCSupport</b>	Indicates if the operating system supports using .ttc (true type font collections) files.
<b>Version9X</b>	Version number for the Windows operating system.
<b>VersionDatabase</b>	Numeric database version of the current installation.
<b>VersionNT</b>	Version number for the operating system.
<b>VersionNT64</b>	Version number for the operating system if the system is running on a 64-bit computer.
<b>Windows build</b>	Build number of the operating system.

## Product Information Properties

The following list provides links to more information about product-specific properties specified in the [Property Table](#).

<b>Property Name</b>	<b>Brief Description</b>
<b>ARPHHELPLINK</b>	Internet address or URL for technical support.
<b>ARPHHELPTELEPHONE</b>	Technical support phone numbers.
<b>DiskPrompt</b>	String displayed by a message box that prompts for a disk.
<b>IsAdminPackage</b>	Set to 1 (one) if the current installation is running from a package created through an administrative installation.
<b>LeftUnit</b>	Places units to the left of the number.
<b>Manufacturer</b>	Name of the application manufacturer. (Required)

<b>MediaSourceDir</b>	The installer sets this property to 1 (one) when the installation uses a media source, such as a CD-ROM.
<b>MSIINSTANCEGUID</b>	The presence of this property indicates that a product code changing transform is registered to the product.
<b>MSINEWINSTANCE</b>	This property indicates the installation of a new instance of a product with instance transforms.
<b>ParentProductCode</b>	The installer sets this property for installations that a Concurrent Installation action runs.
<b>PIDTemplate</b>	String used as a template for the <b>PIDKEY</b> property.
<b>ProductCode</b>	A unique identifier for a specific product release. (Required)
<b>ProductName</b>	Human readable name of an application. (Required)
<b>ProductState</b>	Set to the installed state of a product.
<b>ProductVersion</b>	String format of the product version as a numeric value. (Required)
<b>UpgradeCode</b>	A GUID that represents a related set of products.

## Summary Information Update Properties

The following properties are only set by transforms in .msp files that are used to update the summary information stream of an administrative image.

<b>Property</b>	<b>Description</b>
<b>PATCHNEWPACKAGECODE</b>	The value of this property is written to the <b>Revision Number Summary</b> Property.

<b>PATCHNEWSUMMARYCOMMENTS</b>	The value of this property is written to the <b>Comments Summary</b> Property.
<b>PATCHNEWSUMMARYSUBJECT</b>	The value of this property is written to the <b>Subject Summary</b> Property.

## System Folder Properties

The following list provides links to more information about system folders that the installer sets at setup.

Property	Description
<b>AdminToolsFolder</b>	The full path to the directory that contains administrative tools.
<b>AppDataFolder</b>	The full path to the <b>Roaming</b> folder for the current user.
<b>CommonAppDataFolder</b>	The full path to application data for all users.
<b>CommonFiles64Folder</b>	The full path to the predefined <b>64-bit Common Files</b> folder.
<b>CommonFilesFolder</b>	The full path to the <b>Common Files</b> folder for the current user.
<b>DesktopFolder</b>	The full path to the <b>Desktop</b> folder.
<b>FavoritesFolder</b>	The full path to the <b>Favorites</b> folder for the current user.
<b>FontsFolder</b>	The full path to the <b>Fonts</b> folder.
<b>LocalAppDataFolder</b>	The full path to the folder that contains local (nonroaming) applications.
<b>MyPicturesFolder</b>	The full path to the <b>Pictures</b> folder.
<b>NetHoodFolder</b>	The full path to the <b>NetHood</b> folder.
<b>PersonalFolder</b>	The full path to the <b>Documents</b> folder for the

	current user.
<b>PrintHoodFolder</b>	The full path to the <b>PrintHood</b> folder.
<b>ProgramFiles64Folder</b>	The full path to the predefined <b>64-bit Program Files</b> folder.
<b>ProgramFilesFolder</b>	The full path to the predefined <b>32-bit Program Files</b> folder.
<b>ProgramMenuFolder</b>	The full path to the <b>Program Menu</b> folder.
<b>RecentFolder</b>	The full path to the <b>Recent</b> folder.
<b>SendToFolder</b>	The full path to the <b>SendTo</b> folder for the current user.
<b>StartMenuFolder</b>	The full path to the <b>Start menu</b> folder.
<b>StartupFolder</b>	The full path to the <b>Startup</b> folder.
<b>System16Folder</b>	The full path to folder for 16-bit system DLLs.
<b>System64Folder</b>	The full path to the predefined <b>System64</b> folder.
<b>SystemFolder</b>	The full path to the <b>System</b> folder for the current user.
<b>TempFolder</b>	The full path to the <b>Temp</b> folder.
<b>TemplateFolder</b>	The full path to the <b>Template</b> folder for the current user.
<b>WindowsFolder</b>	The full path to the <b>Windows</b> folder.
<b>WindowsVolume</b>	The volume of the <b>Windows</b> folder.

## User Information Properties

The following list provides links to more information about user-supplied information.

<b>Property</b>	<b>Description</b>
<b>AdminProperties</b>	List of properties that are set during an administration installation.

<b>COMPANYNAME</b>	Organization name of the user who is performing the installation.
<b>LogonUser</b>	User name for the user who is currently logged on.
<b>MsiHiddenProperties</b>	List of properties that are prevented from being written into the log.
<b>PIDKEY</b>	Part of the Product ID that the user enters.
<b>ProductID</b>	Full Product ID after a successful validation.
<b>UserLanguageID</b>	Default language identifier of the current user.
<b>USERNAME</b>	User who is performing the installation.
<b>UserSID property</b>	Set by the installer according to the security identifier (SID) of the user.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ACTION Property

The **ACTION** property can be set to the following values.

## Value

Value	Meaning
INSTALL	<a href="#">INSTALL Action</a>
ADVERTISE	<a href="#">ADVERTISE Action</a>
ADMIN	<a href="#">ADMIN Action</a>

The **ACTION** property determines which action to perform if a Null action name is supplied to [MsiDoAction](#) or the [DoAction Method](#). If no value is defined for the **ACTION** property, the installer calls the [INSTALL Action](#).

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ADDDEFAULT Property

The value of the **ADDDEFAULT** property is a list of features delimited by commas, which are to be installed in their default configuration. The features must be present in the Feature column of the [Feature Table](#). To install all features in their default configurations, use **ADDDEFAULT=ALL** on the command line.

A feature listed in the **ADDDEFAULT** property is installed in the same installation state as if the user requested an installation-on-demand of the feature. The state is determined by the bits that are set for the feature in the Attributes column of the [Feature Table](#), and which bits are set for the feature components in the Attributes column of the [Component Table](#).

## Remarks

The feature names are case sensitive.

The installer always evaluates the following properties in the following order:

1. **ADDLOCAL**
2. **REMOVE**
3. **ADDSOURCE**
4. **ADDDEFAULT**
5. **REINSTALL**
6. **ADVERTISE**
7. **COMPADDLOCAL**
8. **COMPADDSOURCE**
9. **COMPADDDEFAULT**
10. **FILEADDLOCAL**
11. **FILEADDSOURCE**
12. **FILEADDDEFAULT**

For example:

- If the command line specifies: ADDLOCAL=ALL, ADDSOURCE = MyFeature, all the features are first set to run-local and then **MyFeature** is set to run-from-source.
- If the command line is: ADDSOURCE=ALL, ADDLOCAL=MyFeature, first **MyFeature** is set to run-local, and then when ADDSOURCE=ALL is evaluated, all features (including **MyFeature**) are reset to run-from-source.

The installer sets the **Preselected** property to a value of "1" during the resumption of a suspended installation, or when any of the above properties are specified on the command line.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ADDLOCAL Property

The value of the **ADDLOCAL** property is a list of features that are delimited by commas, and are to be installed locally. The features must be present in the Feature column of the [Feature Table](#). To install all features locally, use ADDLOCAL=ALL on the command line. Do not enter ADDLOCAL=ALL into the [Property Table](#), because this generates a locally installed package that cannot be correctly removed.

## Remarks

Feature names are case sensitive. If the SourceOnly bit flag is set in the Attributes column of the [Component Table](#) for a component of a feature in the list, that component is installed as run from source.

The installer always evaluates the following properties in the following order:

1. **ADDLOCAL**
2. **REMOVE**
3. **ADDSOURCE**
4. **ADDDEFAULT**
5. **REINSTALL**
6. **ADVERTISE**
7. **COMPADDLOCAL**
8. **COMPADDSOURCE**
9. **COMPADDDEFAULT**
10. **FILEADDLOCAL**
11. **FILEADDSOURCE**
12. **FILEADDDEFAULT**

For example:

- If the command line specifies: ADDLOCAL=ALL, ADDSOURCE =

**MyFeature**, all the features are first set to run-local and then **MyFeature** is set to run-from-source.

- If the command line is: ADDSOURCE=ALL, ADDLOCAL=**MyFeature**, first **MyFeature** is set to run-local, and then when ADDSOURCE=ALL is evaluated, all features (including **MyFeature**) are reset to run-from-source.

The installer sets the **Preselected** Property to a value of "1" during the resumption of a suspended installation, or when any of the previous properties are specified on the command line.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ADDSOURCE Property

The value of the **ADDSOURCE** property is a list of features that are delimited by commas, and are to be installed to run from the source. The features must be present in the Feature column of the [Feature Table](#). To install all features as run from source, use **ADDSOURCE=ALL** on the command line. Do not enter **ADDSOURCE=ALL** into the [Property Table](#), because this generates a run-from-source package that cannot be correctly removed.

## Remarks

The feature names are case sensitive. If the LocalOnly bit flag is set in the Attributes column of the [Component Table](#) for a component of a feature in the list, that component is installed to run locally.

The installer always evaluates the following properties in the following order:

1. **ADDLOCAL**
2. **REMOVE**
3. **ADDSOURCE**
4. **ADDDEFAULT**
5. **REINSTALL**
6. **ADVERTISE**
7. **COMPADDLOCAL**
8. **COMPADDSOURCE**
9. **COMPADDDEFAULT**
10. **FILEADDLOCAL**
11. **FILEADDSOURCE**
12. **FILEADDDEFAULT**

For example:

- If the command line specifies: **ADDLOCAL=ALL, ADDSOURCE =**

**MyFeature**, all the features are first set to run-local and then **MyFeature** is set to run-from-source.

- If the command line is: ADDSOURCE=ALL, ADDLOCAL=**MyFeature**, first **MyFeature** is set to run-local, and then when ADDSOURCE=ALL is evaluated, all features (including **MyFeature**) are reset to run-from-source.

The installer sets the **Preselected** Property to a value of "1" during the resumption of a suspended installation, or when any of the above properties are specified on the command line.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# AdminProperties Property

The **AdminProperties** should be authored into the [Property Table](#). The value of **AdminProperties** is a list of property names separated by semicolons. The installer saves the values of these listed properties at the time of an [administrative installation](#). When users install from the administrative image, the installation uses the saved values of the properties, rather than the values in the .msi file.

## Remarks

The property names in the list can be uppercase and lowercase letters (private properties), or all uppercase (public properties).

This property must never end with a semicolon.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# AdminToolsFolder Property

The **AdminToolsFolder** property contains the full path to the file system directory that stores administrative tools. Microsoft Management Console (MMC) saves customized consoles to this directory and roams with the user. If the **ALLUSERS** Property is set, this property points to the file system directory that contains the administrative tools for all users of the computer.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# AdminUser Property

The installer sets this property if the user has administrator privileges.

**Windows Server 2008 and Windows Vista:** The **AdminUser** property is the same as the **Privileged** property. Authors should use the **Privileged** property. The installer sets these properties if the user has administrator privileges, if the application has been assigned by a system administrator, or if both the user and machine policies `AlwaysInstallElevated` are set to true.

## Remarks

The differences between these properties may have been used in some legacy packages. For example, **AdminUser** may have been used instead of **Privileged** in conditional statements, because the installer only sets the **AdminUser** property if the user is an administrator. The installer sets the **Privileged** property if the user is an administrator, or if policy enables the user to install with elevated privileges.

**Windows Server 2008 and Windows Vista:** Not supported. The **Privileged** and **AdminUser** are the same. Packages that require distinct **Privileged** and **AdminUser** properties can restore the difference by setting the **MSIUSEREALADMINDETECTION** property.

For more information, see [Installing a Package with Elevated Privileges for a Non-Admin](#), and **Privileged** property.

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Vista or Windows Server 2008. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service

**Version**

pack that is required by a Windows Installer version.

**See Also**

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ADVERTISE Property

The value of the **ADVERTISE** property is a list of features delimited by commas that are to be advertised. The features must be present in the Feature column of the [Feature](#) table. To install all features as advertised, use ADVERTISE=ALL on the command line. Do not enter ADVERTISE=ALL into the [Property table](#) because this generates an advertised package that cannot be installed or removed.

## Remarks

Note that feature names are case-sensitive.

The installer always evaluates the following properties in the following order.

1. **ADDLOCAL**
2. **REMOVE**
3. **ADDSOURCE**
4. **ADDDEFAULT**
5. **REINSTALL**
6. **ADVERTISE**
7. **COMPADDLOCAL**
8. **COMPADDSOURCE**
9. **COMPADDDEFAULT**
10. **FILEADDLOCAL**
11. **FILEADDSOURCE**
12. **FILEADDDEFAULT**

For example, if the command line specifies: ADDLOCAL=ALL, ADDSOURCE = MyFeature, all the features are first set to run-local and then MyFeature is set to run-from-source. If the command line is: ADDSOURCE=ALL, ADDLOCAL=MyFeature, first MyFeature is set to run-local, and then when ADDSOURCE=ALL is evaluated, all features (including MyFeature) are reset to run-from-source.

The installer sets the **Preselected** property to a value of "1" during the resumption of a suspended installation or when any of the above properties are specified on the command line.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# AFTERREBOOT Property

The installer sets the **AFTERREBOOT** property to 1 after a reboot invoked by the [ForceReboot action](#). The installer adds AFTERREBOOT=1 to the command line run immediately after the reboot.

## Remarks

The [ForceReboot action](#) must always be used with a conditional statement such that the installer triggers a reboot only when necessary. A reboot might be required if a particular file was replaced or a particular component was installed. The case is different for each product and a custom action might be needed to determine whether a reboot is needed. The condition on the ForceReboot action commonly makes use of the **AFTERREBOOT** property.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)  
[System Reboots](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ALLUSERS Property

The **ALLUSERS** property configures the installation context of the package. The Windows Installer performs a per-user installation or per-machine installation depending on the access privileges of the user, whether elevated privileges are required to install the application, the value of the **ALLUSERS** property, the value of the **MSIINSTALLPERUSER** property and the version of the operating system.

The value of the **ALLUSERS** property, at installation time, determines the [installation context](#).

- An **ALLUSERS** property value of 1 specifies the per-machine installation context.
- An **ALLUSERS** property value of an empty string ("" ) specifies the per-user installation context.
- If the value of the **ALLUSERS** property is set to 2, the Windows Installer always resets the value of the **ALLUSERS** property to 1 and performs a per-machine installation or it resets the value of the **ALLUSERS** property to an empty string ("" ) and performs a per-user installation. The value **ALLUSERS=2** enables the system to reset the value of **ALLUSERS**, and the installation context, dependent upon the user's privileges and the version of Windows.

**Windows 7:** Set the **ALLUSERS** property to 2 to use the **MSIINSTALLPERUSER** property to specify the installation context. Set the **MSIINSTALLPERUSER** property to an empty string ("" ) for a per-machine installation. Set the **MSIINSTALLPERUSER** property to 1 for a per-user installation. If the package has been written following the development guidelines described in [Single Package Authoring](#), users having user access can install into the per-user context without having to provide UAC credentials. If the user has user access

privileges, the installer performs a per-machine installation only if Admin credentials are provided to the UAC dialog box.

**Windows Vista:** Set the **ALLUSERS** property to 2 and Windows Installer complies with *User Account Control* (UAC). If the user has user access privileges, and ALLUSERS=2, the installer performs a per-machine installation only if Admin credentials are provided to the UAC dialog box. If UAC is enabled and the correct Admin credentials are not provided, the installation fails with an error stating that administrator privileges are required. If UAC is disabled by the registry key, group policy, or the control panel, the UAC dialog box is not displayed and the installation fails with an error stating that administrator privileges are required.

**Windows XP:** Set the **ALLUSERS** property to 2 and Windows Installer performs a per-user installation if the user has user access privileges.

**Windows 2000:** Set the **ALLUSERS** property to 2 and Windows Installer performs a per-machine installation if the user has administrative access privileges on the computer. If the user does not have administrative access privileges, Windows Installer resets the value of the **ALLUSERS** property to an empty string ("") and performs a per-user installation.

- If the value of the **ALLUSERS** property does not equal 2, the Windows Installer ignores the value of the **MSIINSTALLPERUSER** property.

## Default Value

The recommended default installation context is per-user. If **ALLUSERS** is not set, the installer does a per-user installation. You can ensure the

**ALLUSERS** property has not been set by setting its value to an empty string (""), ALLUSERS="".

## Remarks

The [installation context](#) determines the values of the [DesktopFolder](#), [ProgramMenuFolder](#), [StartMenuFolder](#), [StartupFolder](#), [TemplateFolder](#), [AdminToolsFolder](#), [ProgramFilesFolder](#), [CommonFilesFolder](#), [ProgramFiles64Folder](#), and [CommonFiles64Folder](#) properties. The installation context determines the parts of the registry where entries in the [Registry table](#) and [RemoveRegistry table](#), with -1 in the Root column, are written or removed.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[MSIINSTALLPERUSER](#)

[Installation Context](#)

[Single Package Authoring](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Alpha Property

The **Alpha** property is defined only if running on an Alpha processor. The installer sets it to the value of the numeric processor level.

**Note** The Alpha platform is not supported by Windows Installer.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Template Summary Property](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# AppDataFolder Property

The Windows Installer sets the value of the **AppDataFolder** property to the full path of the Roaming folder for the current user.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ARPAUTHORIZEDCDFPREFIX

## Property

The **ARPAUTHORIZEDCDFPREFIX** property is set to the URL of the update channel for the application.

### Requirements

#### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

### See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ARPCOMMENTS Property

The **ARPCOMMENTS** property provides the Add or Remove Programs Control Panel Comments that are written under the [Uninstall Registry Key](#). This property can be set by the command line or a transform.

## Requirements

Version
Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ARPCONTACT Property

The **ARPCONTACT** property provides the Add or Remove Programs Control Panel Contact that is written under the [Uninstall Registry Key](#). This property can be set by the command line or a transform.

## Requirements

Version
Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ARPHHELPLINK Property

The **ARPHHELPLINK** property is the Internet address for technical support. Product maintenance applets display this value.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ARPHelpTelephone Property

The **ARPHelpTelephone** property is technical support phone numbers. Product maintenance applets display this value.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ARPINSTALLLOCATION Property

The **ARPINSTALLLOCATION** property is the full path to the application's primary folder.

## Remarks

Typically needs to be set by a custom action.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ARPNO MODIFY Property

Setting the **ARPNO MODIFY** property disables Add or Remove Programs functionality in Control Panel that modifies the product. For Windows 2000, this disables the **Modify** button for the product in **Add or Remove Programs** in **Control Panel**. On earlier operating systems, clicking the **Add or Remove Programs** button uninstalls the product rather than entering the maintenance mode wizard.

If the **ARPNO MODIFY** property is set, the [RegisterProduct](#) action writes the value "NoModify" under the registry key:

**HKLM\Software\Microsoft\Windows\CurrentVersion\Uninstall\{product key}**

If the **ARPNO MODIFY** property is set and the **ARPNO REMOVE** property is not set, the RegisterProduct action also writes the UninstallString value under this key. The UninstallString value is a command line for removing the product, rather than reconfiguring the product.

## Remarks

On Windows 2000, this disables the **Change** button for the product in the **Add or Remove Programs** of the **Control Panel**.

This property can be set by a customization transform to prevent users from changing an administrator's customization through **Add or Remove Programs**. This property only affects **Add or Remove Programs**.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[A Customization Transform Example](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ARPNOREMOVE Property

Setting the **ARPNOREMOVE** property disables the **Add or Remove Programs** functionality in **Control Panel** that removes the product. For Windows 2000, this disables the **Remove** button for the product from the **Add or Remove Programs in Control Panel**. For earlier operating systems, this has the effect of removing the product from the list of installed products on the **Add or Remove Programs in Control Panel**.

If the **ARPNOREMOVE** property is set, the [RegisterProduct action](#) writes the value "NoRemove" under the registry key:

**HKLM\Software\Microsoft\Windows\CurrentVersion\Uninstall\{product key}**

Setting the **ARPNOREMOVE** property prevents the UninstallString value from being written under this key. The UninstallString value is a command line for removing the product, rather than reconfiguring the product.

## Remarks

For example, this property can be set during a customization transform to prevent users from removing an administrator customization.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 or later on Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ARNOREPAIR Property

Set the **ARNOREPAIR** property to disable the **Repair** button in the **Programs Wizard**. For more information, see [Getting and Setting Properties](#).

## Requirements

Version
Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ARPPRODUCTICON Property

The **ARPPRODUCTICON** property specifies the foreign key to the [Icon table](#), which is the primary icon for the Windows Installer package.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ARPREADME Property

The **ARPREADME** property provides the Add or Remove Programs Control Panel ReadMe for the application that is written under the [Uninstall Registry Key](#). This property can be set by the command line or a transform.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ARPSIZE Property

The **ARPSIZE** property is the estimated size of the application in kilobytes.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ARPSYSTEMCOMPONENT Property

Setting the **ARPSYSTEMCOMPONENT** property to 1 using the command line or a transform prevents the application from being displayed in the **Add or Remove Programs** list of **Control Panel**.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ARPURLINFOABOUT Property

The **ARPURLINFOABOUT** property is the URL for the link to the publishers home page or the application's home page.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ARPURLUPDATEINFO Property

The **ARPURLUPDATEINFO** property is the URL for the link used to update information on the application.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# AVAILABLEFREEREG Property

The **AVAILABLEFREEREG** property specifies in kilobytes the total free space available in the registry after calling the [AllocateRegistrySpace action](#).

The maximum value of the **AVAILABLEFREEREG** property is 2000000 kilobytes.

This property is set only on Windows 2000.

## Remarks

The **AVAILABLEFREEREG** property must be set to a value large enough to ensure sufficient space in the registry for all registration information added by the installation. The minimum value required to ensure sufficient space depends on where the [AllocateRegistrySpace action](#) is located in the action sequence because the installer automatically increases the space as needed when registering information in the [Registry](#), [Class](#), [SelfReg](#), [Extension](#), [MIME](#), and [Verb](#) tables. The installer does not increase the total registry space to the amount specified by **AVAILABLEFREEREG** until reaching [AllocateRegistrySpace](#) in the action sequence.

If [AllocateRegistrySpace](#) is one of the first actions in the action sequence, **AVAILABLEFREEREG** should be set to the total space required by the registration information in the [Registry](#) table, [Class](#) table, [TypeLib](#) table, [SelfReg](#) table, [Extension](#) table, [MIME](#) table, [Verb](#) table, [custom actions](#) registration, self registration, and any other registry information written during the installation. The value of **AVAILABLEFREEREG** is the total amount of information added by the installation and ensures sufficient space in all cases. This is also the most common case.

If the [AllocateRegistrySpace](#) action can be authored into the action sequence after all [standard actions](#) that write registration data, such as [WriteRegistryValues](#) and [RegisterClassInfo](#), the value of **AVAILABLEFREEREG** needs only be set to the space needed to register custom actions, register type libraries, and any other information not yet registered through the tables.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# BorderSide Property

The installer sets the **BorderSide** property to the width of the window borders in pixels.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# BorderTop Property

The installer sets the **BorderTop** property to the height of the window borders in pixels.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[About Properties](#)  
[Using Properties](#)  
[Property Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# CaptionHeight Property

The installer sets the **CaptionHeight** property to the height, in pixels, of the window caption area.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# CCP\_DRIVE Property

The **CCP\_DRIVE** property is set to the root path of the removable volume that is to be searched by [RMCCPSearch](#). The RMCCPSearch action uses file signatures to validate that qualifying products are installed on a system before performing an upgrade installation.

This property is typically set via the user interface or a custom action. This property should be set before the RMCCPSearch action is run.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ColorBits Property

The installer sets the **ColorBits** property to the number of adjacent color bits for each pixel.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# CommonAppDataFolder Property

The [CommonAppDataFolder](#) property is the full path to the file directory containing application data for all users.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# CommonFiles64Folder Property

The installer sets the **CommonFiles64Folder** property to the full path of the predefined 64-bit Common Files folder. The existing **CommonFilesFolder** property is set to the corresponding 32-bit folder.

## Remarks

The installer sets this property on 64-bit Windows. This property is not used on 32-bit Windows. When using 64-bit Windows, the value may be "C:\Program Files\Common Files".

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# CommonFilesFolder Property

The installer sets the **CommonFilesFolder** property to the full path of the Common Files folder for the current user.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# COMPADDDEFAULT Property

The value of the **COMPADDDEFAULT** property is a list of component GUIDs from the ComponentId column of the [Component table](#), delimited by commas, that are installed in their default configuration. For each component ID in the list, the installer installs the feature that requires the least disk space. The component IDs in the list must be present in the ComponentId column of the Component table. A feature is installed in the same installation state as if the user had requested an installation-on-demand of the feature. The state is determined by which bits are set for the feature in the Attributes column of the [Feature table](#), and which bits are set for the feature's components in the Attributes column of the Component table.

## Remarks

Note that if the SourceOnly bit flag is set in the Attributes column of the [Component](#) table for a component, then the component is installed to run from source.

The installer always evaluates the following properties in the following order.

1. **ADDLOCAL**
2. **REMOVE**
3. **ADDSOURCE**
4. **ADDDEFAULT**
5. **REINSTALL**
6. **ADVERTISE**
7. **COMPADDLOCAL**
8. **COMPADDSOURCE**
9. **COMPADDDEFAULT**
10. **FILEADDLOCAL**
11. **FILEADDSOURCE**

## 12. FILEADDDEFAULT

For example, if the command line specifies: ADDLOCAL=ALL, ADDSOURCE = MyFeature, all the features are first set to run-local and then MyFeature is set to run-from-source. If the command line is: ADDSOURCE=ALL, ADDLOCAL=MyFeature, first MyFeature is set to run-local, and then when ADDSOURCE=ALL is evaluated, all features (including MyFeature) are reset to run-from-source.

The installer sets the **Preselected** property to a value of "1" during the resumption of a suspended installation or when any of the above properties are specified on the command line.

### Requirements

#### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

### See Also

#### Properties

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# COMPADDLOCAL Property

The value of the **COMPADDLOCAL** property is a list of component GUIDs from the ComponentId column of the [Component table](#), delimited by commas, that are to be installed locally. The installer uses this list to determine which features are set to be installed locally, based on the specified components. For each listed component ID, the installer examines all features linked to that component through the [FeatureComponents](#) table, and installs the feature that requires the least amount of disk space to install. The components listed must be present in the Component column of the [Component](#) table.

## Remarks

Note that the component names are case-sensitive. Also note that if the SourceOnly bit flag is set in the Attributes column of the [Component](#) table for a component, then the component is installed to run from source.

The installer always evaluates the following properties in the following order.

1. **ADDLOCAL**
2. **REMOVE**
3. **ADDSOURCE**
4. **ADDDEFAULT**
5. **REINSTALL**
6. **ADVERTISE**
7. **COMPADDLOCAL**
8. **COMPADDSOURCE**
9. **COMPADDDEFAULT**
10. **FILEADDLOCAL**
11. **FILEADDSOURCE**
12. **FILEADDDEFAULT**

For example, if the command line specifies: `ADDLOCAL=ALL`,

ADDSOURCE = MyFeature, all the features are first set to run-local and then MyFeature is set to run-from-source. If the command line is: ADDSOURCE=ALL, ADDLOCAL=MyFeature, first MyFeature is set to run-local, and then when ADDSOURCE=ALL is evaluated, all features (including MyFeature) are reset to run-from-source.

The installer sets the **Preselected** property to a value of "1" during the resumption of a suspended installation or when any of the above properties are specified on the command line.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### Properties

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# COMPADDSOURCE Property

The value of the **COMPADDSOURCE** property is a list of component GUIDs from the ComponentId column of the [Component](#) table, delimited by commas, that are to be installed to run from the source media. The installer uses this value to determine which features are set to be installed to run from source, based on the specified components. For each listed component ID, the installer examines all features linked (through the [FeatureComponents](#) table) to that component, and installs the feature that requires the least amount of disk space to install. The components listed must be present in the Component column of the [Component](#) table.

## Remarks

Note that the component names are case-sensitive. Also note that if the LocalOnly bit flag is set in the Attributes column of the [Component](#) table for a component, then the component is installed to run locally.

The installer always evaluates the following properties in the following order:

1. **ADDLOCAL**
2. **REMOVE**
3. **ADDSOURCE**
4. **ADDDEFAULT**
5. **REINSTALL**
6. **ADVERTISE**
7. **COMPADDLOCAL**
8. **COMPADDSOURCE**
9. **COMPADDDEFAULT**
10. **FILEADDLOCAL**
11. **FILEADDSOURCE**
12. **FILEADDDEFAULT**

For example, if the command line specifies: ADDLOCAL=ALL, ADDSOURCE = MyFeature, all the features are first set to run-local and then MyFeature is set to run-from-source. If the command line is: ADDSOURCE=ALL, ADDLOCAL=MyFeature, first MyFeature is set to run-local, and then when ADDSOURCE=ALL is evaluated, all features (including MyFeature) are reset to run-from-source.

The installer sets the **Preselected** property to a value of "1" during the resumption of a suspended installation or when any of the above properties are specified on the command line.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# COMPANYNAME Property

The **COMPANYNAME** property is the organization or company of the user performing the installation.

## Default Value

A default value can be put in the [Property table](#). If the **COMPANYNAME** property is not set, the installer sets it automatically using values from the registry.

## Remarks

Set the **NOCOMPANYNAME** property to suppress the automatic setting of **COMPANYNAME**.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ComputerName Property

The **ComputerName** property is the computer name of the current system. The installer sets this property by a system call to **GetComputerName**.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# CostingComplete Property

The **CostingComplete** property indicates whether the installer has completed disk space costing. This property can be used to author a dialog box triggered if costing has not been completed. The property is set dynamically during disk space costing and is set to 1 as soon as the costing is complete. This property is initialized to 0 by the [CostFinalize action](#).

## Remarks

For an example of how to author a "Please wait . . ." dialog box that pops up during disk space costing, see the section [Authoring a Conditional "Please wait . . ." Message Box](#).

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### Properties

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Date Property

The **Date** property is the current month, day, and year as a string of literal text in the format MM/DD/YYYY. For example, the date June 22, 2005 can be represented as "06/22/2005". The format of the value depends upon the user's locale, and is the format obtained using **GetDateFormat** with the DATE\_SHORTDATE option. The value of this property is set by the Windows Installer and not the package author.

## Remarks

Because this is a text string, it cannot be used in conditional expressions.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DefaultUIFont Property

The **DefaultUIFont** property sets the default font style used for controls. To specify the default, set **DefaultUIFont** to one of the predefined styles in the [TextStyle table](#) in the [Property table](#).

## Remarks

The **DefaultUIFont** property of every installation package with a UI should be set in the [Property table](#) to one of the predefined styles listed in the [TextStyle table](#). If this property is not specified, the installer uses the System font. This can cause the installer to improperly display text strings when the package's code page is different from the user's default UI code page.

The text and font style of a control can be set as described in [Adding Controls and Text](#). If the character string listed in the Text column of the [Control table](#) or [BBControl table](#) does not specify the font style, the control uses the value of the **DefaultUIFont** property as the font style.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DesktopFolder Property

The installer sets the **DesktopFolder** property to the full path of the current user's Desktop folder. If an "All Users" profile exists and the **ALLUSERS** property is set, then this property is set to the folder in the "All Users" profile.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DISABLEADVTSHORTCUTS

## Property

Setting the **DISABLEADVTSHORTCUTS** property disables the generation of shortcuts supporting [installation-on-demand](#) and [advertisement](#). Setting this property specifies that these shortcuts should instead be replaced by regular shortcuts.

## Default Value

By default, installation-on-demand shortcut generation is enabled.

## Remarks

The shortcuts that support advertisement have the name of a feature, rather than a formatted file path of the form [#filekey], entered in the Target column of the [Shortcut table](#). If this property is set and the feature is installed, the installer generates a regular shortcut to the feature.

This property is commonly used by administrators for users who roam between environments that do and do not support advertising. This property can be set by a transform, in the administrative stream, or in the [Property table](#). If it is set using the command line or by a call to [MsiSetProperty](#), then it must be reset again during each subsequent installation.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DISABLEMEDIA Property

Setting the **DISABLEMEDIA** property prevents the installer from registering any media source, such as a CD-ROM, as a valid source for the product. If browsing is enabled, however, a user may still browse to a media source.

## Remarks

Note that the **DISABLEMEDIA** property has a different effect than setting the DisableMedia policy. Setting **DISABLEMEDIA** prevents the registration of any media source, and this also prevents the first time installation of an application from a media sources.

For details about securing the media sources of *managed application* against browsing and installation by non-administrative users, see [Source Resiliency](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DISABLEROLLBACK Property

Setting the **DISABLEROLLBACK** property disables the [rollback](#) option for the current configuration. Set this property to 1 to prevent the installer from generating a rollback script and saving copies of deleted files during the installation.

## Remarks

By default, rollback is enabled.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[PromptRollbackCost property](#)

[System Policy](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DiskPrompt Property

The **DiskPrompt** property value holds a string displayed by a dialog box prompting for a disk. This string should include the name of the product and a placeholder for the text printed on the CD-ROM surface.

## Remarks

Advertised as a product property.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Authoring Disk Prompt Messages](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# EnableUserControl Property

If the **EnableUserControl** property is set to 1, then the installer can pass all [public properties](#) to the server side during a managed installation using elevated privileges. Setting this property has the same effect as setting the **EnableUserControl** [system policy](#).

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)  
[Restricted Public Properties](#)  
[system policy](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# EXECUTEACTION Property

The **EXECUTEACTION** property is set to determine which top-level action the [ExecuteAction action](#) initiates.

## Remarks

The top-level actions, such as the [INSTALL action](#), [ADVERTISE action](#), and [ADMIN action](#) set the **EXECUTEACTION** property to the name of the top-level action.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# EXECUTEMODE Property

The **EXECUTEMODE** property specifies how the installer executes system updates. This property may be useful when it is necessary to run an installation without actually changing the system. The property can be set to None to disable updating operations such as the installation of files and registry values.

This property can have one of the following two values. The installer only examines the first letter of the value and is case insensitive.

## None

No changes are made to the system. The installer runs the user interface and then queries the database.

## Script

All changes to the system are made through script execution. This is the default mode.

## Default Value

If this property is not defined, the execution mode defaults to Script.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# FASTOEM Property

The **FASTOEM** property is designed to enable OEMs to reduce the time it takes to install Windows Installer applications for a specific scenario. Do not author the **FASTOEM** property into the [Property Table](#).

The **FASTOEM** property is only applicable if all of these conditions are true:

- The application is being installed on the same volume as the folder containing the source files.
- The source files are deleted after the installation.
- No user interface is displayed during the installation. The [user interface level](#) is none.
- The installation is performed in the per-machine [installation context](#).
- There is enough space on the computer for a successful installation.
- This is first time installation. The installation is not advertising, reinstalling, removing, or doing an administrative installation.
- No features are installed to run from source.
- The installation package contains no [isolated components](#). Because isolated components require source files to remain located at the source, the **FASTOEM** property cannot currently be used with isolated components.

If all of the previous conditions are true, setting the **FASTOEM** property enables Windows Installer to improve performance by doing the following:

- Move rather than copy files on the same volume. This does not guarantee that all files are moved rather than copied. Note that if the computer has multiple hard drives, you must initialize the [ROOTDRIVE](#) property on the command line to the same drive containing the installation source.
- Omit this source from the application's source list so that subsequent

reinstallation or maintenance installations default to the CD-ROM sources specified in the [Media Table](#).

- Streamline [file costing](#).
- Suppress progress messages sent from Windows Installer to the client.

## Remarks

Because no progress messages are sent when **FASTOEM** is set, it is recommended that authors manually write a value of 1800 for Timeout into the key

**HKLM\Software\Policies\Microsoft\Windows\Installer\Timeout**

Timeout is a **REG\_DWORD** type.

To display the size of the application in Add or Remove Programs in the Windows 2000 Control Panel, you must manually write the value of EstimatedSize into the key

**HKLM\Software\Microsoft\Windows\CurrentVersion\Uninstall\  
<Product Code>**

This is a **REG\_DWORD** type and equals to the size of the application in Kbytes. The installer does not automatically write this value.

Use the following example command line if the CD-ROM shipped to end users stores the application's installation package at the root of the CD-ROM. Note that the volume label in the [Media Table](#) of the .msi file must match the volume label of the CD-ROM.

**Msiexec /I C:\TempImage\package.msi /qn /le logfile.txt  
ALLUSERS=1 FASTOEM=1 DISABLEROLLBACK=1 ROOTDRIVE=C:\**

Use the following example command line if the installation package is not located at the root of the CD-ROM shipped to end users. You must set the **MEDIAPACKAGEPATH** property in this case to the path to the installation package. The volume label in the [Media Table](#) of the .msi file must match the volume label of the CD-ROM. In this case follow this example.

**Msiexec /I C:\TempImage\package.msi /qn /le logfile.txt  
ALLUSERS=1 FASTOEM=1 DISABLEROLLBACK=1**

MEDIAPACKAGEPATH=C:\TempImage\package.msi  
ROOTDRIVE=C:\

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# FavoritesFolder Property

The installer sets the **FavoritesFolder** property to the full path of the Favorites folder for the current user.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# FILEADDDEFAULT Property

The value of the **FILEADDDEFAULT** property is a list of file keys delimited by commas that are installed in their default configuration. For each file key in the list, the installer determines the component that controls that file and installs the feature that requires the least disk space. The file keys in the list must be present in the File column of the [File](#) table. A feature is installed in the same installation state as if the user had requested an installation-on-demand of the feature. The state is determined by which bits are set for the feature in the Attributes column of the [Feature table](#), and which bits are set for the feature's components in the Attributes column of the [Component table](#).

## Remarks

Note that the file key names are case-sensitive. Also note that if the SourceOnly bit flag is set in the Attributes column of the [Component](#) table for a component, then the component is installed to run from source.

The installer always evaluates the following properties in the following order.

1. **ADDLOCAL**
2. **REMOVE**
3. **ADDSOURCE**
4. **ADDDEFAULT**
5. **REINSTALL**
6. **ADVERTISE**
7. **COMPADDLOCAL**
8. **COMPADDSOURCE**
9. **COMPADDDEFAULT**
10. **FILEADDLOCAL**
11. **FILEADDSOURCE**
12. **FILEADDDEFAULT**

For example, if the command line specifies: ADDLOCAL=ALL, ADDSOURCE = MyFeature, all the features are first set to run-local and then MyFeature is set to run-from-source. If the command line is: ADDSOURCE=ALL, ADDLOCAL=MyFeature, first MyFeature is set to run-local, and then when ADDSOURCE=ALL is evaluated, all features (including MyFeature) are reset to run-from-source.

The installer sets the **Preselected** property to a value of "1" during the resumption of a suspended installation or when any of the above properties are specified on the command line.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# FILEADDLOCAL Property

The value of the **FILEADDLOCAL** property denotes a list of file keys delimited by commas that are to be installed to run from the local source media. For each file key in the list, the installer determines the component that controls that file, then examines all features linked to that component by the [FeatureComponents](#) table and installs the feature that requires the least amount of disk space. The file keys in the list must be present in the File column of the [File](#) table.

## Remarks

Note that the file key names are case-sensitive. Also note that if the LocalOnly bit flag is set in the Attributes column of the [Component](#) table for a component, then the component is installed to run locally.

The installer always evaluates the following properties in the following order.

1. **ADDLOCAL**
2. **REMOVE**
3. **ADDSOURCE**
4. **ADDDEFAULT**
5. **REINSTALL**
6. **ADVERTISE**
7. **COMPADDLOCAL**
8. **COMPADDSOURCE**
9. **COMPADDDEFAULT**
10. **FILEADDLOCAL**
11. **FILEADDSOURCE**
12. **FILEADDDEFAULT**

For example, if the command line specifies: `ADDLOCAL=ALL, ADDSOURCE = MyFeature`, all the features are first set to run-local and then MyFeature is set to run-from-source. If the command line is:

ADDSOURCE=ALL, ADDLOCAL=MyFeature, first MyFeature is set to run-local, and then when ADDSOURCE=ALL is evaluated, all features (including MyFeature) are reset to run-from-source.

The installer sets the **Preselected** property to a value of "1" during the resumption of a suspended installation or when any of the above properties are specified on the command line.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# FILEADDSOURCE Property

The value of the **FILEADDSOURCE** property denotes a list of file keys delimited by commas that are to be installed to run from the source media. For each file key in the list, the installer determines the component that controls that file, then examines all features linked to that component by the [FeatureComponents](#) table and installs the feature that requires the least amount of disk space. The file keys in the list must be present in the File column of the [File](#) table.

## Remarks

Note that the file key names are case-sensitive. Also note that if the LocalOnly bit flag is set in the Attributes column of the [Component](#) table for a component, then the component is installed to run locally.

The installer always evaluates the following properties in the following order.

1. **ADDLOCAL**
2. **REMOVE**
3. **ADDSOURCE**
4. **ADDDEFAULT**
5. **REINSTALL**
6. **ADVERTISE**
7. **COMPADDLOCAL**
8. **COMPADDSOURCE**
9. **COMPADDDEFAULT**
10. **FILEADDLOCAL**
11. **FILEADDSOURCE**
12. **FILEADDDEFAULT**

For example, if the command line specifies: **ADDLOCAL=ALL**, **ADDSOURCE = MyFeature**, all the features are first set to run-local and then MyFeature is set to run-from-source. If the command line is:

ADDSOURCE=ALL, ADDLOCAL=MyFeature, first MyFeature is set to run-local, and then when ADDSOURCE=ALL is evaluated, all features (including MyFeature) are reset to run-from-source.

The installer sets the **Preselected** property to a value of "1" during the resumption of a suspended installation or when any of the above properties are specified on the command line.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# FontsFolder Property

The installer sets the **FontsFolder** property to the full path of the system Fonts folder.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installed Property

The **Installed** property is set only if the product is installed per-machine or for the current user. This property is not set if the product is installed for a different user.

The **Installed** property can be used in conditional expressions to determine whether a product is installed per-computer or for the current user. For example, the property can be used in the conditional column of a sequence table or to differentiate between a first installation and a maintenance installation.

To determine whether the product is installed for a different user, check the **ProductState** property.

The value of the **ProductVersion** property is the version of the product in string format.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# INSTALLLEVEL Property

The **INSTALLLEVEL** property is the initial level at which features are selected "ON" for installation by default. A feature is installed only if the value in the Level field of the [Feature table](#) is less than or equal to the current **INSTALLLEVEL** value. The installation level for any installation is specified by the **INSTALLLEVEL** property, and can be an integral from 1 to 32,767. For further discussion of installation levels, see [Feature Table](#).

## Default Value

If no value is specified, the install level defaults to 1.

## Remarks

The installation level specified by the **INSTALLLEVEL** property can be overridden by the following properties:

- **ADDLOCAL**
- **ADDSOURCE**
- **ADDDEFAULT**
- **COMPADDLOCAL**
- **COMPADDSOURCE**
- **FILEADDLOCAL**
- **FILEADDSOURCE**
- **REMOVE**
- **REINSTALL**
- **ADVERTISE**

For example, setting **ADDLOCAL=ALL** installs all features locally regardless of the value of the **INSTALLLEVEL** property. If the value of the Level column in the [Feature table](#) is 0, that feature is not installed and not displayed in the UI.

An administrator can permanently disable a feature by applying a

customization transform that sets a 0 in the Level column for that feature. The application of the customization transform prevents the installation and display of the feature even if the user selects a complete installation using the UI or by setting ADDLOCAL to ALL on the command line. See [A Customization Transform Example](#).

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Intel Property

The **Intel** property is set by the Windows Installer to the numeric processor level when running on an Intel processor. The values are the same as the *wProcessorLevel* field of the **SYSTEM\_INFO** structure. When running on a x64 processor, the Windows Installer sets the **Intel** property to reflect the x86 processor emulated by the x64 computer as reported by the operating system.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Intel64 Property

The **Intel64** property is defined only when it is running on an Itanium processor. The value is set by the Windows Installer to the numeric processor level. This property is not set when running on a 64-bit extended processor. For more information, see [Intel](#) Property.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IsAdminPackage Property

The installer sets the **IsAdminPackage** property to 1 if the current installation is running from a package created through an [administrative installation](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# LeftUnit Property

If the **LeftUnit** property is set, the unit is placed to the left of the number instead of the typical right side. This accommodates languages where this is the accepted format. The UI uses this property to display the size of files and the available space on different volumes.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# LIMITUI Property

If the **LIMITUI** property is set, the user interface (UI) level used when installing the package is restricted to Basic. This property is required in packages that have no authored UI but still contain UI tables such as the [Dialog table](#). For a description of UI levels, see [MsiSetInternalUI](#)

## Remarks

Installation packages containing the **LIMITUI** property must also contain the [ARPNOMODIFY](#) property. This is required for a user to obtain the correct behavior from the **Add or Remove Programs** in the **Control Panel** utility when attempting to configure a product.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[MsiSetInternalUI](#)

[ARPNOMODIFY](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# LocalAppDataFolder Property

The **LocalAppDataFolder** property is the full path to the file system directory that serves as the data repository for local (non-roaming) applications.

## Requirements

Version
Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# LOGACTION Property

The **LOGACTION** property is a list of action names separated only by semicolons and with no spaces. The action names in this property are case-insensitive and the list can end with a semicolon.

## Remarks

The installer logs Action Data messages for the actions that are in the list. The property has no effect on other types of logging.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# LogonUser Property

The **LogonUser** property is the user name for the currently logged on user. Set by the installer by a system call to **GetUserName**.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Manufacturer Property

The **Manufacturer** property is the name of the manufacturer for the product. It is advertised as a product property.

## Remarks

This property is REQUIRED.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MEDIAPACKAGEPATH Property

If the installation package shipping with an application is not at the root of the CD-ROM that customers receive, the **MEDIAPACKAGEPATH** property must be set on the command line to the relative path of the application on the CD-ROM.

For example, if the path to the package on the media is E:\MyPath\My.msi, then use `MEDIAPACKAGEPATH="\MyPath\"`.

Administrators may create CD-ROMs from an administrative installation point. If the location of the root of the installation is changed on the new CD-ROMs, the **MEDIAPACKAGEPATH** property must be set to the new path to install from this media. A source with a location on the CD-ROM different than what is specified in the package is unusable. It is not necessary, however, to use this property when creating an administrative installation point from shipping media.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MediaSourceDir Property

The installer sets the **MediaSourceDir** property to 1 when the installation uses a source located on media, such as a CD-ROM. This property is not set when the installation uses a source located at a network location. For example, the [SelectionTree Control](#) uses **MediaSourceDir** to determine whether the installation is being run from source or run from a network location.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MSIARPSETTINGSIDENTIFIER

## Property

The **MSIARPSETTINGSIDENTIFIER** property contains a semi-colon delimited list of the registry locations where the application stores a user's settings and preferences. During an operating system upgrade, setup can use this information to improve the experience of users who are migrating applications.

The value of the **MSIARPSETTINGSIDENTIFIER** property is literal text that contains the list of registry locations where the application stores its settings. The value can specify multiple possible registry locations. Separate multiple registry locations by using semi-colons. Application settings are typically stored in the **HKCU** or **HKLM** registry hives in the form: *Company\Application\Version*. The following example is a possible value for the **MSIARPSETTINGSIDENTIFIER** property.

*MyCompany\AppSuite\1.0;MyCompany\App1\1.0;MyCompany\App2\1.0*

This property is optional and can be set in the [Property](#) table. If this property appears in the Property table, the value must not be set to NULL.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 4.5 on Windows Server 2003 or Windows XP. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MSICHECKCRCS Property

Cyclic redundancy checking (CRC) is available with Windows Installer version 2.0 and later. The installer does a CRC on files only if the **MSICHECKCRCS** property is set. For more information, see [CRC Checking During an Installation](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MSIDEPLOYMENTCOMPLIANT

## Property

The **MSIDEPLOYMENTCOMPLIANT** property can be set to indicate to the installer that the package has been authored and tested to comply with *User Account Control* (UAC) in Windows Vista. If this property is not set, the installer will determine whether the package complies with UAC on Windows Vista.

For more information about UAC and Windows Installer, see [Using Windows Installer with UAC](#) and [Guidelines for Packages](#).

**Windows Installer 3.1, Windows Installer 3.0 and Windows Installer 2.0:** This property is ignored. This property is only used by Windows Installer 4.0 in Windows Vista.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. See the [Windows Installer Run-Time Requirements](#) for information about the minimum service pack required by a Windows Installer version.

## See Also

### Properties

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MSIDISABLEEEEUI Property

To disable the embedded user interface for the installation defined in the [MsiEmbeddedUI table](#), set the MSIDISABLEEEEUI property to 1 on the command line.

**Windows Installer 4.0 and earlier:** Not supported. Versions earlier than Windows Installer 4.5 ignore the MSIDISABLEEEEUI Property.

## Remarks

In a multiple-package installation, a child package should typically use the user interface of the parent package and not initialize its own embedded UI. If the MSIDISABLEEEEUI property is not set inside the parent package, the child package uses the embedded UI of the parent package by default and ignores the MSIDISABLEEEEUI property within the child package.

The MSIDISABLEEEEUI property disables the embedded user interface for a single package. You can use the [MsiDisableEmbeddedUI](#) policy to disable embedded user interfaces for all packages on the computer.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.5 on Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MSIDISABLELUAPATCHING

## Property

Set the **MSIDISABLELUAPATCHING** property to 1 on the command line or in the [Property](#) table to prevent least-privilege patching of an application. To prevent least-privilege patching of all applications on the computer, set the [DisableLUAPatching](#) policy to 1. For information about least-privilege user account patching, see [User Account Control \(UAC\) Patching](#).

### Remarks

**Windows Installer 2.0:** Not supported. This function is available beginning with Windows Installer version 3.0.

### Requirements

#### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

### See Also

[Properties](#)

[User Account Control \(UAC\) Patching](#)

[DisableLUAPatching](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MSIDISABLERMRESTART Property

The **MSIDISABLERMRESTART** property determines how applications or services that are currently using files affected by an update should be shut down and restarted to enable the installation of the update.

## Value

Value	Meaning
0	All system services that were shut down to install the update are restarted. All applications that registered themselves with the Restart Manager are restarted.
1	Processes that were shut down using the Restart Manager while installing the update will not be restarted after the update is applied.

## Remarks

The **MSIDISABLERMRESTART** Property is ignored if the Restart Manager is unavailable or disabled.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum service pack required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MSIENFORCEUPGRADECOMPONENTEN Property

Set the **MSIENFORCEUPGRADECOMPONENTRULES** property to 1 on the command line or in the [Property table](#) to apply the upgrade component rules during [small updates](#) and [minor upgrades](#) of a particular product. To apply the rules during small updates and minor upgrades of all products on the computer, set the [EnforceUpgradeComponentRules](#) policy to 1.

**Windows Installer 2.0:** This policy is not supported. This property is available beginning with Windows Installer 3.0.

When the property or policy is set to 1, small updates and minor upgrades can fail because the update tries to do the following that violates the upgrade component rules:

- Add a new feature to the top or middle of an existing feature tree. The new feature must be added as a new leaf feature to an existing feature tree.

In this case, the [ProductCode](#) of the product can be changed and the update can be treated as a [major upgrade](#).

- Remove a component from a feature. This can also occur if you change the GUID of a component. The component identified by the original GUID appears to be removed and the component as identified by the new GUID appears as a new component.

**Windows Installer 4.5 and later:** The component can be removed correctly using Windows Installer 4.5 and later by setting the `msidbComponentAttributesUninstallOnSupersedence` attribute in the [Component Table](#) or by setting the

**MSIUNINSTALLSUPERSEDEDCOMPONENTS** property.

Alternatively, the **ProductCode** of the product can be changed and the update can be treated as a major upgrade.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MSIFASTINSTALL Property

The **MSIFASTINSTALL** property can be used to reduce the time required to install a large Windows Installer package. The property can be set on the command line or in the [Property](#) table to configure operations that the user or developer determines are non-essential for the installation.

The value of the **MSIFASTINSTALL** property can be a combination of the following values.

Value	Meaning
0	Default value
1	No system restore point is saved for this installation.
2	Perform only <a href="#">File Costing</a> and skip checking other costs.
4	Reduce the frequency of progress messages.

**Windows Installer 4.5 or earlier:** Not supported. This property is available beginning with Windows Installer 5.0.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiHiddenProperties Property

The **MsiHiddenProperties** property may be used to prevent the installer from displaying passwords or other confidential information in the log file. To prevent a property from being written into the log file, set the value of the **MsiHiddenProperties** property to the name of the property you wish to hide. You can hide multiple properties by specifying a string of property names delimited by semicolons (;).

**Note** When the [Debug](#) policy is set to a value of 7, the installer will write information entered on a command line into the log. This makes public properties entered on a command line visible even if the property is included in the **MsiHiddenProperties** property. This can make confidential information entered on a command line visible in the log. For more information, see [Preventing Confidential Information from Being Written into the Log File](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MSIINSTALLPERUSER Property

The **MSIINSTALLPERUSER** and the **ALLUSERS** properties can be set by the user at installation time, through the user interface or on a command line, to request that the Windows Installer install a dual-purpose package for the current user or all users of the computer. To use the **MSIINSTALLPERUSER** property, the value of the **ALLUSERS** property must be 2 and the package has to have been authored to be capable of installation into either the per-user or a per-machine context. For information about authoring a dual-purpose package, see [Single Package Authoring](#). If the value of the **ALLUSERS** property does not equal 2, the value of the **MSIINSTALLPERUSER** property is ignored and has no effect on the installation. The value of **MSIINSTALLPERUSER** property is ignored when installing the package using Windows Installer 4.5 or earlier.

To request that the Windows Installer install the dual-purpose package in the per-machine [installation context](#), the user can set the value of the **MSIINSTALLPERUSER** property to an empty string ("") and the value of the **ALLUSERS** property to 2 using an authored user interface or a command line.

To request that the Windows Installer install the dual-purpose package in the per-user installation context, the user can set the value of the **MSIINSTALLPERUSER** property to 1 and the value of the **ALLUSERS** property to 2 using an authored user interface or a command line.

If the value of the **ALLUSERS** property does not equal 2, the Windows Installer ignores the value of the **MSIINSTALLPERUSER** property. If Windows Installer installs the application in the per-machine context, it resets the value of the **ALLUSERS** property to 1. If Windows Installer installs the application in the per-user context, it resets the value of the **ALLUSERS** property to an empty string (""). Applications that have been installed per-user therefore receive all updates or repairs on a per-user basis and applications installed per-machine receive updates or repairs on a per-machine basis.

**Windows Installer 4.5 or earlier:** The **MSIINSTALLPERUSER** property is ignored by versions earlier than Windows Installer 5.0.

## Default Value

The recommended default installation context is per-user for a dual-purpose package. Author `MSIINSTALLPERUSER=1` and `ALLUSERS=2` in the [Property table](#) of the dual-purpose package to specify per-user as the default installation context.

## Remarks

You can ensure the **MSIINSTALLPERUSER** property has not been set by setting its value to an empty string (""), `MSIINSTALLPERUSER=""`.

The installation context determines the values of the [DesktopFolder](#), [ProgramMenuFolder](#), [StartMenuFolder](#), [StartupFolder](#), [TemplateFolder](#), [AdminToolsFolder](#), [ProgramFilesFolder](#), [CommonFilesFolder](#), [ProgramFiles64Folder](#), and [CommonFiles64Folder](#) properties. The installation context determines the parts of the registry where entries in the [Registry table](#) and [RemoveRegistry table](#), with -1 in the Root column, are written or removed. For information about the installation context, see [Installation Context](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)  
[ALLUSERS](#)  
[Installation Context](#)  
[Single Package Authoring](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MSIINSTANCEGUID Property

The presence of the **MSIINSTANCEGUID** property indicates that a product code–changing transform is registered to the product. The value of the **MSIINSTANCEGUID** property specifies which instance of a product is to be used for installation. The **MSIINSTANCEGUID** property can only reference a product that has already been installed with an instance transform.

Instance transforms are product code–changing transforms available with the installer running on either Windows Server 2003 or Windows XP. For more information, see [Installing Multiple Instances of Products and Patches](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiLogFileLocation Property

The value of the **MsiLogFileLocation** property is set to the full path of the log file, when logging is enabled. The **MsiLogFileLocation** property can be used to display the log file, or location of the log file, in a user interface at the end of the installation.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 4.5 on Windows Server 2003 or Windows XP. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiLogging Property

The **MsiLogging** property sets the default logging mode for the Windows Installer package. If this optional property is present in the [Property table](#), the installer generates a log file named MSI\*.LOG. The full path to the log file is given by the value of the [MsiLogFileLocation](#) property.

## Value

The value of this property should be a string of the following characters that specify the default logging mode.

Value	Meaning
I	Status messages.
w	Nonfatal warnings.
e	All error messages.
a	Start up of actions.
r	Action-specific records.
u	User requests.
c	Initial UI parameters.
m	Out-of-memory or fatal exit information.
o	Out-of-disk-space messages.
p	Terminal properties.
v	Verbose output.
x	Extra debugging information. Only available on Windows Server 2003.
!	Flush each line to the log.

## Remarks

This property is available starting with Windows Installer 4.0.

You cannot use the "+" and "\*" values of the /L option in the value of the **MsiLogging** property.

The logging mode can be set using policy, a command-line option, or programmatically. This overrides the default logging mode. For more information about all the methods that are available for setting the logging mode, see [Normal Logging](#) in the [Windows Installer Logging](#) section.

If the **MsiLogging** property is present in the Property table, the default logging mode of the package can be modified by changing the value of this property using a [database transform](#). The default logging mode cannot be changed using a [Patch Package](#) (a .msp file.)

If the **MsiLogging** property has been set in the Property table, the default logging mode for all users of the computer can be specified by setting both the [DisableLoggingFromPackage](#) policy and [Logging](#) policy. Setting both the [DisableLoggingFromPackage](#) policy and [Logging](#) policy will override the **MsiLogging** property for all packages.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 4.5 on Windows Server 2003 or Windows XP. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiNetAssemblySupport Property

The **MsiNetAssemblySupport** property indicates whether the computer supports common language run-time assemblies. On systems that support common language run-time assemblies, the installer sets the value of **MsiNetAssemblySupport** to the file version of Fusion.dll. The installer does not set this property if the operating system does not support common language run-time assemblies. When multiple versions of Fusion.dll are installed side-by-side on the user's computer, this property is set to the latest version of the Fusion.dll file. For example, if .NET Framework version 1.0.3705 (Fusion.dll version 1.0.3705.15) and .NET Framework version 1.1.4322 (Fusion.dll version 1.1.4322.314) are installed side-by-side, the **MsiNetAssemblySupport** property is set to 1.1.4322.314. For more information, see [Assemblies](#).

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MSINEWINSTANCE Property

The **MSINEWINSTANCE** property indicates the installation of a new instance of a product with instance transforms. This property supports multiple instances through product code–changing transforms. The value of this property is 1. The presence of this property on the command line requires that the **TRANSFORMS** property also be present. The first transform listed in **TRANSFORMS** must be an instance transform. For more information see, [Installing Multiple Instances of Products and Patches](#)

This property is available with the installer running a system in the Windows Server 2003 or Windows XP.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MSINODISABLEMEDIA Property

Set the **MSINODISABLEMEDIA** property to prevent the installer from setting the **DISABLEMEDIA** property. Setting the **DISABLEMEDIA** property prevents the installer from registering any media source, such as a CD-ROM, as a valid source for the product.

When **MSINODISABLEMEDIA** is not set, the installer might add **DISABLEMEDIA** to the administrative property stream when performing an [administrative installation](#). This prevents users who install from the administrative image from ever installing from media, such as a CD-ROM.

- When performing an administrative installation, the installer only adds **DISABLEMEDIA** to the administrative property stream if the [Page Count Summary](#) Property is less than 150.

If **DISABLEMEDIA** is listed in the [AdminProperties](#) property, setting **MSINODISABLEMEDIA** prevents **DISABLEMEDIA** from being set during an administrative installation. In this case, the installer may register a media source and a user could then have the option to reinstall from the media source if the original administrative installation image became unavailable later.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

## Properties

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiNTProductType Property

The installer sets the **MsiNTProductType** property for Windows NT, Windows 2000, and later operating systems. This property indicates the Windows product type.

For Windows 2000 and later operating systems, the installer sets the following values. Note that values are the same as of the **wProductType** field of the **OSVERSIONINFOEX** structure.

Value	Meaning
1	Windows 2000 Professional and later
2	Windows 2000 domain controller and later
3	Windows 2000 Server and later

For operating systems earlier than Windows 2000, the installer sets the following values.

Value	Meaning
1	Windows NT Workstation
2	Windows NT domain controller
3	Windows NT Server

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiINTSuiteBackOffice Property

On Windows 2000 and later operating systems, the installer sets the **MsiINTSuiteBackOffice** property to 1 if Microsoft BackOffice components are installed. The installer sets this property to 1 only if the VER\_SUITE\_BACKOFFICE flag is set in the **OSVERSIONINFOEX** structure. Otherwise, the installer does not set this property.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiINTSuiteDataCenter Property

On Windows 2000 and later operating systems, the installer sets the **MsiINTSuiteDataCenter** property to 1 if Windows 2000 Datacenter Server is installed. The installer sets this property to 1 only if the VER\_SUITE\_DATACENTER flag is set in the **OSVERSIONINFOEX** structure. Otherwise, the installer does not set this property.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiNTSuiteEnterprise Property

On Windows 2000 and later operating systems, the installer sets the **MsiNTSuiteEnterprise** property to 1 if Windows 2000 Advanced Server is installed. The installer sets this property to 1 only if the VER\_SUITE\_ENTERPRISE flag is set in the **OSVERSIONINFOEX** structure. Otherwise, the installer does not set this property.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiINTSuitePersonal Property

On Windows XP and later operating systems, the installer sets the **MsiINTSuitePersonal** property to 1 if the operating system is Windows XP Home Edition. The installer sets this property to 1 only if the VER\_SUITE\_PERSONAL flag is set in the **OSVERSIONINFOEX** structure. Otherwise the installer does not set this property.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiINTSuiteSmallBusiness Property

On Windows 2000 and later operating systems, the installer sets the **MsiINTSuiteSmallBusiness** property to 1 if Microsoft Small Business Server is installed. The installer sets this property to 1 only if the VER\_SUITE\_SMALLBUSINESS flag is set in the **OSVERSIONINFOEX** structure. Otherwise the installer does not set this property.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiINTSuiteSmallBusinessRestricted Property

On Windows 2000 and later operating systems, the installer sets the **MsiINTSuiteSmallBusinessRestricted** property to 1 if Microsoft Small Business Server is installed with the restrictive client license in force. The installer sets this property to 1 only if the VER\_SUITE\_SMALLBUSINESS\_RESTRICTED flag is set in the OSVERSIONINFOEX structure. Otherwise the installer does not set this property.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiINTSuiteWebServer Property

On Windows 2000 and later operating systems, the installer sets the **MsiINTSuiteWebServer** property to 1 if the installer is running on a web edition of the Windows Server 2003. The installer sets this property to 1 only if the VER\_SUITE\_BLADE flag is set in the **OSVERSIONINFOEX** structure. Otherwise the installer does not set this property.

## Remarks

This property is available only with the Windows Server 2003 release of the installer.

## Requirements

Version
Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiPatchRemovalList Property

The installer sets the value of the **MsiPatchRemovalList** property to a list of patches that are being removed during the installation. The patches are represented in the list by their patch code GUIDs separated by semicolons.

Developers can use the **MsiPatchRemovalList** property to author a Windows Installer package or patch that performs [custom actions](#) on the removal of a patch. The custom action can be authored into the original installation package, a patch that has already been applied to the package, or a patch that is not an [uninstallable patch](#). The custom action can be conditionalized on the **MsiPatchRemovalList** property in the sequence tables. See [Using Properties in Conditional Statements](#) for more information about conditionalizing actions.

The custom action can obtain the GUIDs of patches being removed from the value of the **MsiPatchRemovalList** property. The custom action can determine whether the installation state of the patch is applied, obsolete, or superseded by calling the [MsiGetPatchInfoEx](#) function or the [PatchProperty](#) property of the [Patch object](#).

## Remarks

**Windows Installer 2.0:** Not supported. The **MsiPatchRemovalList** property is available beginning with Windows Installer 3.0.

For more information about removing patches, see [Removing Patches](#).

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows

**Version**

Installer version.

**See Also**[Properties](#)[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MSIPATCHREMOVE Property

The **MSIPATCHREMOVE** property specifies the list of patches to remove during an installation. Individual patches in the list are separated by semicolons and can be represented by patch code GUID or the full path to the patch. The [MsiRemovePatches](#) function and the [RemovePatches](#) method of the [Installer](#) object set the **MSIPATCHREMOVE** property.

The **MSIPATCHREMOVE** property can be set on the command line as follows to remove a patch.

```
msiexec /i A:\Example.msi  
MSIPATCHREMOVE=c:\patches\qfe1.msp;{0BBB87F1-3186-409C-  
8CDD-C88AA2A4A7E0};{A86B443B-E3BF-4009-ADED-  
F716FC735858}/qb
```

## Remarks

**Windows Installer 2.0:** Not supported. The **MSIPATCHREMOVE** property is available beginning with Windows Installer 3.0.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MSIRESTARTMANAGERCONTROL

## Property

The **MSIRESTARTMANAGERCONTROL** Property specifies whether the Windows Installer package uses the Restart Manager or [FilesInUse Dialog](#) functionality.

### Value

Value	Meaning
0	This is the default value if the property is not set. Windows Installer always attempts to use the Restart Manager on Windows Vista.
"Disable"	Disables interaction of the package with the Restart Manager. Windows Installer uses the FilesInUse Dialog.
"DisableShutdown"	Windows Installer uses the FilesInUse Dialog. This setting disables attempts by the Restart Manager to mitigate restarts when installing a Windows Installer package that has not been authored to use the Restart Manager. The installer still uses the Restart Manager to detect files in use by applications.

### Remarks

The **MSIRESTARTMANAGERCONTROL** Property is ignored if the Restart Manager is unavailable or disabled.

The value of this property can be modified using customization transforms or upgrades. Changing the value of this property from custom actions has no effect.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum service pack required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[DisableAutomaticApplicationShutdown](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiRestartManagerSessionKey Property

The installer sets the value of the **MsiRestartManagerSessionKey** property to the session key for the Restart Manager session. Custom actions can use the session key to join the Restart Manager session.

## Remarks

The installer sets the value of the **MsiRestartManagerSessionKey** property at initialization, and then clears the value during the [InstallValidate](#) action. Custom actions that need the value of the **MsiRestartManagerSessionKey** property must be come before the InstallValidate action in the action sequence.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum service pack required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MSIRMSHUTDOWN Property

The **MSIRMSHUTDOWN** property determines how applications or services that are currently using files affected by an update should be shut down to enable the installation of the update.

## Value

Value	Meaning
0	Processes or services that are currently using files affected by the update are shut down.
1	Processes or services that are currently using files affected by the update, and that do not respond to the Restart Manager, are forced to shut down.
2	Processes or services that are currently using files affected by the update are shut down only if they have all been registered for a restart. If any process or service has not been registered for a restart, then no processes or services are shut down.

## Remarks

The **MSIRMSHUTDOWN** property is ignored if the Restart Manager is unavailable or disabled.

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. See the [Windows Installer Run-Time](#)

**Version**

[Requirements](#) for information about the minimum service pack required by a Windows Installer version.

**See Also**

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiRunningElevated Property

The installer sets the value of the **MsiRunningElevated** property to 1 when the installer is running with *elevated* privileges.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. See the [Windows Installer Run-Time Requirements](#) for information about the minimum service pack required by a Windows Installer version.

## See Also

[Properties](#)

[Using Windows Installer with UAC](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSystemRebootPending Property

The installer sets the value of the **MsiSystemRebootPending** property to 1 if a restart of the operating system is currently pending.

Package authors can base a condition in the [LaunchCondition table](#) on this property to prevent the installation of their package if a system restart is pending.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 4.5 on Windows Server 2003 or Windows XP. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties System Reboots](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiTabletPC Property

The installer sets the **MsiTabletPC** property to a nonzero value if the current operating system is Windows XP Tablet PC Edition. The installer uses the **GetSystemMetrics** function with SM\_TABLETPC, and the property receives the nonzero value returned by this function. If the current system is not Windows XP Tablet PC Edition, the **GetSystemMetrics** function returns zero and the installer does not set this property.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 4.5 on Windows Server 2003 or Windows XP. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiUIHideCancel Property

The installer sets the **MsiUIHideCancel** property to 1 when the internal user interface level has been set to include `INSTALLUILEVEL_HIDECANCEL` with the [MsiSetInternalUI](#) function or the [UILevel](#) property of the [Installer](#) object or by using [Command Line Options](#).

## Remarks

**Windows Installer 2.0:** Not supported. The **MsiUIHideCancel** property is available beginning with Windows Installer 3.0.

## Requirements

Version
Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiUIProgressOnly Property

The Installer sets the **MsiUIProgressOnly** property to 1 when the internal install level has been set to include `INSTALLUILEVEL_PROGRESSONLY` with the [MsiSetInternalUI](#) function or the [UILevel property](#) of the [Installer](#) object.

## Remarks

**Windows Installer 2.0:** Not supported. The **MsiUIProgressOnly** property is available beginning with Windows Installer 3.0.

## Requirements

Version
Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiUISourceResOnly Property

The Installer sets the **MsiUISourceResOnly** property to 1 when the internal install level has been set to include `INSTALLUILEVEL_SOURCERESONLY` with the [MsiSetInternalUI](#) function or the [UILevel](#) property of the [Installer](#) object.

## Remarks

**Windows Installer 2.0:** Not supported. The **MsiUISourceResOnly** property is available beginning with Windows Installer 3.0.

## Requirements

Version
Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MSIUNINSTALLSUPERSEDEDCOMPONENTS Property

Set the **MSIUNINSTALLSUPERSEDEDCOMPONENTS** property to 1 in the [Property table](#) or on a command line. When this property has been set to 1, the installer determines whether the components in any superseded patches are becoming redundant. The installer can unregister and uninstall redundant components to prevent leaving behind orphan components on the computer.

Setting this property affects the components in all patches being superseded. To enable this functionality for single components, use the `msidbComponentAttributesUninstallOnSupersedence` attribute in the [Component table](#).

**Windows Installer 4.0 and earlier:** Not supported. Versions earlier than Windows Installer 4.5 ignore the **MSIUNINSTALLSUPERSEDEDCOMPONENTS** Property.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.5 on Windows Vista, Windows XP, Windows Server 2003, and Windows Server 2008. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MSIUSEREALADMINDETECTION

## Property

Set the **MSIUSEREALADMINDETECTION** property to 1 to request that the installer use actual user information when setting the **AdminUser** property. When running on Windows Vista, the **Privileged** and **AdminUser** are the same. Authors should use the **Privileged** property in new packages. Legacy packages that require distinct **Privileged** and **AdminUser** properties can restore the difference by setting the **MSIUSEREALADMINDETECTION** property.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiWin32AssemblySupport Property

The **MsiWin32AssemblySupport** property indicates whether the computer supports Win32 assemblies. On systems that support Win32 assemblies, the installer sets the value of **MsiWin32AssemblySupport** to the file version of sxs.dll. The installer does not set this property if the operating system does not support Win32 assemblies. For more information, see [Assemblies](#).

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Msix64 Property

The **Msix64** property is defined only if running on an x64 processor. The value is set by the installer to the numeric processor level. The values are the same as the **wProcessorLevel** field of the **SYSTEM\_INFO** structure.

## Remarks

**Windows Installer 2.0 and Windows Installer 3.0:** Not supported. The **Msix64** property is available beginning with Windows Installer 3.1.

## Requirements

Version
Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.1 or later on Windows Server 2003, Windows XP, and Windows 2000

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MyPicturesFolder Property

The **MyPicturesFolder** property is the full path to the MyPictures folder.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# NetHoodFolder Property

The **NetHoodFolder** property is the full path to the NetHood folder.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# NOCOMPANYNAME Property

Set the **NOCOMPANYNAME** property to 1 to suppress the automatic setting of the **COMPANYNAME** property by the installer. This property is used by applications that need to collect the company name at the first-run.

## Default Value

Not set. The installer sets the **COMPANYNAME** property automatically using values from the registry.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# NOUSERNAME Property

Set the **NOUSERNAME** property to 1 to suppress the installer from setting of the **USERNAME** property. This property is used by applications that need to collect the user name at the first-run.

## Default Value

Not set. The installer sets the **USERNAME** property automatically using values from the registry.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# OLEAdvtSupport Property

The installer sets the **OLEAdvtSupport** property to true when the current user's system has been upgraded to work with installation-on-demand through COM. Note that for the installer to register a COM class and enable installation-on-demand, the class must be listed in the [Class](#) and [ProgId](#) tables.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[ShellAdvtSupport Property](#)

[Platform Support of Advertisement](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# OriginalDatabase Property

The Windows Installer sets the **OriginalDatabase** property to the path of the installation database used to launch the installation. If the installation is launched from a command line, the value depends on whether the recache package option (the -v flag) is present in the **REINSTALLMODE** property.

<b>Installation Method</b>	<b>OriginalDatabase Value</b>
Any installation launched by invoking the path of the installation package (.msi file).	Path to the installation package (.msi file).
Installation launched from a command line. The installation is not launched from a package path. The recache option (-v flag) is present in the <b>REINSTALLMODE</b> property.	Path to the database on the source.
Installation launched from a command line. The installation is not launched from a package path. The recache option (-v flag) is not present in the <b>REINSTALLMODE</b> property.	Path to the cached database.

## Remarks

During a first time installation, a custom action sequenced before the **ResolveSource** action can use the **OriginalDatabase** property to determine the location of the installation source.

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows

## Version

Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# OutOfDiskSpace Property

The installer sets the **OutOfDiskSpace** property to TRUE if any volume that is a target of the current installation has insufficient disk space to accommodate the installation. If all volumes have sufficient space, the value is FALSE. Selection resolution actions use this value to cancel an installation and generate a dialog box.

This property is valid at any time after the [CostFinalize](#) action is executed. The **OutOfNoRbDiskSpace** property status is dynamically updated any time the total installation cost is recalculated (for example, any time the installation state of any feature is changed through the [Selection](#) dialog).

## Remarks

Any dialog relying on the **OutOfDiskSpace** property to determine whether to bring up a dialog must set the [TrackDiskSpace Dialog Style Bit](#) for the dialog to dynamically update space on the target volumes.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# OutOfNoRbDiskSpace Property

The installer sets the **OutOfNoRbDiskSpace** property to True if any volume that is a target of the installation has insufficient disk space to accommodate the installation. In this case, the **OutOfNoRbDiskSpace** property is set to True even if rollback has been disabled. If all volumes have sufficient space, the value is False.

A developer of an installation package can handle the situation when the **OutOfDiskSpace** property is True and the **OutOfNoRbDiskSpace** property is False by authoring a user interface that presents the user with an option to disable rollback and continue the installation. For information about conditionally displaying a dialog box, see [ControlEvent Overview](#). For information about disabling rollback, see [EnableRollback ControlEvent](#).

The **OutOfNoRbDiskSpace** property is valid at any time after the **CostFinalize action** has been executed. The **OutOfNoRbDiskSpace** property status is dynamically updated any time the total installation cost is recalculated (for example, any time the installation state of any feature is changed through the [Selection dialog](#)). Selection resolution actions use this value to cancel an installation and generate a dialog box.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties ControlEvent Overview](#)

## **OutOfDiskSpace property**

EnableRollback ControlEvent

CostFinalize action

Selection dialog

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ParentOriginalDatabase Property

During a concurrent installation, the installer sets the **ParentOriginalDatabase** property in the concurrent installation's session to the same value as the **OriginalDatabase** property in the parent installation's session. Parent installations use the concurrent installation actions to perform a concurrent installation. An installation package can determine whether it is being installed by a concurrent installation action by checking the value of this property.

**Note** Concurrent installations are not recommended for the installation of applications intended for release to the public. For information about concurrent installations please see [Concurrent Installations](#).

**Note** This property is not set if the concurrent installation is being run by the [RemoveExistingProducts](#) action.

## Remarks

To prevent a package from ever being installed as a concurrent installation, add either of the following conditional statements to the [LaunchCondition](#) table. This prevents the package from ever being installed by a concurrent installation action run by another installation. This does not prevent the package from being installed by the [RemoveExistingProducts](#) action.

```
"Not ParentProductCode"
```

```
"Not ParentOriginalDatabase"
```

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for

**Version**

information about the minimum Windows service pack that is required by a Windows Installer version.

**See Also**

[Properties](#)

**[ParentProductCode](#)**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ParentProductCode Property

During a concurrent installation, the installer sets the **ParentProductCode** property in the concurrent installation's session to the same value as the **ProductCode** property in the parent installation's session. Parent installations run the concurrent installation actions to perform a concurrent installation. An installation package can determine whether it is being installed by a concurrent installation action by checking the value of this property.

**Note** Concurrent Installations are not recommended for the installation of applications intended for release to the public. For information about concurrent installations please see [Concurrent Installations](#).

**Note** This property is not set if the concurrent installation is being run by the [RemoveExistingProducts](#) action.

## Remarks

To prevent a package from ever being installed as a concurrent installation, add either of the following conditional statements to the [LaunchCondition](#) table. This prevents the package from ever being installed by a concurrent installation action run by another installation. This does not prevent the package from being installed by the [RemoveExistingProducts](#) action.

```
"Not ParentProductCode"
```

```
"Not ParentOriginalDatabase"
```

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for

**Version**

information about the minimum Windows service pack that is required by a Windows Installer version.

**See Also**

[Properties](#)

[ParentOriginalDatabase](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PATCH Property

The installer sets the **PATCH** property to a list of patches being applied by calling [MsiApplyPatch](#), [MsiApplyMultiplePatches](#) or the /p [Command Line Option](#). You can also set the **PATCH** property on the command line while installing a package using [MsiInstallProduct](#) or the /i Command Line Option.

The value of the **PATCH** property is a list of the patches that are being installed. Each patch in the list is represented by the full path to the patch's package (.msp file.) The full paths in the list are separated by semicolons.

**Windows Installer 2.0:** Multiple patches are not supported. Windows Installer 3.0 is required to apply multiple patches.

## Remarks

If you create a patch package using [Msimsp.exe](#) and [Patchwiz.dll](#) you can specify that an action or a dialog box only run when a particular patch is being applied. When you create the patch package, for example test.msp, you author an upgraded image of the product and a patch creation properties file. When authoring the patch creation properties file you can enter a property name, for example PATCHFORTEST, in the MediaSrcPropName field of the [ImageFamilies](#) table. When you author the sequence tables of the upgraded image of the product, you can include in the Condition column of the sequence table a conditional statement for the action or dialog box you want to make conditional.

For example, you can use the following conditional statement to run an action or dialog box only when test.msp is being applied.

```
PATCH AND PATCHFORTEST AND PATCH >< PATCHFORTEST
```

**Note** Because the **PATCH** property can contain multiple patches, use the substring operator "><" to test for the presence of a particular patch rather than the equals operator "=". For more information about conditional statements see the [Conditional Statement Syntax](#) section.

The installer sets both properties if you apply a list of patches that includes test.msp. For example, you can use the /p Command Line

Option to apply a list of two patches.

**msiexec /qb /p**

**\\scratch\scratch\XYZ\Patches\test.msp;\\scratch\scratch\XYZ\bar.ms**

The installer sets the **PATCH** and PATCHFORTEST properties as follows.

```
PATCH=\\scratch\scratch\XYZ\Patches\test.msp;\\scratch\scratch\XYZ  
PATCHFORTEST=\\scratch\scratch\XYZ\Patches\test.msp
```

In this case, the condition is TRUE and the above conditional action or dialog box can run for each patch being installed, test.msp and bar.msp.

If test.msp is not being applied, the installer does not include it in the **PATCH** property and does not set PATCHFORTEST. In this case, the condition above is FALSE and the conditional action or dialog box does not run.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Conditional Statement Syntax](#)

[Examples of Conditional Statement Syntax](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PATCHNEWPACKAGECODE

## Property

The **PATCHNEWPACKAGECODE** property updates the [Revision Number Summary](#) property of an administrative image during patching. This property is only set by a transform in a .msp file. The .msp file must include a transform that adds this property to the [Property table](#) and sets its value. The installer then writes the value of **PATCHNEWPACKAGECODE** to the **Revision Number Summary** property.

### Remarks

The **PATCHNEWPACKAGECODE**, **PATCHNEWSUMMARYCOMMENTS**, and **PATCHNEWSUMMARYSUBJECT** properties are used to update the summary information when a patch is installed to an administrative image.

### Requirements

#### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

### See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PATCHNEWSUMMARYCOMMENTS

## Property

The **PATCHNEWSUMMARYCOMMENTS** property updates the **Comments Summary** property of an administrative installation during patching. This property is only set by a transform in a .msp file. The .msp file must include a transform that adds this property to the **Property table** and sets its value. The installer then writes the value of **PATCHNEWSUMMARYCOMMENTS** to the **Revision Number Summary** property.

### Remarks

The **PATCHNEWPACKAGECODE**, **PATCHNEWSUMMARYCOMMENTS**, and **PATCHNEWSUMMARYSUBJECT** properties are used to update the summary information when a patch is installed to an administrative image.

### Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

### See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PATCHNEWSUMMARYSUBJECT

## Property

The **PATCHNEWSUMMARYSUBJECT** property updates the [Subject Summary](#) property of an administrative image during patching. This property is only set by a transform in a .msp file. The .msp file must include a transform that adds this property to the [Property table](#) and sets its value. The installer then writes the value of **PATCHNEWSUMMARYSUBJECT** to the [Revision Number Summary](#) property.

### Remarks

The [PATCHNEWPACKAGECODE](#), [PATCHNEWSUMMARYCOMMENTS](#), and **PATCHNEWSUMMARYSUBJECT** properties are used to update the summary information when a patch is installed to an administrative image.

### Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

### See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PersonalFolder Property

The installer sets the **PersonalFolder** property to the full path of the Personal folder for the current user.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PhysicalMemory Property

The installer sets the **PhysicalMemory** property to the size of the installed RAM in megabytes.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PIDKEY Property

The **PIDKEY** property contains the part of the Product ID entered by the user. The [ValidateProductID action](#) validates the Product ID entered by comparing the **PIDKEY** property to the **PIDTemplate** property.

The **PIDKEY** property can be set on the command line or through the user interface. This property contains only the part of the Product ID that is entered by the user.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PIDTemplate Property

The **PIDTemplate** property contains the string used as a template for the **PIDKEY** property. For the syntax used in the template, see the [MaskedEdit](#) control type. [ValidateProductID Action](#) uses this value to validate the Product ID.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Preselected Property

The **Preselected** property indicates that features have already been selected and that the selection dialog need not be shown.

Conditional expressions which depend on whether features are already installed use this value. For example, the property can be used to suppress any dialogs involving feature selections during the resumption of a suspended installation.

The installer sets the **Preselected** property to a value of "1" during the resumption of a suspended installation or when any of the following properties are specified on the command line. If the **Preselected** property has been set to 1, the installer does not use the [Condition table](#) to evaluate the selection of features. Features are installed based upon the following properties. The installer always evaluates these properties in the following order:

1. **ADDLOCAL**
2. **REMOVE**
3. **ADDSOURCE**
4. **ADDDEFAULT**
5. **REINSTALL**
6. **ADVERTISE**
7. **COMPADDLOCAL**
8. **COMPADDSOURCE**
9. **COMPADDDEFAULT**
10. **FILEADDLOCAL**
11. **FILEADDSOURCE**
12. **FILEADDDEFAULT**

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows

## Version

Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PRIMARYFOLDER Property

The **PRIMARYFOLDER** is a global property that allows the author to designate a primary folder for the installation. The value for this property must be the key name of a directory that exists in the [Directory table](#).

The installer uses the resolved path of the primary folder to determine which volume to use when setting the values of the [PrimaryVolumePath](#) property, [PrimaryVolumeSpaceAvailable](#) property, [PrimaryVolumeSpaceRequired](#) property, and [PrimaryVolumeSpaceRemaining](#) property. The installer provides the values of these latter properties to users in the user interface to help them manage disk space. Commonly package authors can select the largest folder as the primary folder.

The value for this property must be the key name of a directory record found in the [Directory table](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PrimaryVolumePath Property

The installer sets the value of the **PrimaryVolumePath** property to the path of the volume designated by the **PRIMARYFOLDER** property. For example, if the folder referenced by **PRIMARYFOLDER** resolves to "D:\ProgramFiles", **PrimaryVolumePath** is set to "D:".

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PrimaryVolumeSpaceAvailable Property

The installer sets the value of the **PrimaryVolumeSpaceAvailable** property to a string that represents the total number of bytes available, in units of 512, on the volume referenced by the [PrimaryVolumePath](#) property.

## Remarks

For example, if **PrimaryVolumePath** is set to "D:", and volume D: has 446,134,272 bytes free, **PrimaryVolumeSpaceAvailable** is set to 871356.

Note if this value is to be displayed within a static [Text control](#), the [FormatSize](#) bit can be used to automatically format and label this number in units of kilobytes or megabytes as appropriate.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PrimaryVolumeSpaceRemaining Property

The installer sets the value of the **PrimaryVolumeSpaceRemaining** property to a string representing the total number of bytes remaining on the volume referenced by the [PrimaryVolumePath](#) property, if all currently selected features were installed. As with the [PrimaryVolumeSpaceAvailable](#) property, this number is expressed in units of 512 bytes.

## Remarks

Note if this value is to be displayed within a static [Text control](#), the [FormatSize](#) bit can be used to automatically format and label this number in units of kilobytes or megabytes as appropriate.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
----------------	--

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PrimaryVolumeSpaceRequired Property

The installer sets the value of the **PrimaryVolumeSpaceRequired** property to a string representing the total number of bytes required by all selected features on the volume referenced by the [PrimaryVolumePath](#) property. As with the [PrimaryVolumeSpaceAvailable](#) property, this number is expressed in units of 512 bytes.

## Remarks

Note if this value is to be displayed within a static [Text control](#), the [FormatSize](#) bit can be used to automatically format and label this number in units of kilobytes or megabytes as appropriate.

## Requirements

Version
Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PrintHoodFolder Property

The **PrintHoodFolder** property is the full path to the PrintHood folder.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Privileged Property

The **Privileged** property indicates whether the installation is performed in the context of elevated privileges. The installer sets this property if the user has administrator privileges, if the application has been assigned by a system administrator, or if both the user and machine policies `AlwaysInstallElevated` are set to true.

## Default Value

The installer does not set this property if the user is not allowed to install with elevated privileges.

## Remarks

Developers of installer packages can use the **Privileged** property to make the installation conditional upon system policy, the user being an administrator, or assignment by an administrator.

When running on Windows Vista, the **Privileged** and **AdminUser** are the same.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ProductCode Property

The **ProductCode** property is a unique identifier for the particular product release, represented as a string [GUID](#), for example "{12345678-1234-1234-1234-123456789012}". Letters used in this GUID must be uppercase. This ID must vary for different versions and languages.

A product upgrade that updates a product into an entirely new product must also change the product code. The 32-bit and 64-bit versions of an application's package must be assigned different product codes.

The **ProductCode** is advertised as a product property, and is the primary method of specifying the product. This property is **REQUIRED**.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ProductID Property

The **ProductID** property is set to the full **ProductID** after a successful validation.

This value is displayed by the UI and used by the [RegisterProduct action](#).

Following a successful validation, this property is set by the [ValidateProductID action](#) or the [ValidateProductID ControlEvent](#).

Note that the particular sample user interface provided with the SDK as Uisample.msi requires a value for **ProductID**. If you use this UI, but do not want to use **ProductID** to validate product identifiers, then set the **ProductID** property to "none" in the [Property table](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ProductLanguage Property

The **ProductLanguage** property specifies the language the installer should use for any strings in the user interface that are not authored into the database. This property must be a numeric language identifier (LANGID). If a transform changes the language of the user interface in the database, then it should also change the value of this property to reflect the new language.

This property is REQUIRED.

## Remarks

The value of the **ProductLanguage** property must be one of the languages listed in the [Template Summary](#) property.

When authoring a package as language-neutral, set the **ProductLanguage** property to 0 and use the MS Shell Dlg font as the text style for all authored dialogs. Because some fonts do not support all the character sets, by using this font you can ensure that the text is correctly displayed on all localized versions of the operating system.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ProductName Property

The **ProductName** property contains the name of the application being installed. This is used only for display purposes. Advertised as a product property.

This property is REQUIRED.

This property can be changed by a transform.

## Remarks

The **ProductName** property can be no greater than 63 characters in length. No limit exists on the length of the registry key for DisplayName.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ProductState Property

The installer sets the **ProductState** property to the installation state for the product at initialization. This property is set to one of the following **INSTALLSTATE** data types returned by [MsiQueryProductState](#).

<b>INSTALLSTATE</b>	<b>Numeric value</b>	<b>Installation state of product</b>
INSTALLSTATE_UNKNOWN	-1	The product is neither advertised or installed.
INSTALLSTATE_ADVERTISED	1	The product is advertised but not installed.
INSTALLSTATE_ABSENT	2	The product is installed for a different user.
INSTALLSTATE_DEFAULT	5	The product is installed for the current user.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ProductVersion Property

The value of the **ProductVersion** property is the version of the product in string format. This property is REQUIRED.

The format of the string is as follows:

major.minor.build

The first field is the major version and has a maximum value of 255. The second field is the minor version and has a maximum value of 255. The third field is called the build version or the update version and has a maximum value of 65,535.

## Remarks

At least one of the three fields of **ProductVersion** must change for an upgrade using the [Upgrade table](#). Any update that changes only the package code, but leaves **ProductVersion** and **ProductCode** unchanged is called a [small update](#). The three versions fields are provided primarily for convenience. For example, if you want to change **ProductVersion**, but don't want to change either the major or minor versions, you can change the build version.

Note that Windows Installer uses only the first three fields of the product version. If you include a fourth field in your product version, the installer ignores the fourth field.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Patching and Upgrades](#)

[small update](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ProgramFiles64Folder Property

The installer sets the **ProgramFiles64Folder** property to the full path of the predefined 64-bit Program Files folder. The existing **ProgramFilesFolder** property is set to the corresponding 32-bit folder.

## Remarks

The installer sets this property. For example, on 64-bit Windows, the value may be C:\Program Files. This property is not used on 32-bit Windows.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ProgramFilesFolder Property

The installer sets the **ProgramFilesFolder** property to the full path of the Program Files folder.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ProgramMenuFolder Property

The installer sets the **ProgramMenuFolder** property to the full path of the Program Menu folder for the current user. If an "All Users" profile exists and the **ALLUSERS** property is set, then this property is set to the folder in the "All Users" profile.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PROMPTROLLBACKCOST Property

The **PROMPTROLLBACKCOST** property specifies the action the installer takes if [rollback installation](#) capabilities are enabled and there is insufficient disk space to complete the installation.

The possible values of **PROMPTROLLBACKCOST** are as follows.

Value	Action
P	Display a dialog box asking whether to continue without a rollback.
D	Disable rollback and continue without asking user.
F	Fail with the out-of-disk-space error prompt.

## Remarks

When the user interface is run at the Basic or no UI level, the installer handles all the out-of-disk-space logic automatically.

When the user interface runs at the Full level, the user can be given additional options, such as prompting before proceeding with a rollback, disabling rollback, or proceeding without rollback when the disk is full. It is up to the package developer to author the dialog box sequence to provide the appropriate choices to the user. The elements available to do this are the [EnableRollback ControlEvent](#), [OutOfDiskSpace](#) property, [OutOfNoRbDiskSpace](#) property, and the **PROMPTROLLBACKCOST** property.

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service

**Version**

pack that is required by a Windows Installer version.

**See Also**

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# REBOOT Property

The **REBOOT** property suppresses certain prompts for a restart of the system. An administrator typically uses this property with a series of installations to install several products at the same time with only one restart at the end. For more information, see [System Reboots](#).

The [ForceReboot](#) and [ScheduleReboot](#) actions inform the installer to prompt the user to restart the system. The installer can also determine that a restart is necessary whether there are any ForceReboot or ScheduleReboot actions in the sequence. For example, the installer automatically prompts for a restart if it needs to replace any files in use during the installation.

You can suppress certain prompts for restarts by setting the **REBOOT** property as follows.

<b>REBOOT value</b>	<b>Description</b>
Force	Always prompt for a restart at the end of the installation. The UI always prompts the user with an option to restart at the end. If there is no user interface, and this is not a <a href="#">multiple-package installation</a> , the system automatically restarts at the end of the installation. If this is a multiple-package installation, there is no automatic restart of the system and the installer returns <code>ERROR_SUCCESS_REBOOT_REQUIRED</code> .
Suppress	Suppress prompts for a restart at the end of the installation. The installer still prompts the user with an option to restart during the installation whenever it encounters the <a href="#">ForceReboot action</a> . If there is no user interface, the system automatically restarts at each ForceReboot. Restarts at the end of the installation (for example, caused by an attempt to install a file in use) are suppressed.
ReallySuppress	Suppress all restarts and restart prompts initiated by ForceReboot during the installation. Suppress all restarts and restart prompts at the end of the installation. Both the restart prompt and the restart itself are suppressed. For

example, restarts at the end of the installation, caused by an attempt to install a file in use, are suppressed.

## Remarks

The installer only evaluates the first character of the **REBOOT** property.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[REBOOTPROMPT Property](#)

[System Reboots](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# REBOOTPROMPT Property

If the **REBOOTPROMPT** property is set to Suppress (or just S) any reboot performed by the Windows Installer happens automatically without interaction from the user. Setting this property does not initiate a reboot if one is not needed, it only suppresses the display of any prompts for reboots to the user.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[System Reboots](#)

[REBOOT Property](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RecentFolder Property

The **RecentFolder** property is the full path to the Recent folder.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RedirectedDLLSupport Property

The installer sets the **RedirectedDLLSupport** property if the system platform performing the installation supports [Isolated Components](#).

The installer sets the **RedirectedDLLSupport** property to a value of 1 if the system performing the installation supports Isolated Components based on .LOCAL files. The installer sets the **RedirectedDLLSupport** property to a value of 2 if the system that performs the installation supports isolation using either .LOCAL files or Win32 side-by-side assemblies.

The **RedirectedDLLSupport** property can be used to condition the [IsolateComponents action](#) in the [InstallUISequence table](#) or [InstallExecuteSequence table](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# REINSTALL Property

The value of the **REINSTALL** property is a list of features delimited by commas that are to be reinstalled. The features listed must be present in the Feature column of the [Feature](#) table. To reinstall all features use REINSTALL=ALL on the command line.

## Remarks

Note that the feature names are case-sensitive.

If the **REINSTALL** property is set, the **REINSTALLMODE** property should also be set, to indicate the type of reinstall to be performed. If the **REINSTALLMODE** property is not set, then by default all files that are currently installed are reinstalled, IF the currently installed file is a lesser version (or is not present). By default, no registry entries are rewritten.

Note that even if **REINSTALL** is set to ALL, only those features that were already installed previously are reinstalled. Thus, if **REINSTALL** is set for a product that is yet to be installed, no installation action will take place at all.

The installer always evaluates the following properties in the following order:

1. **ADDLOCAL**
2. **REMOVE**
3. **ADDSOURCE**
4. **ADDDEFAULT**
5. **REINSTALL**
6. **ADVERTISE**
7. **COMPADDLOCAL**
8. **COMPADDSOURCE**
9. **FILEADDLOCAL**
10. **FILEADDSOURCE**
11. **FILEADDDEFAULT**

For example, if the command line specifies: ADDLOCAL=ALL, ADDSOURCE = MyFeature, all the features are first set to run-local and then MyFeature is set to run-from-source. If the command line is: ADDSOURCE=ALL, ADDLOCAL=MyFeature, first MyFeature is set to run-local, and then when ADDSOURCE=ALL is evaluated, all features (including MyFeature) are reset to run-from-source.

The installer sets the **Preselected** property to a value of "1" during the resumption of a suspended installation or when any of the above properties are specified on the command line.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# REINSTALLMODE Property

The **REINSTALLMODE** property is a string that contains letters specifying the type of reinstall to perform. Options are case-insensitive and order-independent. This property should normally always be used in conjunction with the **REINSTALL** property. However, this property can also be used during installation, not just reinstall.

**Note** The Windows Installer ignores the **REINSTALLMODE** property during an [administrative installation](#).

## Reinstall Option Codes

By default the **REINSTALLMODE** is "omus".

Code Option	
p	Reinstall only if the file is missing.
o	Reinstall if the file is missing or is an older version.
e	Reinstall if the file is missing, or is an equal or older version.
d	Reinstall if the file is missing or a different version is present.
c	Verify the checksum values, and reinstall the file if they are missing or corrupt. This flag only repairs files that have msidbFileAttributesChecksum in the Attributes column of the <a href="#">File Table</a> .
a	Force all files to be reinstalled, regardless of checksum or version.
u	Rewrite all required registry entries from the <a href="#">Registry Table</a> that go to the <b>HKEY_CURRENT_USER</b> or <b>HKEY_USERS</b> registry hive.
m	Rewrite all required registry entries from the <a href="#">Registry Table</a> that go to the <b>HKEY_LOCAL_MACHINE</b>

	<p>or <b>HKEY_CLASSES_ROOT</b></p> <p>registry hive. Rewrite all information from the <a href="#">Class Table</a>, <a href="#">Verb Table</a>, <a href="#">PublishComponent Table</a>, <a href="#">ProgID Table</a>, <a href="#">MIME Table</a>, <a href="#">Icon Table</a>, <a href="#">Extension Table</a>, and <a href="#">AppID Table</a> regardless of machine or user assignment. Reinstall all <b>qualified components</b>. When reinstalling an application, this option runs the <a href="#">RegisterTypeLibraries</a> and <a href="#">InstallODBC</a> actions.</p>
s	Reinstall all shortcuts and re-cache all icons overwriting any existing shortcuts and icons.
v	Use to run from the source package and re-cache the local package. Do not use the v reinstall option code for the first installation of an application or feature.

If the **REINSTALLMODE** property is defined without also defining the **REINSTALL** property, then the specified "detection" modes still apply and specify the "overwrite" mode for a normal installation. The **REINSTALLMODE** property only affects those features that are selected normally for installation. The presence of the **REINSTALLMODE** property does not reinstall features. The reinstallation of features requires the presence of the **REINSTALL** property.

The option codes for this property correspond to the [command-line option /f](#). The command-line option has a default value of 'pecms'.

**Note** Only those files containing checksum information are ever verified and repaired. The REINSTALLMODE\_FILEVERIFY flag (the ccode above) only repairs files that have msidbFileAttributesChecksum in the Attributes column of the [File Table](#).

## Requirements

	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See
--	---

## Version

the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RemoteAdminTS Property

The installer sets the **RemoteAdminTS** property when the system is configured for Remote Administration using Windows Terminal Services. Note that in this case the installer does not set the **TerminalServer** property.

## Default Value

None. The property is undefined when not running on a remote administration server with Terminal Services.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# REMOVE Property

The value of the **REMOVE** property is a list of features delimited by commas that are to be removed. The features must be present in the Feature column of the [Feature table](#). Note that if you use REMOVE=ALL on the command line, the installer removes all features having an install level greater than 0. In this case, the installer does not remove features having an install level of 0. For more information about the install level of features see [Feature table](#).

## Remarks

To determine whether a product has been set to be completely uninstalled, a package author may use a conditional expression to check whether REMOVE=ALL. Note that if the product is removed by setting its top feature to absent, the **REMOVE** property may not equal ALL until after the [InstallValidate action](#). This means that any custom action that depends upon REMOVE=ALL must be sequenced after the InstallValidate. For more information see also [Conditioning Actions to Run During Removal](#). Note that the feature names are case-sensitive.

The installer always evaluates the following properties in the following order:

1. **ADDLOCAL**
2. **REMOVE**
3. **ADDSOURCE**
4. **ADDDEFAULT**
5. **REINSTALL**
6. **ADVERTISE**
7. **COMPADDLOCAL**
8. **COMPADDSOURCE**
9. **COMPADDDEFAULT**
10. **FILEADDLOCAL**
11. **FILEADDSOURCE**

## 12. FILEADDDEFAULT

For example, if the command line specifies ADDLOCAL=ALL, ADDSOURCE = MyFeature, all the features are first set to run-local and then MyFeature is set to run-from-source. If the command line is ADDSOURCE=ALL, ADDLOCAL=MyFeature, first MyFeature is set to run-local, then when ADDSOURCE=ALL is evaluated, all features (including MyFeature) are reset to run-from-source.

The installer sets the **Preselected** property to a value of "1" during the resumption of a suspended installation or when any of the above properties are specified on the command line.

### Requirements

#### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

### See Also

#### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ReplacedInUseFiles Property

The **ReplacedInUseFiles** property is set if the installer writes over a file that is being held in use. This property is used by custom actions to detect that a reboot is required to complete the installation.

Set during the [InstallExecute](#) and [InstallFinalize](#) actions.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RestrictedUserControl Property

The **RestrictedUserControl** property should be set to 1 if restricting which [public properties](#) can be sent to the server side. Authors can use this property in conditional statement to make the execution of dialogs or actions depend upon whether there are restrictions on public properties.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Restricted Public Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RESUME Property

The **RESUME** property is set when resuming a suspended installation. This property can determine what text to display in the UI. For example, when not in resume mode the user could be asked "Are you ready to install?" while in resume mode the user could be asked "Are you ready to complete your installation?".

## Remarks

For backward-compatibility, the installer also supports the name **Resume** for this property.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RollbackDisabled Property

The installer sets the **RollbackDisabled** property when rollback has been disabled. **RollbackDisabled** is used by package authors who need to ensure that the installer has not disabled rollback. The **RollbackDisabled** property can be used in a conditional expression that effectively refuses to continue with the installation if **RollbackDisabled** property is set.

## Default Value

By default, rollback is enabled.

## Remarks

Because [rollback](#) and [commit](#) do not run while rollback is disabled, the installer cannot properly install a package that uses these types of custom actions in a transaction during the installation. In this case, the author of the package should include a condition using `DisableRollback` that prevents the installation from continuing if rollback is disabled.

The `DisableRollback` policy value can be set by an administrator as a part of assigning system policy. Administrators are advised to not disable rollback unless necessary. For more information about `DisableRollback` policy value, see [System Policy](#).

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Rollback Installation](#)

[System Policy](#)

[rollback custom actions](#)

[commit custom actions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ROOTDRIVE Property

The **ROOTDRIVE** property specifies the default drive for the destination directory of the installation. If the Directory column of the [Directory table](#) indicates the root destination directory by a property name that is undefined, the installer uses the value of the **ROOTDRIVE** property to resolve the path to the destination directory.

If **ROOTDRIVE** is not set at a command line or authored into the [Property table](#), the installer sets this property. During an [administrative installation](#) the installer sets **ROOTDRIVE** to the first connected network drive it finds that can be written to. If it is not an administrative installation, or if the installer can find no network drives, the installer sets **ROOTDRIVE** to the local drive that can be written to having the most free space.

The value for this property must end with '\'.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ScreenX Property

The installer sets the **ScreenX** property to the width of the screen in pixels.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ScreenY Property

The installer sets the **ScreenY** property to the height of the screen in pixels.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SecureCustomProperties Property

The **SecureCustomProperties** is a list of [public properties](#) delimited by semi-colons. These properties are included with the default list of restricted public properties that the installer can pass to the server side when doing a managed installation with elevated privileges.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Restricted Public Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SendToFolder Property

The installer sets the **SendToFolder** property to the full path of the SendTo folder for the current user.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SEQUENCE Property

The **SEQUENCE** property specifies a table having the same schema as the InstallExecuteSequence table, that is Action, Condition, and Sequence columns. This property is used by the [SEQUENCE action](#).

## Remarks

The SEQUENCE action runs the actions in the table in the order specified by the Sequence column of the table.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ServicePackLevel Property

The installer sets the **ServicePackLevel** property to the numerical value of the operating system service pack level, if one is installed.

## Remarks

The minor revision number, if there is one, is held by the **ServicePackLevelMinor** property.

For more information, see [Operating System Property Values](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ServicePackLevelMinor Property

The **ServicePackLevelMinor** property is used in conjunction with the [ServicePackLevel](#) property. It holds the minor revision number of the service pack installed on the machine.

## Remarks

This property is set only on Windows 2000.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SharedWindows Property

The installer sets the **SharedWindows** property when the system is operating as Shared Windows.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ShellAdvtSupport Property

The **ShellAdvtSupport** property is set by the installer if the system's **IShellLink** interface supports installer descriptor resolution.

## Remarks

If this property is set, the system's **IShellLink** interface supports installer descriptor resolution. This is supported by Windows 2000 and systems running Internet Explorer 4.01. If this property is not set, the installer has the default behavior of creating non-advertised features during installation, either locally or run from source.

## Requirements

Version
Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Platform Support of Advertisement](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SHORTFILENAMES Property

Setting the **SHORTFILENAMES** property causes short file names to be used for the names of target files, rather than long file names. It does not affect the file names the installer looks for at the source.

## Default Value

Long names are used by the installer if the **SHORTFILENAMES** property is not set and the target volume supports long names. Short names are used by the installer if the target volume does not support long names.

## Remarks

When the **SHORTFILENAMES** property is set, the installer creates folders and installs files using short file names from any short|long pairs listed in the [File table](#) or [Directory table](#).

Applications can use the **GetShortPathName** function to retrieve the short path form of a specified file path.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SourceDir Property

The **SourceDir** property is the root directory that contains the source cabinet file or the source file tree of the installation package. This value is used for directory resolution.

## Default Value

The directory that contains the installation package.

## Remarks

The [ResolveSource action](#) must be called before using the **SourceDir** property in any expression or attempting to retrieve the value of **SourceDir** with [MsiGetProperty](#). The ResolveSource action should not be run while the source is unavailable, such as when uninstalling an application, because this can cause an unintended prompt for the source media.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SOURCELIST Property

The **SOURCELIST** property is a semicolon-delimited list of network or URL source paths to the application's installation package. This list is appended to the end of each user's existing source list for the application. The installer locates a source by enumerating the list of source paths and uses the first accessible location it finds. Only this source can be used for the remainder of the installation. Each path specified in the source list must therefore be to a location having a complete source for the application. The entire directory tree at each source location must be the same and must include all of the required source files, including any cabinets. Each location must have an .msi file with the same file name and product code.

## Default Value

The installer only checks the **SOURCELIST** property if the product has not already been advertised or installed. In all other cases the installer uses the existing source list that is in the registry.

## Remarks

Sources added using the **SOURCELIST** property are added directly to the end of the list for each type of source and always come after the default source specified by the [SourceDir](#) property.

For Windows Installer the number of sources in the **SOURCELIST** property is unlimited. [MsiSourceListAddSource](#) can be used to add network sources.

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for

**Version**

information about the minimum Windows service pack that is required by a Windows Installer version.

**See Also**

[Properties](#)

[Source Resiliency](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# StartMenuFolder Property

The installer sets the **StartMenuFolder** property to the full path of the Start Menu folder. By default, this property is set to the folder for the current user. If an "All Users" profile exists and the **ALLUSERS** property is set, then this property is set to the folder in the "All Users" profile.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# StartupFolder Property

The installer sets the **StartupFolder** property to the full path of the Startup folder. By default, this property is set to the folder for the current user. If an "All Users" profile exists and the **ALLUSERS** property is set, then this property is set to the folder in the "All Users" profile.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# System16Folder Property

Windows Installer sets the **System16Folder** property to the full path to folder for 16-bit system DLLs. Note that the **System16Folder** property is not defined for 64-bit Windows because this system does not support 16-bit applications or components.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# System64Folder Property

The installer sets the **System64Folder** property to the full path to the predefined System64 folder. The existing **System64Folder** property is set to the corresponding 32-bit folder.

## Remarks

The installer sets this property on 64-bit Windows. For example, on 64-bit Windows, the value may be C:\Window\System32. This property is not used on 32-bit Windows.

This folder is normally a subdirectory of the Windows folder. However, it resides on a server when configured for Shared Windows.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SystemFolder Property

The installer sets the **SystemFolder** property to the full path of the System folder.

## Remarks

The installer sets this property. For example, on 32-bit Windows the value may be C:\Windows\System32. On 64-bit Windows, the value may be C:\Windows\SysWow64.

This folder is normally a subdirectory of the Windows folder. However, it resides on a server when configured for Shared Windows.

This folder is local, even when configured for shared Windows.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SystemLanguageID Property

The **SystemLanguageID** property is the default language identifier for the system. The installer sets it by a call to **GetSystemDefaultLangID**.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# TARGETDIR Property

The **TARGETDIR** property specifies the root destination directory for the installation. **TARGETDIR** must be the name of one root in the [Directory](#) table. There may be only a single root destination directory. During an [administrative installation](#) this property specifies the location to copy the installation package. During a non-administrative installation this property specifies the root destination directory.

To specify the root destination directory, set the Directory column of the Directory table to the **TARGETDIR** property and the DefaultDir column to the [SourceDir](#) property. If the **TARGETDIR** property is defined, the destination directory is resolved to the property's value. If the **TARGETDIR** property is undefined, the [ROOTDRIVE](#) property is used to resolve the path. For more information about using the **TARGETDIR** property, see [Using the Directory Table](#).

Note that the value of the **TARGETDIR** property is typically set at the command line or through a user interface. Setting **TARGETDIR** by authoring a path into the [Property table](#) is not recommended because computers differ in the set up of the local drive.

For more information on how to change TARGETDIR during an installation see [Changing the Target Location for a Directory](#)

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# TempFolder Property

The installer sets the **TempFolder** property to the full path to the Temp folder.

Authors should not need to set the **TempFolder** property. Windows Installer uses the **GetTempPath** function to retrieve the path of the directory designated for temporary files and to set this property.

## Remarks

A common value for this property is C:\Temp.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# TemplateFolder Property

The installer sets the **TemplateFolder** property to the full path to the Template folder for the current user.

On Windows 2000 only, if the **ALLUSERS** property is set, this property points to a file system directory that contains the templates that are available to all users.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# TerminalServer Property

The installer sets the **TerminalServer** property when the system is a server with Terminal Services.

## Default Value

None. The property is undefined when not running on Terminal Services.

## Remarks

The [RemoteAdminTS](#) property can only be set on Microsoft Windows 2000 or later. The **TerminalServer** property can be set on Windows 2000.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[RemoteAdminTS property](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# TextHeight Property

The installer sets the **TextHeight** property to the height (ascent + descent) of characters in logical units.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Time Property

The **Time** property is the current time in hours, minutes, and seconds as a string of literal text in the format HH:MM:SS using a 24 hour clock. For example, the time 6:57 p.m. is represented by "18:57:00". The format of the value depends upon the user's locale, and is the format obtained using **GetTimeFormat** function with the TIME\_FORCE24HOURFORMAT | TIME\_NOTIMEMARKER option. The value of this property is set by the Windows Installer and not the package author.

## Remarks

Because this is a text string, it cannot be used in conditional expressions.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# TRANSFORMS Property

The **TRANSFORMS** property is a list of the transforms that the installer applies when installing the package. The installer applies the transforms in the same order as they are listed in the property. Transforms can be specified by their filename or full path. To specify multiple transforms, separate each file name or full path with a semicolon (;). For example, to apply three transforms to a package, set **TRANSFORMS** to a list of file names or to a list of full paths.

```
TRANSFORMS=transform1.mst;transform2.mst;transform3.mst  
TRANSFORMS=\\server\share\path\transform1.mst;\\server2\share\path\transform2.mst;transform3.mst
```

You can indicate that a transform file is embedded in a storage of the .msi file, rather than as a stand-alone file, by prefixing the filename with a colon (:). For example, the following example indicates that transform1.mst and transform2.mst are embedded inside the .msi file and that transform3.mst is a stand-alone file.

```
TRANSFORMS=:transform1.mst;:transform2.mst;transform3.mst
```

The installer requires the transforms listed in **TRANSFORMS** at every installation, advertisement, installation-on-demand, or maintenance installation of the package. The [TransformsSecure policy](#), the **TRANSFORMS** property, and the first character of the **TRANSFORMS** string informs the installer how to handle the source resiliency of stand-alone transform files. Windows Installer treats setting [TransformsAtSource policy](#) or **TRANSFORMSATSOURCE** the same as [TransformsSecure policy](#) and **TRANSFORMSSECURE**. Note that transforms embedded in the .msi file are not cached and are always obtained from the package.

## TRANSFORMS Property

@*[list of filenames]*

Example:

```
@transform1.mst;transform2.mst; transform3.mst
```

---

[[*list of paths*]

Example:

```
|\\server\share\path\transform1.mst;\\server2\share2\path2\
```

---

*[list of filenames]*

The first character is not @ or |.

Example:

transform1.mst;transform2.mst; transform3.mst

---

*[list of filenames]*

The first character is not @ or |.

Example:

transform1.mst;transform2.mst; transform3.mst

---

*[list of paths]*

The first character is not @ or |.

Example:

```
\\server\share\path\transform1.mst;\\server2\share2\path2\t
```

You cannot use filenames and paths together in the same **TRANSFORMS** list. You cannot specify secure and profile transforms together in the same list. You may include transforms embedded in the package in a list with other transforms.

```
@transform1.mst;:transform2.mst  
|\\server\share\path\transform1.mst;:transform2.mst
```

Note that because the list delimiter for transforms is the semicolon character, semicolons must not be used in a transform filename or path.

## Remarks

In cases where the TransformsSecure policy or the **TRANSFORMSSECURE** property has been set with Windows Installer, it is not necessary to pass the @ or | symbol when setting **TRANSFORMS** using the command line. The installer assumes Secure-At-Source or Secure-Full-Path if the list consists entirely of file names located at the source or consists entirely of full paths. You still cannot mix the two types of transform sources.

Note that the installer uses a different search order for unsecured

transforms applied during first time and maintenance installations. For more information, see [Unsecured Transforms](#).

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Database Transforms](#)

[Merges and Transforms](#)

[Source Resiliency](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# TRANSFORMSATSOURCE Property

Setting this property is like setting the [TransformsAtSource policy](#) policy except that the scope is different. Setting TransformsAtSource policy applies to all packages installed by a given user. Setting the **TRANSFORMSATSOURCE** property applies to the package regardless of the users.

## Remarks

Windows Installer interprets the **TRANSFORMSATSOURCE** property as though it were the **TRANSFORMSSECURE** property. If the @ flag is passed in the **TRANSFORMS** property, Windows Installer treats the transforms in the list as [secure-at-source transforms](#).

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# TRANSFORMSSECURE Property

Setting the **TRANSFORMSSECURE** property to 1 informs the installer that transforms are to be cached locally on the user's computer in a location where the user does not have write access. Setting this property is like setting the [TransformsSecure policy](#) except that the scope is different. Setting **TransformsSecure** policy applies to all packages installed by a given user. Setting the **TRANSFORMSSECURE** property applies to the package regardless of the user.

The purpose of this property is to provide for secure transform storage with traveling users of Windows 2000. When this property is set, a [maintenance installation](#) can only use the transform from the specified path. If the path is not available the maintenance installation fails. A source for each secure transform must therefore reside at the location of the source of the installation package. Then if the installer finds that the transform is missing on the local computer, it can restore the transform from this source.

## Remarks

Windows Installer interprets the **TRANSFORMSATSOURCE** property to be the same as the **TRANSFORMSSECURE** property.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

## Database Transforms

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# TTCSupport Property

The installer sets the **TTCSupport** property to indicate whether the operating system supports .ttc (true type font collections) files.

## Remarks

This property is set if the installer detects Windows 2000 or any of the following operating systems:

- Japan - 932
- China - 936
- Taiwan - 950
- Korea - 949
- Hong Kong SAR - 950

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UILevel Property

The installer sets the **UILevel** property to the level of the user interface. The internal UI is enabled and set by [MsiSetInternalUI](#). This property is set to one of the following INSTALLUILEVEL data types.

<b>INSTALLUILEVEL</b>	<b>Numeric value</b>	<b>User interface level</b>
INSTALLUILEVEL_NONE	2	Completely silent installation.
INSTALLUILEVEL_BASIC	3	Simple progress and error handling.
INSTALLUILEVEL_REDUCED	4	Authored UI, wizard dialogs suppressed.
INSTALLUILEVEL_FULL	5	Authored UI with wizards, progress, errors.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[User Interface](#)

[Determining UI Level from a Custom Action](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UpdateStarted Property

The installer sets the **UpdateStarted** property when changes to the system have begun for this installation, including resuming a suspended installation.

UI condition statements use this value to select a dialog box. If this property is set, the user is asked after an error and cancellation whether to restore or to continue later.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UpgradeCode Property

The **UpgradeCode** property is a GUID representing a related set of products. The **UpgradeCode** is used in the [Upgrade Table](#) to search for related versions of the product that are already installed.

This property is used by the [RegisterProduct action](#).

## Remarks

It is strongly recommended that authors of installation packages specify an **UpgradeCode** for their application.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	---

## See Also

[Properties](#)

[Patching and Upgrades](#)

[Preparing an Application for Future Major Upgrades](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UPGRADINGPRODUCTCODE

## Property

The **UPGRADINGPRODUCTCODE** property is set by Windows Installer when an upgrade removes an application. The installer sets this property when it runs the [RemoveExistingProducts action](#). This property is not set by removing an application using the Add or Remove Programs in Control Panel. An application determines whether it is being removed by an upgrade or the Add or Remove Programs by checking **UPGRADINGPRODUCTCODE**.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UserLanguageID Property

The **UserLanguageID** property is the default language identifier for the current user. The installer sets this property by calling [GetUserDefaultLangID](#).

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# USERNAME Property

The **USERNAME** property is a user performing installation.

## Default Value

Default value may be put in the Property table. If the **USERNAME** property is not set, then it is set automatically using values from the registry.

## Remarks

Set the **NOUSERNAME** property to suppress the automatic setting of **USERNAME**.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UserSID Property

The installer sets the value of the **UserSID** property to the string representation of the security identifier (SID) of the user running the installation. For more information, see [Authorization Structures](#).

## Default Value

None.

## Remarks

The Windows Installer set this property on Windows 2000, Windows XP and Windows Vista. This property is not defined on all other operating systems.

Note that this property has the special attribute that it can be retrieved from a deferred custom action. A custom action running with elevated privileges can still return the user's SID in **UserSID** property. For information, see [Obtaining Context Information for Deferred Execution Custom Actions](#).

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Version9X Property

The **Version9X** property gives the version number for Microsoft Windows 95, Windows 98, or Windows Me operating systems.

The value of this property is an integer: MajorVersion \* 100 + MinorVersion. The property remains undefined if the operating system is not Windows Me/98/95. For more information, see [Operating System Property Values](#).

## Remarks

Condition expressions can test for the version of Windows Me/98/95 by using the property name or may verify the version by using a comparison operator.

All property names are case-sensitive. Note that the X in the name **Version9X** is uppercase.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# VersionDatabase Property

The value of the **VersionDatabase** property is the numeric database version of the current installation.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000 See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# VersionMsi Property

The installer sets the **VersionMsi** property to the version of Windows Installer run during the installation.

## Remarks

For more information, see [Released Versions of Windows Installer](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000 See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	--

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# VersionNT Property

The installer sets the **VersionNT** property to the version number for the operating system, undefined if the operating system is not Windows NT, Windows 2000, Windows XP, Windows Vista, Windows Server 2008, or Windows 7. The value is an integer: MajorVersion \* 100 + MinorVersion.

Conditional statements that depend upon the operating system can use this property.

See also [Operating System Property Values](#).

## Remarks

Condition expressions can test for Windows NT, Windows 2000, or Windows XP by using the property name, or may verify the version by using a comparison operator.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000 See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# VersionNT64 Property

The installer sets the **VersionNT64** property to the version number for the operating system only if the system is running on a 64-bit computer. The property is undefined if the operating system is not 64-bit.

The value is an integer: MajorVersion \* 100 + MinorVersion.

## Remarks

Conditional expressions test for 64-bit Windows simply by using the property name, or by verifying the version using a comparison operator.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000 See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# VirtualMemory Property

The installer sets the **VirtualMemory** property to the amount of available page file space in megabytes.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000 See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# WindowsBuild Property

The installer sets the **WindowsBuild** property to the build number of the operating system. For more information, see [Operating System Property Values](#).

## Requirements

Version
Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000 See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# WindowsFolder Property

The installer sets the **WindowsFolder** property to the full path of the Windows folder.

## Remarks

This folder is local, even when configured for shared Windows.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000 See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
----------------	--

## See Also

[Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# WindowsVolume Property

The installer sets the **WindowsVolume** property to the volume of the windows folder. The property always ends with a backslash. This can be used to set the default location to the volume the Windows folder is on because the **ROOTDRIVE** property does not necessarily equal this drive.

## Remarks

Do not use the **WindowsVolume** property in the Directory column of the **Directory** table. The **WindowsFolder** property contains the path to the Windows folder.

## Requirements

<b>Version</b>
Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000 See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.

## See Also

### [Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Operating System Property Values

The table in this topic displays values for selected Windows Installer [Operating System Properties](#).

<b>System</b>	<b>VersionNT</b>	<b>WindowsBuild</b>	<b>ServicePackLevel</b>
Windows 2000	500	2195	Not applicable
Windows 2000 + Service Pack 1	500	2195	1
Windows 2000 + Service Pack 2	500	2195	2
Windows 2000 + Service Pack 3	500	2195	3
Windows 2000 + Service Pack 4	500	2195	4
Windows XP	501	2600	Not applicable
Windows XP with Service Pack 1 (SP1)	501	2600	1
Windows XP with Service Pack 2 (SP2)	501	2600	2
Windows XP with Service Pack 3 (SP3)	501	2600	3
Windows Server 2003	502	3790	Not applicable
Windows Server 2003 with Service Pack 1 (SP1)	502	3790	1
Windows Server 2003 with Service Pack 2 (SP2)	502	3790	2
Windows Vista	600	6000	Not applicable
Windows Vista with Service Pack 1 (SP1)	600	6001	1

Windows Vista with Service Pack 2 (SP2)	600	6002	2
Windows Server 2008	600	6001	Not applicable
Windows Server 2008 with Service Pack 2 (SP2)	600	6002	2
Windows Server 2008 R2	601	greater than 7100	Not applicable
Windows 7	601	greater than 7100	Not applicable

For more information, see [Released Versions of Windows Installer](#), [VersionNT](#), [WindowsBuild](#), and [ServicePackLevel](#).

## See Also

[About Properties](#)  
[Property Reference](#)  
[Using Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Uninstall Registry Key

The following installer properties give the values written under the registry key:

**HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\CurrentVersion**

The values are stored in a subkey identified by the application's product code GUID.

Value	Windows Installer property
DisplayName	<b>ProductName</b> property
DisplayVersion	Derived from <b>ProductVersion</b> property
Publisher	<b>Manufacturer</b> property
VersionMinor	Derived from <b>ProductVersion</b> property
VersionMajor	Derived from <b>ProductVersion</b> property
Version	Derived from <b>ProductVersion</b> property
HelpLink	<b>ARHELPLINK</b> property
HelpTelephone	<b>ARHELPTELEPHONE</b> property
InstallDate	Installation date
InstallLocation	<b>ARPINSTALLLOCATION</b> property
InstallSource	<b>SourceDir</b> property
URLInfoAbout	<b>ARPURLINFOABOUT</b> property
URLUpdateInfo	<b>ARPURLUPDATEINFO</b> property
AuthorizedCDFPrefix	<b>ARPAUTHORIZEDCDFPREFIX</b> property
Comments	<b>ARPCOMMENTS</b> property Comments provided to the <b>Add or Remove Programs</b> control panel.
Contact	<b>ARPCONTACT</b> property Contact provided to the <b>Add or Remove Programs</b>

	control panel.
EstimatedSize	Determined and set by the Windows Installer.
Language	<b>ProductLanguage</b> property
ModifyPath	Determined and set by the Windows Installer.
Readme	<b>ARPREADME</b> property Readme provided to the <b>Add or Remove Programs</b> control panel.
UninstallString	Determined and set by Windows Installer.
SettingsIdentifier	<b>MSIARPSETTINGSIDENTIFIER</b> property

## See Also

[About Properties](#)

[Configuring Add/Remove Programs with Windows Installer](#)

[Property Reference](#)

[Using Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Summary Information Stream

The material in this section is intended for developers who are writing their own setup programs who want to learn more about Windows Installer database tables. For general information, see [About Windows Installer](#).

The summary information stream is used by the installer for two purposes. First, it contains information about the package that can be viewed through Microsoft Windows Explorer. This information is accessible through the **IStream** interface. Second, it contains properties that are used by the installer to install the package. The following topics provide information about how to use the summary information stream with the installer:

- [About the Summary Information Stream](#)
- [Using the Summary Information Stream](#)
- [Summary Information Stream Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# About the Summary Information Stream

The summary information stream contains information about the package that can be viewed through Microsoft Windows Explorer. This information is accessible through the **IStream** interface. It is up to the author to provide the values for each of these properties.

The summary information stream uses COM to provide structured storage of databases. COM supports the concept of structured storage accessible through the **IStream** interface. Structured storage, in turn, supports the concept of property sets as a flexible method for serializing almost any information. The COM specification defines a single standard property set, summary information, which is used to populate property sheets viewable from Windows Explorer. So, information stored in the summary information stream can be viewed by users when they right-click an installer database or transform and select [Properties](#).

For a list of the summary property set see [Summary Information Stream Property Set](#).

For brief descriptions of the Summary Information properties used with databases, transforms, and patches, see [Summary Property Descriptions](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Summary Information Stream Property Set

The following table shows the summary information stream property set, which includes the properties, their corresponding property IDs, PIDs, and types. For more information about how the installer uses these properties, see [Summary Property Descriptions](#). For information about the Window Installer functions that are used to configure the summary information properties, see [Using the Summary Information Stream](#).

Property name	Property ID	PID	Type
<b>Codepage</b>	PID_CODEPAGE	1	VT_I2
<b>Title</b>	PID_TITLE	2	VT_LPSTR
<b>Subject</b>	PID_SUBJECT	3	VT_LPSTR
<b>Author</b>	PID_AUTHOR	4	VT_LPSTR
<b>Keywords</b>	PID_KEYWORDS	5	VT_LPSTR
<b>Comments</b>	PID_COMMENTS	6	VT_LPSTR
<b>Template</b>	PID_TEMPLATE	7	VT_LPSTR
<b>Last Saved By</b>	PID_LASTAUTHOR	8	VT_LPSTR
<b>Revision Number</b>	PID_REVNUMBER	9	VT_LPSTR
<b>Last Printed</b>	PID_LASTPRINTED	11	VT_FILETIME
<b>Create Time/Date</b>	PID_CREATE_DTM	12	VT_FILETIME
<b>Last Save Time/Date</b>	PID_LASTSAVE_DTM	13	VT_FILETIME
<b>Page Count</b>	PID_PAGECOUNT	14	VT_I4
<b>Word Count</b>	PID_WORDCOUNT	15	VT_I4
<b>Character Count</b>	PID_CHARCOUNT	16	VT_I4
<b>Creating Application</b>	PID_APPNAME	18	VT_LPSTR
<b>Security</b>	PID_SECURITY	19	VT_I4

The installer currently maintains three storage formats for installation packages, transforms, and patch packages. The CLSID for the storage is set to the appropriate format class for the particular format, independent of the summary information for the storage.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Summary Property Descriptions

Summary properties for installation packages, transforms, and patches are described in the following tables. Note that the meaning of a particular summary property can be different depending on whether the database belongs to an installation package, transform, or patch package. For more information about the property IDs, PIDs, and types, see [Summary Information Stream Property Set](#).

## Installation Packages

Summary property	Meaning of this property in an Installation database
<b>Title</b>	A description of this file as an installation package. The description should include the phrase " <a href="#">Installation Database</a> ."
<b>Subject</b>	The name of the product installed by this package. This should be the same name as in the <a href="#">ProductName</a> property.
<b>Author</b>	The name of the manufacturer of this product. This should be the same name as in the <a href="#">Manufacturer</a> property.
<b>Keywords</b>	A list of keywords that may be used by file browsers to do keyword searches for a file. The keywords should include "Installer" as well as product-specific keywords.
<b>Comments</b>	Contains the phrase: "This installer database contains the logic and data required to install <i>&lt;product name&gt;</i> ."
<b>Template (REQUIRED)</b>	The platform and languages compatible with this installation package. See <a href="#">Template</a> for syntax.
<b>Last Saved By</b>	Developers of database editing tools may use this property during the development process to track the last person to modify the installation database. This property should be set to Null in a final shipping database. The installer sets this property to the value of the <a href="#">LogonUser</a> property during an <a href="#">administrative installation</a> . The installer never uses this property and a user never needs to modify it.

<b>Revision Number</b> (REQUIRED)	Contains the <a href="#">package code</a> for the installer package.
<b>Last Printed</b>	May be set to the date and time during an <a href="#">administrative installation</a> to record when the administrative image was created.
<b>Create Time/Date</b>	The time and date when this installer database was created.
<b>Last Saved Time/Date</b>	The current system time and date at the time the installer database was last saved. Initially null.
<b>Page Count</b> (REQUIRED)	Contains a value used to identify the minimum installer version required by this installation package. For example, if the package requires at minimum the 2.0 version of the installer, this property should be set to an integer of 200. <b>Note</b> The value of this property must be 200 or greater with <a href="#">64-bit Windows Installer Packages</a> .
<b>Word Count</b> (REQUIRED)	The type of the source file image. Stored in place of the standard Count property. This property is a bit field. See the <a href="#">Word Count</a> topic for the values.  Starting with Windows Installer version 4.0 on Windows Vista, this property includes bits to specify whether elevated privileges are required.
<b>Character Count</b>	Null
<b>Creating Application</b>	Contains the name of the software used to author this installation database.
<b>Security</b>	The value of this property should be 2 - Recommended read-only.
<b>Codepage</b>	The numeric value of the ANSI code page used to display the Summary Information. This property must be set before any string properties are set in the summary information.

## Transforms

Summary property	Meaning of this property in a Transform
<b>Title</b>	A description of this file as a transform. The description should include the phrase: " <b>Transform</b> ."
<b>Subject</b>	The name of the product installed by the original installation package. This should be the same value as the <b>Subject Summary</b> property in the original installation package.
<b>Author</b>	The name of the manufacturer of this transform.
<b>Keywords</b>	A list of keywords that may be used by file browsers to do keyword searches for a file. The keywords should include "Installer" as well as product-specific keywords.
<b>Comments</b>	Contains the phrase: "This transform contains the logic and data required to install <i>&lt;product name&gt;</i> ."
<b>Template (REQUIRED)</b>	The platform and language versions compatible with this transform. This value may be left blank if there are no restrictions. Only one language can be specified for a transform. For more information about the syntax, see <b>Template</b> .
<b>Last Saved By</b>	The platform and language ID(s) that the database has after it has been transformed. The <b>Template Summary</b> property in the new database should be set to the same values.
<b>Revision Number (REQUIRED)</b>	A list of the product code GUIDs and version of the new and original products and the upgrade code GUID. The list is separated by semicolons as follows. <i>Original-Product-Code Original-Product-Version;New-Product Code New-Product-Version; Upgrade-Code</i>
<b>Last Printed</b>	Null
<b>Create Time/Date</b>	The time and date when the transform was created.
<b>Last Saved Time/Date</b>	The current system time and date at the time the transform was saved. Initially null.

<b>Page Count</b> (REQUIRED)	A value used to indicate the minimum installer version required to process the transform. The value should be set to the greater of the two <b>Page Count</b> property values that belong to the databases used to generate the transform.
<b>Word Count</b> (REQUIRED)	Null
<b>Character Count</b>	This part of the summary information stream is divided into two 16-bit words. The upper word contains <i>transform validation flags</i> . Lower word contains <i>transform error condition flags</i> .
<b>Creating Application</b>	Contains the name of the software used to create this transform.
<b>Security</b>	The value of this property should be 4 - Enforced read-only.
<b>Codepage</b>	The numeric value of the ANSI code page used to display the Summary Information. This property must be set before any string properties can be set in the summary information.

## Patch Packages

Summary property	Meaning of this property in a Patch package
<b>Title</b>	A description of this file as a patch package. The description should include the phrase: " <b>Patch</b> ."
<b>Subject</b>	A description of the patch that includes the name of the product.
<b>Author</b>	The name of the manufacturer of the patch package.
<b>Keywords</b>	A semicolon delimited list of sources of the patch.
<b>Comments</b>	Contains the phrase: "This patch contains the logic and data required to install <i>&lt;product name&gt;</i> ."
<b>Template</b> (REQUIRED)	A semicolon delimited list of the product codes that can accept this patch.

<b>Last Saved By</b>	A semicolon delimited list of transform substorage names in the order they are applied by this patch.
<b>Revision Number (REQUIRED)</b>	Contains the GUID patch code for the patch. This may be followed by a list of patch code GUIDs for patches that are removed when this patch is applied. The patch codes are concatenated with no delimiters separating GUIDs in the list. <b>Note</b> If the patch package contains a <b>MsiPatchSequence</b> table, the patch does not remove patches listed after the current patch's GUID.
<b>Last Printed</b>	Null
<b>Create Time/Date</b>	The time and date when patch file was created.
<b>Last Saved Time/Date</b>	The current system time and date at the time the patch package was saved. This value is initially null.
<b>Page Count (REQUIRED)</b>	Null
<b>Word Count (REQUIRED)</b>	Contains a value that indicates the minimum Windows Installer version that is required to install the patch. A patch with a word count value of 4 requires Windows Installer version 3.0 or higher for the patch to be applied. A value of 3 indicates that Windows Installer version 2.0 or higher is required. A value of 2 indicates that Windows Installer version 1.2 or higher is required. The default value is 1.
<b>Character Count</b>	Null
<b>Creating Application</b>	The name of the software used to create the patch.
<b>Security</b>	The value of this property should be 4 - Enforced read-only.
<b>Codepage</b>	The numeric value of the ANSI code page used to display the Summary Information. This property must be set before any string properties are set in the summary information.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using the Summary Information Stream

This section describes which functions in the Windows Installer API can call the summary information stream properties. For more information on the summary information stream and how it works with databases, see [About the Summary Information Stream](#).

- It is important to remember that the installer contains different types of databases, and some properties of the summary information stream have different meanings with different databases. For more information, see [Summary Property Descriptions](#).
- When a database is opened as the output of another database, the summary information stream of the output database is actually a read-only mirror of the original database and thus cannot be changed. Additionally, it will not be persisted with the database. To create or modify the summary information for the output database it must be closed and re-opened.

The following steps describe how to use the summary information stream functions:

## ▶ To use the summary information stream properties

1. Obtain a handle to the database containing the summary information stream by calling the [MsiGetSummaryInformation](#) function.
2. Call the [MsiSummaryInfoGetPropertyCount](#) function to obtain the number of existing properties.
3. Call the [MsiSummaryInfoGetProperty](#) function to view a single summary information property.
4. Call the [MsiSummaryInfo SetProperty](#) function to set a single property

5. Call the [MsiSummaryInfoPersist](#) function to save the summary information property.
6. Call the [MsiCreateTransformSummaryInfo](#) function to create the summary information for an existing transform.

[Orca.exe](#) and [Msiinfo.exe](#) are tools that can be used to edit or display the summary information stream of a database. These tools are only available in the [Windows SDK Components for Windows Installer Developers](#).

The summary information stream can also be accessed using the following methods and properties of the Windows Installer [Automation Interface](#).

- [SummaryInfo.Property](#)
- [SummaryInfo.PropertyCount](#)
- [SummaryInfo.Persist](#)
- [Installer.SummaryInformation](#)
- [Database.SummaryInformation](#)
- [Database.CreateTransformSummaryInfo](#)

The VBScript file [WiSumInf.vbs](#) is provided in the Windows SDK Components for Windows Installer Developers. This sample script can be used to manage the summary information stream of a Windows Installer package. For more information about [WiSumInf.vbs](#), see [Manage Summary Information](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Summary Information Stream Reference

The following list shows the summary information stream properties for Windows Installer:

- [Author Summary Property](#)
- [Character Count Summary Property](#)
- [Codepage Summary Property](#)
- [Comments Summary Property](#)
- [Create Time/Date Summary Property](#)
- [Creating Application Summary Property](#)
- [Keywords Summary Property](#)
- [Last Printed Summary Property](#)
- [Last Saved By Summary Property](#)
- [Last Saved Time/Date Summary Property](#)
- [Word Count Summary Property](#)
- [Page Count Summary Property](#)
- [Revision Number Summary Property](#)
- [Security Summary Property](#)
- [Subject Summary Property](#)
- [Template Summary Property](#)
- [Title Summary Property](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Author Summary Property

The **Author Summary** property conveys the manufacturer of the installation package, transform, or patch package.

- Set the **Author Summary** property in an installation package to the same value as the **Manufacturer** property.
- Set the **Author Summary** property in a transform to the name of the manufacturer of the transform.
- Set the **Author Summary** property in a patch package to the name of the manufacturer of the patch package.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
----------------	--

## See Also

[Summary Property Descriptions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Character Count Summary Property

The **Character Count Summary** property is only used in transforms. This part of the summary information stream is divided into two 16-bit words. The upper word contains the *transform validation flags*. The lower word contains the *transform error condition flags*.

This property should be Null in an installation package or patch package.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
----------------	--

## See Also

[Summary Property Descriptions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Codepage Summary Property

The **Codepage Summary** property is the numeric value of the ANSI code page used for any strings that are stored in the summary information. Note that this is not the same code page for strings in the installation database. The **Codepage Summary** property is used to translate the strings in the summary information into Unicode when calling the Unicode API functions. The **Codepage Summary** property must be set before any string properties are set in the summary information.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
----------------	--

## See Also

[Summary Property Descriptions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Comments Summary Property

The **Comments Summary** property conveys the general purpose of the installation package, transform, or patch package.

An author of an installation package, transform, or patch package should set the value of the **Comments Summary** property to one of the following values:

"This installer database contains the logic and data required to install *<product>*."

"This transform contains the logic and data required to install *<product>*."

"This patch contains the logic and data required to install *<product>*."

where *<product>* is the name of the product.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
----------------	--

## See Also

[Summary Property Descriptions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Create Time/Date Summary Property

The **Create Time/Date Summary** property conveys the time and date when an author created the installation package, transform, or patch package.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
----------------	--

## See Also

[Summary Property Descriptions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Creating Application Summary Property

The **Creating Application Summary** property conveys which application created the installer database. In general, the value for this summary property is the name of the software that is used to author this database.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
----------------	--

## See Also

[Summary Property Descriptions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Keywords Summary Property

The **Keywords Summary** property in installation databases or transforms contains a list of keywords. The keywords may be used by file browsers to do keyword searches for a file. This property contains a list of sources of the patch in a patch package.

It is up to the author of an installation database, transform, or patch package to provide the value of this property in the summary information. Authors should do the following to determine the correct value.

- In an installation package, set the value of this property to a list of keywords. The keyword should include "Installer" as well as product-specific keywords, and may be localized.
- In a transform, set the value of this property to a list of keywords. The keyword should include "Installer" as well as product-specific keywords, and may be localized.
- In a patch package, set the value of this property to a semicolon-delimited list of network or URL locations for the sources of the patch. When the patch is installed, the installer adds these to the source list for the patch package. If the cached patch becomes missing, the installer can search for a source in the original location, a location added to the source list by the **Keywords Summary** property, or a location added to the source list using the [MsiSourceListAddSource](#) or [MsiSourceListAddSourceEx](#) functions.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000

## See Also

[Summary Property Descriptions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Last Printed Summary Property

In an installation package, the **Last Printed Summary** property can be set to the date and time during an [administrative installation](#) to record when the administrative image was created. For non-administrative installations, this property is the same as the [Create Time/Date Summary](#) property.

In a transform, this property should be null.

In a patch package, this property should be null.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
----------------	--

## See Also

[Summary Property Descriptions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Last Saved By Summary Property

The installer sets the **Last Saved by Summary** Property to a value that depends on whether this is an installation package, transform, or patch package.

In an installation package, the installer sets this property to the value of the **LogonUser** property during an **administrative installation**. Developers of database editing tools may use this property during the development process to track the last person to modify the installation database. This property should be set to Null in a final shipping database. The installer never uses this property and a user never needs to modify it.

In a transform, this summary property contains the platform and language ID(s) that a database should have after it has been transformed. The property specifies to what the **Template Summary** should be set in the new database.

In a patch package, this summary property contains a semicolon-delimited list of transform substorage names in the order they are applied by this patch.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
----------------	--

## See Also

[Summary Property Descriptions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Last Saved Time/Date Summary Property

The **Last Saved Time/Date Summary** property conveys the last time when this installation package, transform, or patch package was modified.

Initially, an author should set the value of the **Last Saved Time/Date Summary** property to Null to indicate that no changes have yet been made to the package. An author should then update the **Last Saved Time/Date Summary** property to the current system time/date each time a modified installation database, transform, or patch package is saved.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
----------------	--

## See Also

[Summary Property Descriptions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Page Count Summary Property

The **Page Count Summary** property contains the minimum installer version required by the installation package. For a minimum of Windows Installer 2.0, this property must be set to the integer 200. For a minimum of Windows Installer 3.0, this property must be set to the integer 300. For a minimum of Windows Installer 3.1, this property must be set to 301. For a minimum of Windows Installer 4.5, this property must be set to 405. For a minimum of Windows Installer 5.0, this property must be set to 500.

For [64-bit Windows Installer Packages](#), this property must be set to the integer 200 or greater.

For a transform package, the **Page Count Summary** property contains the minimum installer version required to process the transform. Set to the greater of the two **Page Count Summary** property values that belong to the databases used to generate the transform.

For a patch package, the **Page Count Summary** property is set to Null.

This summary property is required.

This property can be used to author a package that can be installed only by the specified minimum or later version of the Windows Installer.

## Requirements

Version
Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000

## See Also

[Summary Property Descriptions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Revision Number Summary Property

For an installation package, the **Revision Number Summary** property contains the [package code](#) for the installer package. For more information about package codes, see [Package Codes](#).

For a transform, the **Revision Number Summary** property lists the product code GUIDs and version of the new and original products and the upgrade code GUID. The list is separated with semicolons as follows.

"Original-Product-Code Original-Product-Version ; New-Product Code New-Product-Version; Upgrade-Code"

For a patch package, the **Revision Number Summary** property specifies the GUID patch code for the patch. This can be followed by a list of patch code GUIDs for obsolete patches that are to be removed when this patch is applied. The patch codes are concatenated with no delimiters separating GUIDs in the list.

**Windows Installer 3.0:** If there is sequencing information present in the [MsiPatchSequence table](#), Windows Installer 3.0 uses the sequencing information in the table and ignores the list of obsolete patches included in the **Revision Number Summary** property. Windows Installer 3.0 can still use the obsolete patch information in the **Revision Number Summary** property if the package does not contain a MsiPatchSequence table.

**Windows Installer 2.0:** The MsiPatchSequence table is not supported. Windows Installer 2.0 can still use the obsolete patch information in the **Revision Number Summary** property if the package does not contain a MsiPatchSequence table.

This summary property is required.

## Requirements

	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows
--	--

**Version**

Server 2003, Windows XP, and Windows 2000

**See Also**

[Summary Property Descriptions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Security Summary Property

The **Security Summary** property conveys whether the package should be opened as read-only. The database editing tool should not modify a read-only enforced database and should issue a warning at attempts to modify a read-only recommended database. The following values of this property are applicable to Windows Installer files.

Value	Description
0	No restriction
2	Read-only recommended
4	Read-only enforced

This property should be set to read-only recommended (2) for an installation database and to read-only enforced (4) for a transform or patch.

## Requirements

Version
Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000

## See Also

[Summary Property Descriptions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Subject Summary Property

The value of the **Subject Summary** property conveys the name of the product, transform, or patch that is installed by the package.

It is up to the author of an installation database, transform, or patch package to provide the value of this property in the summary information. Authors should do the following to determine the correct value.

- Set the **Subject Summary** property in an installation package to the same value as the **ProductName** property.
- Set the **Subject Summary** property in a transform to the same value as the **Subject Summary** property in the original installation package.
- Set the **Subject Summary** property in the summary information of a patch package to a short description of the patch that includes the name of the product.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
----------------	--

## See Also

### [PATCHNEWSUMMARYSUBJECT](#)

[Summary Property Descriptions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Template Summary Property

For an installation package, the **Template Summary** property indicates the platform and language versions that are compatible with this installation database. The syntax of the **Template Summary** property information for an installation database is the following:

*[platform property];[language id][,language id][,...].*

The following examples are valid values for the **Template Summary** property:

- x64;1033
- Intel;1033
- Intel64;1033
- ;1033
- Intel ;1033,2046
- Intel64;1033,2046
- Intel;0

The **Template Summary** property of a transform indicates the platform and language versions compatible with the transform. In a transform file, only one language may be specified. The specified platform and language determine whether a transform can be applied to a particular database. The platform property and the language property can be left blank if no transform restriction relies on them to validate the transform.

The **Template Summary** property of a patch package is a semicolon-delimited list of the product codes that can accept the patch. If you use [Msimsp.exe](#) and [Patchwiz.dll](#) to generate the patch package, this list is obtained from the [TargetImages table](#) of the patch creation file.

This summary property is required.

## Remarks

If the current platform does not match one of the platforms specified in the **Template Summary** property then the installer does not process the

package.

If the platform specification is missing in the **Template Summary** property value, the installer assumes the Intel architecture.

If this is a 64-bit Windows Installer package being run on a Intel64 platform, enter Intel64 in the **Template Summary** property.

If this is a 64-bit Windows Installer package being run on a x64 platform, enter x64 in the **Template Summary** property.

A Windows Installer package cannot be marked as supporting both Intel64 and x64; for example, the **Template Summary** property value of Intel64,x64 is invalid.

A Windows Installer package cannot be marked as supporting both 32-bit and 64-bit platforms; for example, **Template Summary** property values such as Intel,x64 or Intel,Intel64 are invalid.

Entering 0 (zero) in the language ID field of the **Template Summary** property, or leaving this field empty, indicates that the package is language neutral.

Merge Modules are the only packages that may have multiple languages. Only one language can be specified in a source installer database. For more information, see [Multiple Language Merge Modules](#).

The Alpha platform is not supported by Windows Installer.

**Windows Installer:** The following syntax is not supported:

*[platform property][,platform property][,...][language id][,language id][,...].*

The following examples are not valid values for the **Template Summary** property:

- Alpha,Intel;1033
- Intel,Alpha;1033
- Alpha;1033
- Alpha;1033,2046

## Requirements

---

**Version**

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000

**See Also**

[Summary Property Descriptions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Title Summary Property

The **Title Summary** property briefly describes the type of the installer package. Phrases such as "Installation Database" or "Transform" or "Patch" may be used for this property.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
----------------	--

## See Also

[Summary Property Descriptions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Word Count Summary Property

In the summary information of an installation package, the **Word Count Summary** property indicates the type of source file image. If this property is not present, it defaults to zero (0).

This property is a bit field. New bits may be added in the future. At present the following bits are available.

Bit	Value	Description
0	0	Long file names.
0	1	Short file names.
1	0	Source is uncompressed.
1	2	Source is compressed.
2	0	Source is original media.
2	4	Source is a administrative image created by an administrative installation.
3	0	Elevated privileges can be required to install this package.
3	8	Elevated privileges are not required to install this package.
		Available starting with Windows Installer version 4.0 and Windows Vista or Windows Server 2008.

These are combined to give the **Word Count Summary** property one of the following values that indicate a type of source file image.

Value	Type
0	Original source using long file names. Matches tree in <a href="#">Directory Table</a> . Elevated privileges can be required to install this package.
1	Original source using short file names. Matches tree in <a href="#">Directory Table</a> . Elevated privileges can be required to install this package.
2	Compressed source files using long file names. Matches cabinets and files in the <a href="#">Media Table</a> . Elevated privileges can be required to install this package.

3	Compressed source files using short file names. Matches cabinets and files in the Media Table. Elevated privileges can be required to install this package.
4	Administrative image using long file names. Matches tree in Directory Table. Elevated privileges can be required to install this package.
5	Administrative image using short file names. Matches tree in Directory Table. Elevated privileges can be required to install this package.
8	Elevated privileges are not required to install this package. Use this value when <a href="#">Authoring Packages without the UAC Dialog Box</a> . Available starting with Windows Installer version 4.0 and Windows Vista or Windows Server 2008.

Note that if the package is marked as compressed (Bit 1 is set), the Windows Installer only installs files located at the root of the source. In this case, even files marked as uncompressed in the [File Table](#) must be located at the root to be installed. To specify a source image that has both a cabinet file (compressed files) and uncompressed files that match the tree in the [Directory Table](#), mark the package as uncompressed by leaving Bit 1 unset (value=0) in the **Word Count Summary** property and set `msidbFileAttributesCompressed` (value=16384) in the Attributes column of the [File Table](#) for each file in the cabinet.

In a transform, the **Word Count Summary** property should be Null.

In the summary information stream of a patch package, the **Word Count Summary** property indicates the minimum Windows Installer version that is required to install the patch.

Value	Meaning
1	The default value, which indicates that MSPATCH was used to create the patch.
2	Requires at minimum Windows Installer 1.2 for the patch to be applied. A patch with a Word Count of "2" fails immediately if used with a Windows Installer version earlier than 1.2.

3	Requires at minimum Windows Installer 2.0 for the patch to be applied. A patch with a Word Count of "3" fails immediately if used with a Windows Installer version earlier than 2.0.
4	Requires at minimum Windows Installer 3.0 for the patch to be applied. A patch with a Word Count of "4" fails if used with a Windows Installer version earlier than 3.0.
5	Requires at minimum Windows Installer 3.1 for the patch to be applied.

This summary property is REQUIRED.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
----------------	--

## See Also

[Summary Property Descriptions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patching and Upgrades

Because an installation package can contain the files that make up an application as well the information needed for their installation, Windows Installer can be used to update the application. The installer can update information in the following parts of the installation package:

- The .msi file.
- The files of the application.
- The Windows Installer registration information.

The type of update can be characterized by the changes the update makes to the application's product code, product version, and package code. The application's product version is stored in the **ProductVersion** property. The application's product code is stored in the **ProductCode** property. The application's **package code** is stored in the **Revision Number Summary** Property.

An update that changes the application into another product is required to change the **ProductCode** of the application. For more information about which updates require changing the **ProductCode** see [Changing the Product Code](#). The update can change the **ProductVersion** and leave the **ProductCode** unchanged if future versions of the application will need to differentiate between the updated and nonupdated versions of the current product. The Package Code uniquely identifies the installation package and should always be changed whenever update or upgrade changes any information in the installation package.

When deciding whether to change the product version, you should consider If future versions of the application will need to differentiate between the updated and nonupdated versions of the current product. To ensure differentiation in the future, a **minor upgrade** should be used instead of a **small update**.

- If an update changes the .msi file and application files, but does not change the **ProductCode** property or **ProductVersion** property, it is termed a small update.
- If the update changes the **ProductVersion**, but does not change the

**ProductCode**, it is termed a minor upgrade.

- If the update changes the installation into an entirely different product, the **ProductCode** must change and the update is termed a [major upgrade](#).

**Note** To ensure differentiation of versions of the current product in the future, a [minor upgrade](#) should be used instead of a [small update](#).

The following table summarizes the different types of updates.

Type of update	Productcode	ProductVersion	Description
<a href="#">Small Update</a>	No change	No change	An update to one or two files that is too small to warrant changing the <b>ProductVersion</b> . The package code in the <a href="#">Revision Number Summary</a> Property does change. Can be shipped as a full installation package or as a <a href="#">patch package</a> .
<a href="#">Minor Upgrade</a>	No change	Changed	A small update making changes significant enough to warrant changing the <b>ProductVersion</b> property. Can be shipped as a full installation package or as a <a href="#">patch package</a> .
<a href="#">Major Upgrades</a>	Changed	Changed	A comprehensive update of the product warranting a change in the <b>ProductCode</b> property. Shipped as a <a href="#">patch package</a> or as a full product installation package.

**Note** The Windows Installer can install an application, or an update, for all users of a computer (per-machine context) or for a particular user (per-user context) depending on the access privileges of the user, the value of the **ALLUSERS** property, and the version of the operating

system. Application developers should consider in which context updates will be installed. If the contexts of the application and update are different, the application may not be updated as expected.

Users can update to an application by reinstalling a Windows Installer package for the application. Note that Minor Upgrades can be applied in the same way as Small Updates. For more information about updating an application by reinstalling the application, see these sections:

- [Applying Small Updates by Reinstalling the Product](#)
- [Applying Major Upgrades by Installing the Product](#)

An update to an application can be provided to users as a Windows Installer patch package. A patch can contain an entire file or only the file bits necessary to update part of a file. This means that the user can download an upgrade patch that is much smaller than the entire product and that preserves user customizations through the upgrade. Note that Minor Upgrades can be applied in the same way as Small Updates. For more information about updating an application using a patch, see these sections:

- [Patching](#)
- [Creating a Small Update Patch](#)
- [Applying Small Updates by Patching the Local Installation of the Product](#)
- [Applying Small Updates by Patching an Administrative Image](#)
- [Applying Major Upgrades by Patching the Local Installation of the Product](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Small Updates

A small update makes changes to one or more application files that are too minor to warrant changing the product code. A small update is also commonly referred to as a quick fix engineering (QFE) update. A small update does not permit reorganization of the feature-component tree.

A typical small update changes only one or two files or a registry key. Because a small update changes the information in the .msi file, the installation package code must be changed. The package code is stored in the [Revision Number Summary](#) property of the [Summary Information Stream](#).

The product code is never changed with a small update, so all of the changes introduced by a small update have to be consistent with the guidelines described in [Changing the Product Code](#). An update requires a [major upgrade](#) to change the **ProductCode**. If it is necessary to differentiate between products without changing the product code, use a [minor upgrade](#).

For information on how to apply a small update patch package to a Windows Installer package, see [Creating a Small Update Patch](#), [Applying Small Updates by Patching the Local Installation of the Product](#), [Applying Small Updates by Reinstalling the Product](#), and [Applying Small Updates by Patching an Administrative Image](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Creating a Small Update Patch

When creating a patch for [small updates](#), authors should adhere to the following guidelines:

- Small update patches must be designed for a single, target installation.
- Small update patches should use the earliest version as the target installation.
- A small update patch should replace and make obsolete any earlier small update patches.

The following scenario illustrates when a small update patch is best.

Your company ships version 1.0 of Myproduct.msi. Shortly thereafter, you ship a small update patch for Myproduct.msi called QFE1. This does not change the **ProductCode** property or the **ProductVersion** property.

Later, you author a second small update patch for Myproduct.msi called QFE2. This second patch must target Myproduct.msi version 1.0. This second patch must not target both Myproduct.msi version 1.0 and Myproduct.msi version 1.0 + QFE1. When QFE2 is applied it should remove QFE1.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Applying Small Updates by Patching the Local Installation of the Product

A small update can be applied to an application by patching the local installation of the application.

## ► To apply a small update patch to a local installation of the product

1. Launch the installation of the patch from the command line or by using an executable. To launch from the command line, use **msiexec /p patch.msp REINSTALL=[*Feature list*] REINSTALLMODE=omus**. To launch from an executable, call **MsiApplyPatch** or the **ApplyPatch Method** and provide the same command line arguments.
2. When patching a client installation, the installer ignores the installation source and proceeds to patch the files that are already installed on the user's computer.
3. The installer changes any patched components marked as run-from-source to run-locally. Users are unable to run these components from the source as long as the patch remains on the computer.
4. The installer adds any transforms used to update the .msi file or adds patch-specific information to the user's profile.
5. The installer caches the .msi file on the user's computer so that it can perform installation-on-demand, reinstall, and repair of the application. After a patch is applied to a standalone installation, the installer references two or more source lists to external files: one for the original source and one for each patch that has been applied.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Applying Small Updates by Reinstalling the Product

A small update can be applied to an application by completely or partially reinstalling the application from the command line or from a program.

▶ **To propagate the small update to current users (this is a complete reinstall) from the command line**

1. From the command line use either: `msiexec /fvomus [path to updated .msi file]` or `msiexec /I [path to updated .msi file] REINSTALL=ALL REINSTALLMODE=vomus`.
2. The updated .msi file is cached on the user's computer. Note that it is not possible for the user to reinstall the product using Add/Remove Programs because the updated .msi file is not yet on the user's computer.

▶ **To propagate a small update to current users (this is a complete reinstall) from a program**

1. From a program, call `MsiReinstallProduct` and specify `REINSTALLMODE_PACKAGE`, `REINSTALLMODE_FILEOLDERVERSION`, `REINSTALLMODE_MACHINEDATA`, `REINSTALLMODE_USERDATA`, and `REINSTALLMODE_SHORTCUT`
2. The updated .msi file is cached on the user's computer.

The following method launches a reinstallation of only those features or components that are affected by the small update.

▶ **To propagate a small update to current users (this is a partial reinstall)**

1. Obtain a list of the names of features and components that are

affected by the small update.

2. From the command prompt use: **msiexec /I [path to updated .msi file] REINSTALL=[Feature list] REINSTALLMODE=vomus.**
3. The updated .msi file is cached on the user's computer.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Applying Small Updates by Patching an Administrative Image

An [administrative installation](#) creates a source image of an application or product on a network. Users in a workgroup who have access to this administrative image can install the product from this source. Updating the application for this workgroup is done in two steps:

- First, the small update patch can be applied to the administrative image.
- Second, the changes in the small update need to be propagated to the users.

## ▶ To apply a small update patch to an administrative image

1. Obtain the small update in the form of a [patch package](#). For example, the small update named Patch.msp.
2. Obtain the path to the administrative image.
3. From the command line use:  
**msiexec /a [path to administrative image .msi file] /p patch.msp**
4. This updates the application files and the .msi file of the administrative image. For a list of the options that can be used with Msiexec.exe, see [Command Line Options](#). Windows Installer automatically determines whether the administrative image is using short file names and sets the **SHORTFILENAMES** property.
5. The resulting administrative image is the same as that produced by an administrative installation using a full product CD-ROM that includes the update. When new users install the application from this source they receive the updated application.

## ▶ To propagate the small update to the workgroup

- Members of the workgroup obtain the changes by reinstalling the

application from the administrative image using the procedure described in [Applying Small Updates by Reinstalling the Product](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Minor Upgrades

A minor upgrade is an update that makes changes to many resources. None of the changes can require changing the **ProductCode**. An update requires a [major upgrade](#) to change the **ProductCode**. A minor upgrade can be used to add new features and components but cannot reorganize the feature-component tree. Minor upgrades provide product differentiation without actually defining a different product. A typical minor upgrade includes all fixes in previous small updates combined into a patch. A minor upgrade is also commonly referred to as a service pack (SP) update. For more information about which updates do not require changing the **ProductCode** see [Changing the Product Code](#).

A minor upgrade changes the **ProductVersion** property. Changing the product version of the application means that the different updates have an order. For example, if a patch existed to update v 9.0 to v 9.1, and another patch existed to patch v 9.1 to v 9.2, the installer can enforce the correct order by checking the product version before applying the patch. This also prevents the v 9.1 to v 9.2 patch from being applied to v 9.0. For patches, this ordering is enforced through the product version-validation bits set in the transforms included in the [patch package](#).

A minor upgrade and a small update differ in that a minor upgrade changes the package code and product version. See [Small Updates](#) for guidelines on the kinds of updates that can be handled by a small update or minor upgrade. Minor upgrades are shipped as a full product installation package or as a patch package. However, a minor upgrade cannot use a different volume label for the new version.

For information on how to apply a minor upgrade, see the following topics:

- [Applying Small Updates by Patching the Local Installation of the Product](#)
- [Applying Small Updates by Reinstalling the Product](#)
- [Applying Small Updates by Patching an Administrative Image](#)

[Send comments about this topic to Microsoft](#)

© 2009 Microsoft Corporation. All rights reserved.

# Major Upgrades

A major upgrade is a comprehensive update of a product that needs a change of the **ProductCode** Property.

A typical major upgrade removes a previous version of an application and installs a new version. A major upgrade can reorganize the feature component tree. For more information, see **ProductCode** and [Changing the Product Code](#).

During a major upgrade using Windows Installer, the installer searches the user's computer for applications that are related to the pending upgrade, and when it detects one, it retrieves the version of the installed application from the system registry. The installer then uses information in the upgrade database to determine whether to upgrade the installed application.

To enable the installer upgrade capabilities, each package should have an **UpgradeCode** Property and an [Upgrade Table](#). Each stand-alone product or product suite should have its own **UpgradeCode**. For more information about using the **UpgradeCode** see the section [Using an UpgradeCode](#). Each record in the Upgrade table gives a combination of the upgrade code, product version, and language information used to identify a set of products affected by the upgrade. When the [FindRelatedProducts Action](#) detects that an affected product is installed on the system, it appends the product code to a property in the ActionProperty column of the Upgrade table. The [RemoveExistingProducts action](#) and the [MigrateFeatureStates Action](#) remove or migrate the products listed in the ActionProperty list. Package authors can also follow the procedure described in the topic: [Preparing an Application for Future Major Upgrades](#).

Windows Installer upgrade packages can be authored such that major upgrades will not install if the user already has a newer version of the application installed. For more information about how to author a package that will not install over a newer version, see [Preventing an Old Package from Installing Over a Newer Version](#)

**Note** Windows Installer uses only the first three fields of the product version. See **ProductVersion** Property for descriptions of these fields. If you include a fourth field in your product version, the installer ignores the

fourth field.

The recommended method of applying a major upgrade by installing the full package for the updated product. For information about how to apply a major upgrade by installing the product, see [Applying Major Upgrades by Installing the Product](#).

A major upgrade applied as a [Patch Package](#) for the product cannot be sequenced with other updates and is not an [uninstallable patch](#). For information about how to apply a major upgrade patch package to a Windows Installer package see [Applying Major Upgrades by Patching the Local Installation of the Product](#). The application of a major upgrade using a patch package is not recommended, instead apply major upgrades by installing the full product.

**Note** If an application is installed in the per-user [installation context](#), any major upgrade to the application must also be performed using the per-user context. If an application is installed in the per-machine installation context, any major upgrade to the application must also be performed using the per-machine context. The Windows Installer will not install major upgrades across installation context.

You can condition custom actions that are sequenced after [InstallValidate](#) to handle major upgrades by using the **UPGRADINGPRODUCTCODE** property:

- If you want a custom action to run during an uninstallation of the product, but not during the removal of the product by a major upgrade, use this condition.  
**REMOVE="ALL" AND NOT UPGRADINGPRODUCTCODE**
- If you want a custom action to run only during a major upgrade, use this condition.  
**UPGRADINGPRODUCTCODE**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Changing the Product Code

The product code is a GUID that is the principal identification of an application or product. See [Product Codes](#).

An update that meets the following guidelines generally does not require a change of the product code and can be handled as a [small update](#), or if the version is to change, as a [minor upgrade](#):

- The update can enlarge or reduce the feature-component tree but it must not reorganize the existing hierarchy of features and components described by the [Feature](#) and [FeatureComponents](#) tables. It can add a new feature to the existing feature-component tree. If it removes a parent feature, it must also remove all the child features of the removed feature.
- The update can add a new component to a new or an existing feature.
- The update must not change the component code of any component. Consequently, a small update or minor upgrade must never change the name of a component's key file because this would require changing the component code.
- The update must not change the name of the .msi file of the installation package. Instead, because it modifies the package, it should change the package code. Note that this means that the update can change the tables, custom actions, and dialogs in the .msi file without changing the file's name.
- The update can add, remove, or modify the files, registry keys, or shortcuts of components that are not shared by two or more features. If the update modifies a versioned file, that file's version must be incremented in the [File table](#). If the update removes resources, it should also update the [RemoveFile](#) and [RemoveRegistry](#) tables to remove any unused files, registry keys, or shortcuts that have already been installed.

- The update of a component that is shared by two or more features must be backward compatible with all applications and features that use the component. The update can modify the resource of a shared component, such as files, registry entries, and shortcuts, as long as the changes are backward compatible. It is not recommended that the update add or remove files, registry entries, or shortcuts from a shared component.
- A small update is shipped as a Windows Installer [patch package](#). (A full product CD-ROM is usually not provided with a small update.)

The product code must be changed if any of the following are true for the update:

- Coexisting installations of both original and updated products on the same system must be possible.
- The name of the .msi file has been changed.
- The component code of an existing component has changed.
- A component is removed from an existing feature.
- An existing feature has been made into a child of an existing feature.
- An existing child feature has been removed from its parent feature.
- A component may be added to an existing feature without requiring a product code change.

Note that adding a new child feature, consisting entirely of new components, to an existing feature does not require changing the product code.

New child features can be authored by including `msidbFeatureAttributesFollowParent` and `msidbFeatureAttributesUIDisallowAbsent` in the `Attributes` field of the [Feature table](#). If the minor upgrade only adds new child features, then `REINSTALL=ALL` is sufficient to force the installation of the new child features. For more information, see [Controlling Feature Selection States](#).

A new child feature may be hidden from the user. To synchronize the installation state of a new child feature with its parent feature, set the

msidbFeatureAttributesFollowParent and  
msidbFeatureAttributesUIDisallowAbsent bits for the child feature.

## **See Also**

[About Properties](#)  
[Using Properties](#)  
[Property Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Preparing an Application for Future Major Upgrades

Authors of installation packages should include upgrading information in their .msi files to ensure that their installation package can take advantage of the full upgrading functionality available with the Microsoft Windows Installer.

Every application, or suite of applications, should be assigned an **UpgradeCode** Property, **ProductVersion** Property, and **ProductLanguage** Property. The **UpgradeCode** property indicates a family of related applications consisting of different versions and different language versions of the same product. For more information about using the **UpgradeCode** property, see [Using an UpgradeCode](#).

## ► Preparing an application for future major upgrades

1. Determine a new package code value for the application. For more information about package codes, see [Package Codes](#). Enter the new package code into the **Revision Number Summary** Property of the [Summary Information Stream](#).
2. Determine a new **ProductCode** property for the application. See [Changing the Product Code](#) for more information. Enter **ProductCode** and its value into the [Property table](#).
3. Determine the application's version and the **ProductVersion** property. The **ProductVersion** should increase with each new version of the application. Note that the installer uses only the first three fields of the product version. If you include a fourth field in your product version, the installer ignores the fourth field. Enter **ProductVersion** and its value into the [Property table](#).
4. Determine the language of the package and the **ProductLanguage** property. The value of this property must be a numeric language identifier (LANGID). Enter **ProductLanguage** and its value into the [Property table](#). Note that the

[FindRelatedProducts](#) action uses the language returned by [MsiGetProductInfo](#). For FindRelatedProducts to work correctly, the package author must be sure that the **ProductLanguage** property is set in the Property table to a language that is also listed in the **Template Summary** property.

5. If you are authoring an installation package for the first version of your product, use a new **UpgradeCode**. If your package is intended for a newer version of an existing product, or is the same version as an existing product in a different language, use the same **UpgradeCode** as the existing product. No two products with the same **ProductVersion** and the same **ProductLanguage** can have the same **UpgradeCode**, unless one is a [small update](#) of the other.
6. The **UpgradeCode** has the format of a **GUID**. Enter the **UpgradeCode** GUID into the Property table.

For more information, see [Preventing an Old Package from Installing Over a Newer Version](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Preventing an Old Package from Installing Over a Newer Version

Windows Installer upgrade packages can be authored to have major upgrades not install if a user already has a newer version installed. The procedure in this topic can only prevent downgrades that might be caused by running a major upgrade package. This procedure relies on the [FindRelatedProducts Action](#), which only runs during a first-time installation and does not run in maintenance mode (reinstallation). Because minor upgrades are performed using reinstallation, this procedure cannot be used to determine whether a minor upgrade package is attempting to downgrade an application. For more information, see [Preparing an Application for Future Major Upgrades](#).

## ► To prevent an old package from installing over a newer version

1. Enter the [UpgradeCode](#) Property for the group of related products that may be eligible to receive this upgrade into the UpgradeCode column of the [Upgrade Table](#).
2. Enter the msidbUpgradeAttributesOnlyDetect bit flag in the Attributes column of the Upgrade Table.
3. Enter the version of the upgrade provided by this package into the VersionMin column of the Upgrade Table. Leave the VersionMax column blank.
4. Enter the name of the property that is to be set by the FindRelatedProducts Action into the ActionProperty column of the [Upgrade Table](#).
5. Add the [SecureCustomProperties](#) property and the property named in the ActionProperty column of the [Upgrade Table](#) to the [Property Table](#).
6. Add a [Custom Action Type 19](#) after the FindRelatedProducts action in the [InstallExecuteSequence Table](#). Include a record in the [CustomAction Table](#) for this action and enter the text to be

displayed in the Target column. The type 19 custom action is built into the installer, so there is no code to write.

7. Enter the name of the ActionProperty into the Condition column of the record in InstallExecuteSequence Table that contains the Custom Action Type 19. This conditions the custom action to only be executed when the Upgrade Table detects that a newer version is already installed.

For example, a Windows Installer package that upgrades a group of related products to version 3.0 may include the following records in its Upgrade, CustomAction, InstallExecuteSequence, and Property tables. All the related products in the group have the same UpgradeCode, but the installer does not install this upgrade package if a version later than 3.0 is already installed on the computer. In this case, the Installer presents an error message and the installation fails. The version 3.0 upgrade package installs over versions 1.0 and 2.0.

### Upgrade Table

UpgradeCode	VersionMin	VersionMax	Language	Attributes
{E7BE6D45-49FF-4701-A17E-BDCC98CE180D}	3.0			msidbUpgr
{E7BE6D45-49FF-4701-A17E-BDCC98CE180D}	1.0	3.0		msidbUpgr

### CustomAction Table

Action	Type	Source	Target
CA1	19		A higher upgrade is already installed.

## InstallExecuteSequence Table

Action	Condition	Sequence
FindRelatedProducts		200
CA1	NEWPRODUCTFOUND	201

## Property Table

Property	Value
<b>SecureCustomProperties</b>	NEWPRODUCTFOUND;UPGRADEFOU

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using an UpgradeCode

The **UpgradeCode** is primarily used for supporting major upgrades, although small and minor upgrade patches may use the **UpgradeCode** for product validation. During major upgrades, the **FindRelatedProducts**, **MigrateFeatureStates**, and **RemoveExistingProducts** actions detect, migrate, and remove previous versions of the product. The **FindRelatedProducts** action searches for products using criteria based upon the **UpgradeCode**, **ProductLanguage**, and **ProductVersion**. These criteria are specified in the **Upgrade** table.

Given the criteria used by the **FindRelatedProducts** action, the **UpgradeCode** can be the same for different languages and versions of a single product. This is because the **Upgrade** table allows for differentiating between products along version and language lines.

Across different versions of the same product, you may never need to change the **UpgradeCode**. Each stand-alone product should have its own **UpgradeCode**. A product suite should also have its own **UpgradeCode**. Doing so will allow the suite to upgrade previous versions of the suite or stand-alone products by using multiple rows in the **Upgrade** table.

The following two scenarios illustrate the use of the **UpgradeCode**.

- Product A and Product B were shipped with the same **ProductLanguage**, **ProductVersion**, and **UpgradeCode**. Product A and Product B have different **ProductCodes**. Because the products were assigned the same **UpgradeCode**, the **Upgrade** table cannot be authored to differentiate the older version of Product A from the older version of Product B. In this case, you will be unable to have an upgrade installation of Product A that ignores Product B. Because these were different products, they should have each been assigned a different **UpgradeCode**.
- The English and French versions of Product A were shipped with the same **ProductVersion** and **UpgradeCode**. Both the English and French versions of Product A have different **ProductLanguages** and

**ProductCodes.** Even though both the English and French language versions share the same **UpgradeCode**, it is possible to author the Upgrade table such that only the older English language version will be detected and upgraded and the older French version ignored. Different language versions of a product can use the same **UpgradeCode**.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Applying Major Upgrades by Patching the Local Installation of the Product

A major upgrade can be applied to an application by patching the local installation of the application from the command line or by using an executable.

**Note** Providing a major upgrade as a patch package is not recommended because a major upgrade patch package cannot be sequenced with other updates and because the patch is not an [uninstallable patch](#). The [Msimsp.exe](#) utility cannot be used to generate a patch package that applies a major upgrade. Instead, apply major upgrades as described in [Applying Major Upgrades by Installing the Product](#).

## ► To apply a major upgrade patch to a local installation of the product

1. Launch the installation of the patch from the command line or by using an executable. To launch from the command line, use `msiexec /p patch.msp`. To launch from an executable, call [MsiApplyPatch](#) or the [ApplyPatch Method](#) and provide the same command line arguments.
2. When patching a client installation, the installer ignores the installation source and proceeds to patch the files that are already installed on the user's computer.
3. The installer changes any patched components marked as run-from-source to run-locally. Users are unable to run these components from the source as long as the patch remains on the computer.
4. The installer adds any transforms used to update the .msi file or adds patch-specific information to the user's profile.
5. The installer caches the .msi file on the user's computer so that it

can perform installation-on-demand, reinstall, and repair of the application. After a patch is applied to a stand alone installation, the installer references two or more source lists to external files: one for the original source and one for each patch that has been applied.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Applying Major Upgrades by Installing the Product

A major upgrade can be applied by installing the new installation package for the upgraded product. Because major upgrades get a different product code than the original product, installing the upgrade must be treated as an installation of a new product. The upgrade can simply be installed like another product. You can have the new installation package handle the removal of the old product by including the [Upgrade table](#) and the [FindRelatedProducts action](#) and [RemoveExistingProducts action](#).

▶ **To propagate a major upgrade to current users from the command line**

- From the command line, use: **msiexec /i** *[path to updated msi file]*

▶ **To propagate a major upgrade to current users from a program**

- From a program, call **MsiInstallProduct** and specify the path to the updated msi file.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patching

An application that has been installed using the Microsoft Windows Installer can be upgraded by reinstalling an updated installation package (.msi file), or by applying a Windows Installer patch (an .msp file) to the application.

A Windows Installer patch (.msp file) is a self-contained package that contains the updates to the application and describes which versions of the application can receive the patch. Patches contain at a minimum, two database transforms and can contain patch files that are stored in the cabinet file stream of the patch package. For more information about the parts of a Windows Installer patch package, see [Patch Packages](#).

Servicing applications by delivering a Windows Installer patch, rather than a complete installation package for the updated product can have advantages. A patch can contain an entire file or only the file bits necessary to update part of the file. This can enable the user to download an upgrade patch that is much smaller than the installation package for the entire product. An update using a patch can preserve a user customization of the application through the upgrade.

## **Windows Installer 4.5 and later:**

Beginning with Windows Installer 4.5, developers can mark components in a patch with the `msidbComponentAttributesUninstallOnSupersedence` value in the [Component table](#). If a subsequent patch is installed, marked with the `msidbPatchSequenceSupersedeEarlier` value in its [MsiPatchSequence table](#) to supersede the first patch, Windows Installer 4.5 and later can unregister and uninstall components marked `msidbComponentAttributesUninstallOnSupersedence` to prevent leaving behind unused components on the computer. If the component is not marked with this bit, installation of the superseding patch can leave an unused component on the computer. Setting the [MSIUNINSTALLSUPERSEDEDCOMPONENTS](#) property has the same effect as setting this bit for all components.

## **Windows Installer 3.0 and later:**

Developers who use Windows Installer 3.0, and author patch packages that have the MsiPatchSequence table can create patch packages that do the following:

- Use the product baseline cached by the installer to more easily service applications with smaller delta patches. For more information about using the product baseline, see [Reducing Patch Size](#).
- Skip actions associated with specific tables that are unmodified by the patch. This can significantly reduce the time required to install the patch. For more information about which tables can be skipped, see [Patch Optimization](#).
- Create and install patches that can be uninstalled singly, and in any order, without having to uninstall and reinstall the entire application and other patches. For more information about uninstalling patches, see [Removing Patches](#).
- Apply patches in a constant order regardless of the order that the patches are provided to the system. For more information about how the Windows Installer determines the sequence used to apply patches, see [Sequencing Patches](#).
- Apply patches to an application that has been installed in a per-user-managed context. For more information, see [Patching Per-User Managed Applications](#).

### **Windows Installer 2.0:**

The MsiPatchSequence table is not supported. Beginning with Windows Installer 3.0, patch packages can contain information that describes the patching sequence for the patch relative to other updates and additional descriptive information.

The recommended method for creating a patch package is to use patch creation tools such as [Msimsp.exe](#) and [Patchwiz.dll](#). Developers can generate a patch creation file as described in the section: [Creating a Patch Package](#). The creation of a small update patch is described in the

section: [A Small Update Patching Example](#).

Microsoft Windows Installer accepts a Uniform Resource Locator (URL) as a valid source for a patch. For more information about how to install a patch located on a Web server, see [Downloading and Installing a Patch From the Internet](#).

A single Windows Installer patch (.msp file) can be applied to the installation package when installing an application for the first time. For more information, see [Patching Initial Installations](#).

It is not possible to eliminate all circumstances when the application of a patch may require access to the original installation source. However, to minimize the possibility that your patch will require access to the original source, adhere to the points listed in the following section: [Preventing a Patch from Requiring Access to the Original Installation Source](#).

To minimize the possibility that your patch is not broken by a subsequent customization transform, typically the patch is installed first, followed by the customization. Installing customization transforms first, and then the patch, may break the customization. For more information about patching customized applications, see [Patching Customized Applications](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Patches

The following topics identify how you can use patches:

- [Downloading and Installing a Patch From the Internet](#)
- [User Account Control \(UAC\) Patching](#)
- [Patching Per-User Managed Applications](#)
- [Patching Initial Installations](#)
- [Patching Customized Applications](#)
- [Preventing a Patch from Requiring Access to the Original Installation Source](#)
- [Removing Patches](#)
- [Creating a Patch Package](#)
- [Installing Multiple Patches](#)
- [Extracting Patch Information as XML](#)
- [Listing the Files that can be Updated](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Downloading and Installing a Patch From the Internet

Microsoft Windows Installer accepts a Uniform Resource Locator (URL) as a valid source for a [patch](#). To install a patch located on a Web server at <http://MyWeb/MyPatch.msp>, use the following command line:

```
msiexec /p http://MyWeb/MyPatch.msp
```

To avoid unexpected results, do not launch a patch by clicking the link on the Web page showing the patch file's URL. You can also install a patch by using a script like the following:

```
<SCRIPT LANGUAGE="VBScript">
<!--
Dim Installer
On Error Resume Next
set Installer=CreateObject("WindowsInstaller.Installer")
Installer.ApplyPatch "http://server/share/patch.msp", "", C
set Installer=Nothing
-->
</SCRIPT>
```

Note that because the **Installer** object is not marked as [SafeForScripting](#) on the user's computer, users need to adjust their browser security settings for the example to work correctly.

For more information, see [Guidelines for Authoring Secure Installations](#) and [Digital Signatures and Windows Installer](#).

## See Also

[Patch Packages](#)

[Creating a Patch Package](#)

[Patching Customized Applications](#)

[Downloading an Installation From the Internet](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patching Initial Installations

A Windows Installer Patch (MSP) can be applied when installing an application for the first time by using the **PATCH** property.

To apply a patch the first time the application is installed, the **PATCH** property must be set on the command line. Specify the full path to the patch on the command line as the "PATCH={*path to patch*}" property-value pair.

Note that specifying the **PATCH** property on the command line overrides the patch applicability checks performed when using **MsiApplyPatch** or the /p [Command Line Option](#).

If a patch is applied using **MsiApplyPatch** or the /p Command Line Option, the installer compares the applications currently installed on the computer to the list of product codes eligible to receive the patch in the [Template Summary](#) property.

When you set the **PATCH** property on the command line to install on first installation, the applications eligible to receive the patch is determined by validation conditions on the transforms embedded in the patch package. The recommended method for generating a patch package is to use a patch creation tool such as [Msimsp.exe](#) and [PATCHWIZ.DLL](#). The validation conditions on transforms in the patch originate from the ProductValidateFlags column in the [TargetImages](#) table of the Patch Creation Properties (.pcp) file.

The patch can be applied the first time the application is installed by a command line, another application, or script.

The following shows first-time patching from the command line.

```
msiexec /I package.msi PATCH="c:\directory\patch.msp"
```

The following shows first-time patching from another application.

```
UINT uiStat = MsiInstallProduct(_T("package.msi"), _T("PATCH"))
```

The following shows first-time patching from script.

```
Dim Installer as Object  
Set Installer = CreateObject("WindowsInstaller.Installer")
```

```
Installer.InstallProduct "package.msi", "PATCH=c:\directory
```

### **Windows Installer 3.0 and later:**

Beginning with Windows Installer version 3.0, multiple patches can be applied when installing an application for the first time. Set the **PATCH** property to a semicolon delimited list of the patches' full paths. The following shows first-time patching of multiple patches from the command line.

```
msiexec /I package.msi
```

```
PATCH="c:\directory\patch.msp;c:\directory\patch2.msp;c:\directory\pat
```

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patching Customized Applications

When installing a patch and one or more customization transforms to an application, the patch is typically installed first, followed by the customization transforms. By design, the patch is not broken by the subsequent installation of the customization. However, installing the transforms first, and then the patch, may break the customization.

For example, a break in the customization could occur when a patch is used to update a product from version 1 to version 2 and a customization transform that works for version 1 does not work for version 2. In this case, the version update patch cannot be applied to a customized product without first uninstalling and then reinstalling the original product.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Preventing a Patch from Requiring Access to the Original Installation Source

It is not possible to eliminate all circumstances under which the application of a patch may require access to the original installation source.

Adhere to following points to minimize the possibility that your patch will require access to the original source:

- Use whole-file only patches. This eliminates the need to create binary patches for all previously released versions of the file. Note that whole file patches are generally larger in size than binary patches. You can easily set a patch to be a whole file patch by authoring the `IncludeWholeFilesOnly` property with a value of 1 (one) in the Patch Creation Properties (PCP) file.
- Ensure that none of your custom actions access the original source location.
- Ensure that the `ResolveSource` action is conditionalized so that it only runs when needed, or alternatively is not present at all.
- Populate the [MsiFileHash Table](#) for all unversioned files. The Windows Installer SDK tool, `Msifiler.exe`, can easily do this for you.
- Ensure that all files have the correct version and language information. The Windows Installer SDK tool, `Msifiler.exe`, can easily do this for you.

## Source Requirements When Patching

Access to the original installation sources may be required to apply the patch in the following cases:

- The patch applies to a feature that is currently run from source. In

this case, the feature is transitioned from the run-from-source state to the local state.

- The patch applies to a component that has a missing or corrupted file.
- The patch applies to a file in a component that also contains unversioned files with no MsiFileHash entries. A populated MsiFileHash Table is required to prevent unnecessary recopying of unversioned files from the source location.
- The patch was applied with a REINSTALLMODE of amus or emus. This option is dangerous in that it performs file copy operations regardless of file version. This can lead to down-leveling of files and almost always requires the source. The recommended REINSTALLMODE value is omus.
- The cached package for the product is missing. The cached package is needed for application of a patch. The cached package is stored in the %windir%\Installer folder.
- The package is authored to make a call to the [ResolveSource Action](#). This action should generally be avoided or conditionalized appropriately, because its execution always results in an access to the source.
- The package has a custom action that attempts to access the source in some manner. The most common example is a type 23 concurrent installation custom action.  
**Note** Concurrent installations are not recommended for the installation of applications intended for release to the public. For information about concurrent installations please see [Concurrent Installations](#).
- The patch package consists of binary patches that do not apply to the current version of the file on the computer.

Consider the following example where Windows Installer requires access

to the original source when applying a patch:

1. Install RTM version of the product Example.
2. Apply patch Qfe1.msp to the computer. This patches version 1.0 of Example.dll to version 1.1.
3. A new patch, Qfe2.msp is provided, which updates Example.dll to version 1.2 and obsoletes Qfe1.msp. However, the patch was only created to target version 1.0 of Example.dll because it was generated using the RTM version of the product. Example.dll version 1.2 includes the fix contained in Example.dll version 1.1, but the .msp file was generated between the RTM and QFE2 images. So, when Qfe2.msp is applied to the computer, Windows Installer needs to access the original source. The binary patch for Example.dll cannot apply to version 1.1; it can only apply to version 1.0. This results in the Installer recopying version 1.0 of Example.dll from the original source location so that the patch can be applied successfully.

## Source Requirements When Removing a Patch

Access to the original installation sources may be required to remove a patch if the Windows Installer has not stored baseline information about the patch. Beginning with Windows Installer 3.0, the installer selectively saves baseline information about files when they are updated. For more information about the baseline cache, see [Reducing Patch Size](#) .

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Extracting Patch Information as XML

The patch sequencing and applicability information that is returned by the [MsiExtractPatchXMLData](#) function or the [ExtractPatchXMLData](#) method is in the format of an XML blob that contains the elements and attributes that are identified in this topic. The XML blob can be provided to [MsiDeterminePatchSequence](#) and [MsiDetermineApplicablePatches](#) instead of the full patch file.

- The MsiPatch element is the top element of the XML blob, and contains information about the patch.  
The SchemaVersion attribute specifies the version of the schema definition. The schema is specified by MSIPatchApplicability.xsd and the current schema version is 1.0.0.0. The value of the PatchGUID attribute is the GUID patch code for the patch package obtained from the [Revision Number Summary](#) Property in the [Summary Information Stream](#) of the patch. The MinMsiVersion is the minimum version of the Windows Installer required to install the patch obtained from the [Word Count Summary](#) Property.
- The TargetProduct element is a container element for information about an application that a patch targets.  
There can be multiple TargetProduct elements if the patch can be applied to multiple applications. The information in the TargetProduct element is extracted from transforms that are embedded within the patch.
- The TargetProductCode element contains the value of the [ProductCode](#) property of the target application before the patch has been applied.  
There can be multiple TargetProductCode elements if the patch can be applied to multiple applications.
- The UpdatedProductCode element contains the product code GUID of the target application after the patch is applied.

This element is only present if the patch changes the value of the **ProductCode** property. A patch that changes the **ProductCode** is referred to as a [Major Upgrade](#).

- The TargetVersion element contains the **ProductVersion** property of the target application before the patch has been applied.
- The UpdateVersion element contains the value of the **ProductVersion** property of the target application after the patch is applied.

This element is only present if the patch changes the value of the **ProductVersion** property. The XML blob for a patch that implements a [Small Update](#), also referred to as a QFE, will not include this element. The XML blob for a patch that implements a minor upgrade, also referred to as a service pack, will include this element.

- The TargetLanguage element contains the value of the **ProductLanguage** property of the target application before the patch has been applied.
- The UpdatedLanguages element contains the value of the **ProductLanguage** property after the patch has been applied.
- The UpgradeCode element contains the value of the **UpgradeCode** property of the target application.
- The ObsoletedPatch element contains the patch codes (GUIDs) of the patches that are specified as obsolete by this patch. The list of obsolete patches is obtained from **Revision Number Summary** in the Summary Information Stream of the patch.
- The SequenceData element contains patch sequencing information for the patch.

There can be multiple SequenceData elements in the XML blob. Each SequenceData element contains the information in one row of the [MsiPatchSequence table](#) of the patch. The SequenceData element contains a ProductCode, Sequence, and Attributes

subelement for the information in the corresponding fields in the MsiPatchSequence table. See the [MsiPatchSequence table](#) section for a description of each field.

## Extracting Applicability Information

The following example shows you how to extract the applicability information for a Windows Installer Patch (.msp file) using **MsiExtractPatchXMLData**. The extracted XML blob is based on the schema definition in MSIPatchApplicability.xsd and returned to szXMLData.

```
#include <windows.h>
#include <msi.h>

#pragma comment( lib, "msi.lib" )

void main()
{
    TCHAR szPatchPath[] = TEXT("c:\\scratch\\RTM-RTMQFE
    TCHAR* szXMLData = NULL;
    DWORD cchXMLData = 0;

    UINT uiStatus = ERROR_SUCCESS;

    // Determine size of XML blob buffer.
    if (ERROR_SUCCESS == (uiStatus = MsiExtractPatchXML
        /*dwReserved: must be 0*/ 0, szXMLData, &c
    {
        // cchXMLData now includes size of szXMLData
        ++cchXMLData;

        szXMLData = new TCHAR[cchXMLData];
        if (ERROR_SUCCESS == (uiStatus = MsiExtract
            /*dwReserved: must be 0*/ 0, szXMLD
        {
            //
            // szXMLData now contains the XML p
            // provided to MsiDetermineApplicat
            // proper format to evaluate patch
```

```
        //
    }

    delete [] szXMLData;
    szXMLData = NULL;
}
}
```

The following example shows you how to extract the applicability information for a Windows Installer Patch (.msp file) in XML form. The extracted XML blob is based on the schema definition in MSIPatchApplicability.xsd and returned in strPatchXML.

```
Dim installer
Set installer = CreateObject("WindowsInstaller.Installer")
strPatchXML = installer.ExtractPatchXMLData("c:\example\patc
```

## Patch Applicability Schema Definition

Copy the following text into Notepad or another text editor to create the schema definition file for the patch applicability information in the XML blob. Name this file MSIPatchApplicability.XSD.

```
<?xml version="1.0" encoding="utf-8" ?>
<xs:schema id="Applicability"
    targetNamespace="http://www.microsoft.com/msi/patch_
    elementFormDefault="qualified"
    xmlns="http://www.microsoft.com/msi/patch_applicabil
    xmlns:mstns="http://www.microsoft.com/msi/patch_app
    xmlns:xs="http://www.w3.org/2001/XMLSchema">

    <xs:element name="MsiPatch">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="TargetProc
                    <xs:complexType>
                        <xs:sequence>
                            <xs:
```



```
        <xs:extension base="GUID">
            <xs:attribute name="Validate" type="boolean" use="optional" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="ValidateVersion">
    <xs:simpleContent>
        <xs:extension base="Version">
            <xs:attribute name="Comparison" type="string" use="optional" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:complexType name="ValidateLanguage">
    <xs:simpleContent>
        <xs:extension base="xs:int">
            <xs:attribute name="Validate" type="boolean" use="optional" />
        </xs:extension>
    </xs:simpleContent>
</xs:complexType>
<xs:simpleType name="intList">
```

```
        <xs:list itemType="xs:int" />
    </xs:simpleType>
    <xs:simpleType name="Identifier">
        <xs:restriction base="xs:string">
            <xs:pattern value="[_a-zA-Z][_a-zA-Z-2
        </xs:restriction>
    </xs:simpleType>
</xs:schema>
```

## See Also

[ExtractPatchXMLData](#)

[MsiDeterminePatchSequence](#)

[MsiDetermineApplicablePatches](#)

[MsiExtractPatchXMLData](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Listing the Files that can be Updated

The **MsiGetPatchFileList** function and **PatchFiles** property of the **Installer Object** take a list of Windows Installer patches (.msp files) and return a list of files that can be updated by the patches. The **MsiGetPatchFileList** function and **PatchFiles** property are available beginning with Windows Installer 4.0.

## Listing Files that can be Updated

The following example shows you how to extract the applicability information for a list of Windows Installer patches (.msp files) using **MsiGetPatchFileList**. The **MsiGetPatchFileList** function is provided the product code for the target of the patches and a list of .msp files delimited by semicolons. This example requires Windows Installer 4.0 running on Windows Vista.

```
#define UNICODE
#include <windows.h>
#include <stdio.h>
#include <Shellapi.h>
#include <msi.h>
#include <Msiquery.h>
#pragma comment(lib, "msi.lib")
#pragma comment(lib, "shell32.lib")

void CloseMsiHandles(MSIHANDLE* phFileListRec, DWORD dwcFiles)

int __cdecl main()
{
    UINT uiRet = ERROR_SUCCESS;
    int argc;
    WCHAR** argv = CommandLineToArgvW(GetCommandLine(),
    &argc);

    MSIHANDLE *phFileListRec = NULL;
    DWORD dwcFiles = 0;
    WCHAR* szProductCode = argv[1];
    WCHAR* szPatchFileList = argv[2];
    if(ERROR_SUCCESS != (uiRet = MsiGetPatchFileList(szProductCode, szPatchFileList,
    &phFileListRec, &dwcFiles)))
    {
        return uiRet;
    }
}
```

```

        printf("MsiGetPatchFileListW(%S, ...) Failed
        return uiRet;
    }

    DWORD cchBuf = 1;
    DWORD cchBufSize = 1;
    WCHAR* szBuf = new WCHAR[cchBufSize];
    if (!szBuf)
    {
        printf("Failed to allocate memory");
        CloseMsiHandles(phFileListRec, dwcFiles);
        return ERROR_OUTOFMEMORY;
    }
    memset(szBuf, 0, sizeof(WCHAR)*cchBufSize);

    for(unsigned int i = 0; i < dwcFiles; i++)
    {
        cchBuf = cchBufSize;
        while(ERROR_MORE_DATA == (uiRet = MsiRecord
        {
            if (szBuf)
                delete[] szBuf;
            cchBufSize = ++cchBuf;
            szBuf = new WCHAR[cchBufSize];
            if (!szBuf)
            {
                printf("Failed to allocate
                CloseMsiHandles(phFileListR
                return ERROR_OUTOFMEMORY;
            }
        }

        if(uiRet != ERROR_SUCCESS)
        {
            printf("MsiRecordGetString(phFileLi
            CloseMsiHandles(phFileListRec, dwcF
            if (szBuf)
                delete[] szBuf;
            return uiRet;
        }
        else
        {

```

```

        printf("File %d:%S\n", i, szBuf);
    }
}

CloseMsiHandles(phFileListRec, dwcFiles);
if (szBuf)
    delete[] szBuf;
return 0;
}

void CloseMsiHandles(MSIHANDLE* phFileListRec, DWORD dwcFiles)
{
    if (!phFileListRec)
        return;

    for (unsigned int i = 0; i < dwcFiles; i++)
    {
        if (phFileListRec[i])
            MsiCloseHandle(phFileListRec[i]);
    }
}
//

```

The following example shows you how to extract the applicability information for a list of Windows Installer patches (.msp files) using **PatchFiles** property of the **Installer Object**. The **PatchFiles** property is provided the product code for the target of the patches and a list of .msp files delimited by semicolons. This example requires Windows Installer 4.0 running on Windows Vista.

```

Dim FileList
Dim installer : Set installer = Nothing
Dim argCount:argCount = Wscript.Arguments.Count

Set installer = Wscript.CreateObject("WindowsInstaller.Installer")

If (argCount > 0) Then
    sProdCode = Wscript.Arguments(0)
    sPatchPckgPath = Wscript.Arguments(1)
    Set FileList = installer.PatchFiles (sProdCode, sPatchPckgPath)

```

```
        For each File in FileList
            Wscript.Echo "Affected file: " & File
        Next
Else
    Usage
End If

Sub Usage
    Wscript.Echo "Windows Installer utility to list files
vbNewLine & " 1st argument is the product code (GUID)
vbNewLine & " 2nd argument is the list of patches"
vbNewLine & _
vbNewLine & "Copyright (c) Microsoft Corporation. All rights reserved."
    Wscript.Quit 1
End Sub
```

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Creating a Patch Package

Developers create a patch package by generating a patch creation file and using [Msimsp.exe](#) to call the [UiCreatePatchPackageEx](#) function in [Patchwiz.dll](#). Msimsp.exe and Patchwiz.dll are provided in the Windows Installer SDK. For more information, see [A Small Update Patching Example](#).

Because the application of a patch to a Windows Installer package results in the installation of the original sources using a new .msi file, the new .msi file must remain compatible with the layout of the original source.

When you author a patch package you must use an uncompressed setup image to create a patch, for example, an administrative image or an uncompressed setup image from a CD-ROM. You must also adhere to the following restrictions:

- Do not move files from one folder to another.
- Do not move files from one cabinet to another.
- Do not change the order of files in a cabinet.
- Do not change the sequence number of existing files. The sequence number is the value specified in the Sequence column of the [File Table](#).
- Any new files that are added by the patch must be placed at the end of the existing file sequence. The sequence number of any new file in the upgraded image must be greater than the largest sequence number of existing files in the target image.
- Do not add new files to the end of an existing cabinet file. All new files must be added after the last cabinet file in the sequence.
- Do not change the primary keys in the File Table between the original and new .msi file versions.

**Note** The file must have the same key in the File Table of both the target image and the updated image. The string values in the File

column of both tables must be identical, including the case.

- Do not author a package with File Table keys that differ only in case, for example, avoid the following table example.

<b>File</b>	<b>Component_</b>	<b>FileName</b>
readme.txt	Comp1	readme.txt
ReadMe.txt	Comp2	readme.txt

The Windows Installer can allow the previous table example when Comp1 and Comp2 are installed on different directories, but then you cannot use Msimsp.exe or Patchwiz.dll to generate a patch for the package. Msimsp.exe and Patchwiz.dll call Makecab.exe, which is case-insensitive and fails.

When using merge modules in the setup, ensure that file sequence numbers and layout adhere to the above guidelines.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patch Packages

A Windows Installer patch (.msp file) is a file used to deliver updates to Windows Installer applications. The patch is a self-contained package that contains all the information required to update the application. A patch package (.msp file) can be much smaller than the Windows Installer package (.msi file) for the entire updated application. For more information about delivering smaller updates to applications, see [Reducing Patch Size](#).

A patch package contains the actual updates to the application and describes which versions of the application can receive the patch. Patches contain at minimum two database transforms. One transform updates the information in the installation database of the application. The other transform adds information that the installer uses for patching files. The installer uses the information provided by the transforms to apply patch files that are stored in the cabinet file stream of the patch package. A patch package does not have a database like an installation package (.msi file.)

Beginning with Windows Installer version 3.0, patch packages can contain information that describe the patching sequence for the patch relative to other updates in the [MsiPatchSequence](#) table and additional descriptive information in the [MsiPatchMetadata](#) table.

Users can install applications and updates from a network administrative image. Although patch packages can be applied to administrative installations, the recommended method to deliver updates is to have users install the original application and then apply the patches to the local instance of the application on to their computer. This keep users in synchronization with the administrative image. If a patch is applied to the administrative installation, all clients of that administrative installation must recache and reinstall the application to receive the update. Until a user recaches and reinstalls, the user is unable to install-on-demand and repair installations from the patched administrative installation.

Beginning with Windows Installer 3.0, non-administrators can apply patches to per-user-managed applications after the patch has been approved as trusted by an administrator. For more information on how to do this, see [Patching Per-User Managed Applications](#). Another method is

to use least privileged user account patching.

**Note** If the [AllowLockdownPatch](#) policy has been set, non-administrator users can apply a patch to an existing application while running an installation at elevated privileges. This method is not recommended because it enables untrusted patches to be applied to an application that can run with elevated privileges.

Patch packages are comprised of the following parts. For more information about the construction of patch packages, see [Creating a Patch Package](#).

## Summary Information Stream

The summary information stream of the patch package provides information about the identity and purpose of the patch.

The summary information stream holds a minimum of the following:

- A GUID that uniquely identifies the patch. The GUID for this patch is appended with a list of GUIDs for earlier patches that are replaced by this patch.
- A semicolon-delimited list of product codes for valid targets for this patch.
- A semicolon-delimited list of transform substorage names in the order they are to be processed.
- A semicolon-delimited list of sources for this patch.

## Transform Substorage

A patch package contains transforms that can add or remove files, registry entries, user interfaces, and customizations. Transforms are included as substorages in the package. A patch package contains two transforms for each target database. One transform is the actual updates to the installation database and is generated from the differences between the original and updated images of the installation package. The other transform adds entries to the [Patch](#), [PatchPackage](#), [Media](#), [InstallExecuteSequence](#), and [AdminExecuteSequence](#) tables.

Information in the substorage ties it to a specific **UpgradeCode**, **ProductCode**, **ProductVersion**, and **ProductLanguage**. A patch package that can be applied to multiple targets contains more than one pair of these transforms.

## Cabinet File Stream

The cabinet file stream included in a patch can contain these types of files:

- Patch files containing the information required to change the old version of the file into the new version. A single patch file can be used to update one or more old versions of a file.
- Additional files being added to the application that are not present in the old version.
- An entire replacement file. In the rare case where the new version of a file is smaller than the patch required to update the old version of that file, the new file can be included in its entirety. These are new files that are installed over their old versions.

## See Also

[Creating a Patch Package](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patch Optimization

Windows Installer can optimize patching to reduce the time that is required to apply patches to installed applications.

**Windows Installer 2.0:** Not supported. For versions of Windows Installer released before Windows Installer 3.0, patching runs a complete repair installation of the application, which can take significantly more time.

**Windows Installer 3.0 and later:** The patching process only changes the parts of an application that are modified by a patch.

**Windows Installer 3.1 and later:** Beginning with Windows Installer 3.1, patch optimization requires that all patches in the transaction have the `OptimizedInstallMode` property set to 1 (one) in the [MsiPatchMetadata Table](#).

If a patch only modifies the following tables, Windows Installer 3.0 or later skips the actions that are associated with all the other tables, even if those actions are listed in the sequence tables of the original application installation package (.msi file).

- [AdminExecuteSequence](#)
- [AdminUISequence](#)
- [Condition](#)
- [CustomAction](#)
- [File](#)
- [FileSFPCatalog](#)
- [InstallExecuteSequence](#)
- [InstallUISequence](#)
- [Media](#)
- [MoveFile](#)
- [MsiAssembly](#)

- [MsiDigitalCertificate](#)
- [MsiDigitalSignature](#)
- [MsiFileHash](#)
- [MsiPatchHeaders](#)
- [Patch](#)
- [PatchPackage](#)
- [Property](#)
- [Registry](#)
- [SFPCatalog](#)
- [TypeLib](#)
- [\\_Columns](#)
- [\\_Storages](#)
- [\\_Streams](#)
- [\\_Tables](#)
- [\\_TransformView Table](#)
- [\\_Validation](#)

To turn off the patch optimization option, use the [DisableFlyWeightPatching](#) policy.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Reducing Patch Size

Beginning with Windows Installer version 3.0, patch authors can use the product baseline cached by the installer to more easily service applications with smaller delta patches. In many cases, a *delta patch* that delivers servicing information to an application can be significantly smaller than a full-file patch or installation package that delivers the same information.

**Windows Installer 2.0:** Not supported. Beginning with Windows Installer 3.0, the installer selectively saves baseline information about files when they are updated.

Windows Installer provides three methods for updating and servicing applications: *small updates*, *minor upgrades*, and *major upgrades*. A small update is also referred to as a quick fix engineering (QFE) update, and a minor upgrade is also referred to as a service pack (SP) update. A typical major upgrade removes a previous application and installs a new application. Windows Installer can deliver servicing information to applications as an *installation package* (.msi file) or as a *patch package* (.msp file).

A Windows Installer patch package that delivers servicing information for a small update or minor upgrade is generally much smaller than the equivalent installation package that delivers the same servicing information. It is recommended that patch packages be used for the distribution of small and minor upgrades. It is recommended that an installation package be used for the distribution of a major upgrade.

Windows Installer patches (.msp files) can be generated from either full files or from file differences (also called file deltas.) A Windows Installer patch generated from file deltas can be much smaller than the equivalent full-file patch. All versions of the Windows Installer can use both full-file patches or delta patches.

Beginning with Windows Installer version 3.0, the installer selectively saves baseline information about files when they are updated. Information about the original base application (the RTM version) and the most recent minor upgrade (service pack) are saved in a private location when the application is installed or receives a minor upgrade.

The installer does the following to minimize the size of the baseline cache:

- No more than two baselines are maintained for each application: a baseline of the file as originally released (RTM) and a baseline of the file at the most recent minor upgrade (service pack.)
- A file is not added to the cache until it is patched. The baseline cache is copy-on-write.
- If the application has never been updated, there are no files in the baseline cache.
- When the application's last servicing was a minor upgrade (service pack) the application is at a baseline level and at most two copies of a file can be present on the computer. One copy of the file is in the target directory of the installation. The other copy can be in the RTM baseline cache.
- When the application's last servicing was a small update (QFE) the application is not at a baseline level and at most three copies of a file can be present on the computer. The first copy of the file is in the target directory of the installation. The second copy of the file is in the RTM baseline cache. The last copy of the file is in the most recent baseline cache.
- The application's baseline cache is removed when the product is uninstalled.

Beginning with Windows Installer version 3.0, the installer can use the baseline cache when patches are applied to the application. The baseline information can be used to apply a delta patch or to revert a file to a previous version during a patch uninstall. This can enable patch authors to benefit from smaller delta patches. If the installer finds that the delta patch cannot be applied to the target file, the installer can attempt to use a file saved in the baseline cache as a starting point. The installer only resorts to requesting the original installation source after trying all the possibilities in the cache.

Adherence to the following guidelines can help patch authors use

Windows Installer version 3.0 patches and the baseline cache to create smaller delta patches:

- Author patches that include the [MsiPatchSequence table](#). This table is required to use the baseline cache and is available beginning with Windows Installer version 3.0.
- Do not set policy that prevents baseline caching. The value of the [MaxPatchCacheSize](#) policy specifies the maximum percentage of disk space that can be used. If the MaxPatchCacheSize policy is set to 0, no additional files are saved in the baseline cache. If the policy is not set, the default is that a maximum of 10% of the disk space can be used. If the total size of the cache reaches the maximum percentage of disk space, no additional files are saved. The policy does not affect files that have already been saved. Even when caching is disabled, the installer can use existing product baseline caches.
- If the first patch applied includes the MsiPatchSequence table, caching is enabled for the application.
- If any patch in the servicing transaction does not include the MsiPatchSequence table, caching is enabled for the application only if a minor upgrade patch (service pack) that includes the MsiPatchSequence table is successfully applied to the product.
- Generate the patch package using patch creation tools such as [Msimsp.exe](#) and [PATCHWIZ.DLL](#).
- Always target patches for the RTM version of the application or a minor upgrade (service pack) version of the application. The targets specified in the [TargetImages table](#) of the Patch Creation Properties (PCP) file should be product check points defined by the first three fields of the [ProductVersion](#) property.
- Never target patches at small update images. The targets for building the patch should not include previous small update upgrade images.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installing Multiple Patches

Beginning with Windows Installer 3.0, multiple patches can be applied to a product in a constant order, regardless of the order that the patches are provided to the system.

**Windows Installer 2.0:** Not supported. Windows Installer versions earlier than version 3.0 always install patches in the order that they are provided to the system.

**Windows Installer 3.0 and later:** The installer can use the information provided in the [MsiPatchSequence](#) table to determine which patches are applicable to the Windows Installer package and in which order the patches should be applied. Applications can use the [MsiDetermineApplicablePatches](#) and [MsiDeterminePatchSequence](#) functions.

The **MsiDetermineApplicablePatches** function determines which patches apply to the Windows Installer package and in what sequence. The function can account for superseded or obsolete patches. This function does not account for products or patches that are installed on the system that are not specified in the set.

The **MsiDeterminePatchSequence** Sequence function can determine the best sequence of application for the patches to a specified installed product. This function accounts for patches that have already been applied to the product, and accounts for obsolete and superseded patches.

When the patch package does not have a [MsiPatchSequence](#) table, the installer always applies the patches in the order that they are provide to the system.

When the patch package contains a mixture of patches with sequence information in the [MsiPatchSequence](#) table and some patches without this information, Windows installer version 3.0 sequences the patches in the order described in the following section: [Sequencing Patches](#).

A Windows Installer package can install no more than 127 patches when installing or updating an application. When many updates are necessary, they should be combined and previous obsolete patches should be

eliminated from the patching sequence.

A patch that should not be used can be eliminated from the patching sequence. This prevents the patch from being applied when the target application is patched. This is different than removing a patch that has already been applied to an application. For more information about eliminating patches from the patching sequence, see [Eliminating Patches](#). For information about removing applied patches, see [Removing Patches](#).

For an example of how Windows Installer applies multiple patches when all have MsiPatchSequence tables, see the [Multiple Patching Example](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Sequencing Patches

Beginning with Windows Installer 3.0, authors can add patch sequencing information to the patch package database in the [MsiPatchSequence](#) table. The installer can use this information to determine which patches are applicable to an installation package, to determine the best patching sequence, and to install patches in an constant order independent of the order they are provided to the system.

**Windows Installer 2.0:** Not supported. Windows Installer versions previous to Windows Installer 3.0 install patches in the order that they are provided to the system regardless of whether they contain an MsiPatchSequence table.

The following are required to use the patch sequencing functionality.

- [Patch packages](#) (.msp files) must have a MsiPatchSequence table containing sequencing information. The installer installs patches that do not have a MsiPatchSequence table in the order that they are provided to the system.
- Patches are installed using Windows Installer 3.0 or later.

Windows Installer version 3.0 has the following functions that applications can use to determine the best patching sequence.

- The [MsiDeterminePatchSequence](#) function takes a list of patches and determines in what sequence they can be applied to an installed product. This function accounts for any patches or products that have already been installed on the system.
- The [MsiDetermineApplicablePatches](#) function takes a list of patches and determines in what sequence they can be applied to an installed product. This function does not account for any patches or products that have already been installed on the system.

Windows Installer version 3.0 can apply multiple patches to a product in a single patching installation. The group of patches can contain patches

that include patching sequence information (a MsiPatchSequence table) and patches that do not. The Windows Installer installs the patch packages without this table in the order that they are provided to the system. The installer accounts for patch packages that lack a MsiPatchSequence table, but that have been marked as obsolete or superseded patches by the method described in the following section.

When Windows Installer version 3.0 installs multiple patches, it follows these steps to determine the sequence in which individual patches are applied to the product:

1. Installed patches without a MsiPatchSequence table are put in the sequence in the order that they were applied to the product. The first patch that was applied is placed first in the sequence.
2. New patches without a MsiPatchSequence table are put in the sequence. These patches are being applied by the current patching installation. They are put in the order that they are provided to the system, and placed after all the patches in step 1.
3. Obsolete patches are eliminated from the sequence of patches.

**Note** A patch package can specify in the **Revision Number Summary** property an explicit list of obsolete patches to be removed by the patch. This list is intended for use by Windows Installer versions earlier than version 3.0. Windows Installer version 3.0 removes the patches marked as obsolete from the sequence, only if the patches do not have the MsiPatchSequence table.

4. The installer steps through the patching sequence and determines which patches are applicable in the given sequence. When multiple patches are applied to a product, each patch in the sequence also transforms the product's installation database (.msi file). A patch is applicable in a particular sequence only if its database transform is capable of taking the **product code**, **version**, **language**, and **upgradecode** that result from applying the transforms of all preceding patch packages to the product

database. The installer eliminates any inapplicable patches from the sequence.

5. The installer begins placing patches that have sequencing information in their MsiPatchSequence table. **Minor upgrade** patches that have the MsiPatchSequence table are placed in the sequence after the patches that were sequenced in previous steps and in the order of their lowest to highest product versions after being upgraded. Windows Installer then eliminates any minor upgrade patches that are inapplicable in this sequence.
6. **Small update** patches targeting minor upgrades having a MsiPatchSequence table, are assigned to the highest version of the minor upgrade patch in the sequence.
7. All small update patches that remain unassigned after the previous steps, and that have the MsiPatchSequence table, are put in the sequence before the first minor upgrade that has the MsiPatchSequence table, and after the .msi installation database and any patches without the MsiPatchSequence table. Windows Installer then eliminates any small update patches that are inapplicable in this sequence.
8. Windows Installer version 3.0 eliminates superseded patches from the sequence. When a patch supersedes patches that occur earlier in the patch sequence, the patch contains all the fixes in the earlier patches. The earlier patches are no longer required. The Windows Installer requires the information in the MsiPatchSequence table to eliminate superseded patches.  
**Note** Patches intended to supersede an earlier set of patches must be authored to supersede the earlier patches in all patch families. Small update patches can only supersede small updates. Minor upgrades can supersede both small updates and other minor upgrades.
9. Small update patches that carry MsiPatchSequence tables, get

sequenced within product versions according to the sequencing information in their MsiPatchSequence tables. This determines the final patching sequence.

A patch that should no longer be used can be eliminated from the patching sequence. For more information about how to eliminate patches from the patching sequence, see [Eliminating Patches](#).

For an example of how the MsiPatchSequence table can be used to apply patches in the order in which they are authored, see the [Multiple Patching Example](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Eliminating Patches

A patch that should no longer be used can be eliminated from the patching sequence. This prevents the patch from being applied when the target application is patched. This is different than removing a patch that is already applied to an application. For information about removing applied patches, see [Removing Patches](#).

## **Windows Installer 3.0 and later:**

Patches that have the [MsiPatchSequence](#) table can use this table to eliminate patches from the patching sequence. A patch can eliminate patches that come before it in the patching sequence, and replace the information from those patches with its own information. Both the patch that specifies which patches to eliminate and the patches being eliminated must have a MsiPatchSequence table that contains information.

If the eliminated patches and replacement patch do not have MsiPatchSequence tables, the patch package can specify a list of patches to be eliminated from the patching sequence in its [Revision Number Summary](#) property. Windows Installer 3.0 ignores this list if either the eliminated or replacement patches have a MsiPatchSequence table.

When the patch package contains patches with sequence information in the MsiPatchSequence table and some patches without this information, Windows installer 3.0 sequences the patches in the order described in the following section: [Sequencing Patches](#).

For example, Patch1, Patch2, and Patch3 can be three patches that do not have the MsiPatchSequence table. Patch2 can be a patch that is only applicable if Patch1 has already been applied to the application. Patch3 can be a later patch that has all the information in Patch1 and also eliminates Patch1 from the patching sequence. This means that when Patch3 is applied, Patch 2 also becomes inapplicable, because it requires Patch1. Any information in Patch2 alone does not get delivered to the application.

**Windows Installer 2.0:** Not supported. The only method available

is to specify the list of patches to be eliminated from the patching sequence in the **Revision Number Summary** property.

**Note** Patch authors should use the [MsiDeterminePatchSequence](#) and [MsiDetermineApplicablePatches](#) functions to determine the sequence of patches that actually get applied to the product because the elimination of some patches can render other patches inapplicable.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Multiple Patching Example

The following example shows how Windows Installer 3.0 and later can be used to apply patches in the order in which they are authored.

## Example

In this example there are three patches, QFE1, QFE2, and ServicePack1, and they each have a [MsiPatchSequence](#) table. These patches have been authored to be applied to version 1.0 of the application.

Patch Name	Patch Type	Sequence Number
QFE1	Small Update	1.1.0
QFE2	Small Update	1.2.0
ServicePack1	Minor Upgrade	1.3.0

The MsiPatchSequence table of each patch has only one record that contains the patch family, product code, and sequence number. The three patches are all applied to the same product and belong to the same patch family, named AppPatch. None of the patches have the **msidbPatchSequenceSupersedeEarlier** attribute.

MsiPatchSequence Table for the QFE1 [small update](#).

PatchFamily	ProductCode	Sequence	Attributes
AppPatch	{18A9233C-0B34-4127-A966-C257386270BC}	1.1.0	

MsiPatchSequence Table for the QFE2 small update.

PatchFamily	ProductCode	Sequence	Attributes
AppPatch	{18A9233C-0B34-4127-A966-C257386270BC}	1.2.0	

MsiPatchSequence Table for ServicePack1 [minor upgrade](#).

PatchFamily	ProductCode	Sequence	Attributes
AppPatch	{18A9233C-0B34-4127-A966-C257386270BC}	1.3.0	

If a user installs version 1.0 of the product, and then applies QFE2, and then at a later date decides to apply QFE1, Windows Installer ensures that the effective sequence of patch application to the product is QFE1 applied ahead of QFE2. If the user applies ServicePack1, then applies QFE2 and QFE1 together at a later date, Windows Installer ensures that the effective sequence of patch application to the product is QFE1 ahead of QFE2 and ahead of ServicePack1.

If ServicePack1 has **msidbPatchSequenceSupersedeEarlier** set in the Attributes column of its MsiPatchSequence table, this means that the service pack contains all the changes in QFE1 and QFE2. In this case, QFE1 and QFE2 are not applied when ServicePack1 is applied.

**Windows Installer 2.0:** Not supported. Versions earlier than Windows Installer 3.0 can install only one patch per transaction and patches are applied in the sequence that they are provided. For the preceding example, if QFE2 is applied first and then QFE1 is applied, that is two transactions and the patches are applied to version 1.0 of the application in the sequence QFE2 followed by QFE1. If ServicePack1 is applied first, then QFE1 or QFE2 cannot be applied in a later transaction because ServicePack1 is a minor upgrade that changes the application version. QFE1 and QFE2 can only be applied to version 1.0 of the application.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Removing Patches

Beginning with Windows Installer version 3.0, it is possible to create and install patches that can be uninstalled singly, and in any order, without having to uninstall and reinstall the entire application and other patches. Windows Installer 3.0 also enables [Patch Packages](#) to be authored with a [MsiPatchSequence Table](#) that contains patch sequencing information. With versions of Windows Installer earlier than Windows Installer 3.0, the only method to remove particular patches from an application is to uninstall the entire patched application and then reinstall without reapplying any patches being removed.

Whether a patch can be uninstalled depends upon how the patch was authored, the version of Windows Installer used to install the patch, and the changes made by the patch to the application. If a patch is not uninstallable, then the only way to remove the patch is to uninstall the entire application and reinstall without applying the patch being removed.

You can uninstall one or more patches using a command line option, the scripting interface, or by calling [MsiRemovePatches](#) from another application. See [Uninstalling Patches](#) for more information about how to uninstall patches.

The value of the [MSIPATCHREMOVE](#) property lists the patches to be uninstalled. For each patch in the list, the installer verifies that the patch is uninstallable. If the user does not have privileges to remove the patch, the patch is unknown for the product, patch policy prevents removal, or the patch was marked as not uninstallable, the installer returns an error that indicates a failed installation transaction. See [Uninstallable Patches](#) for more information about what determines whether a patch is not uninstallable.

Once the patch is verified as removable, the installer removes the patch from the patch application sequence. For more information about how Windows Installer 3.0 determines what sequence to use when applying patches see [Sequencing Patches](#). Note that removing patches from the sequence can cause patches marked obsolete or superseded to become active.

All patches selected for removal are listed in the [MsiPatchRemovalList](#) property. This property is a private property that is set by the installer and

can be used in conditional expressions or queried by [custom actions](#). The property contains the list of patch code GUIDs of patches to be removed. A custom action can determine whether the installation state of the patch is applied, obsolete or superseded by calling the [MsiGetPatchInfoEx](#) or the [PatchProperty](#) property of the [Patch Object](#).

After a patch is removed the state of the application is the same as if the patch was never installed. If possible, the installer restricts the process to the subset of features affected by the patch being removed. The installer automatically sets the [REINSTALL](#) property to the list of affected features. Files that were added by the patch are removed and files that were modified by the patch are overwritten. Files and registry entries are restored to the version expected by the product minus the patch. Features and components that were added by the patch are unregistered from the application. Note that additional content added by the patch can remain on the user's computer if the content is used by another patch that is still applicable.

If a file of a shared component is updated by a patch, the change affects all applications that share the component. When the patch is removed, again, the change affects all applications that share the component. This means that removal of a patch by one application can restore the file of the shared component to a lower version than required by another application. This could fix the first application, but cause the second application to stop working. In this case, the second application can be repaired by reinstalling the second application using the methods describe in [Reinstalling a Feature or Application](#). This will restore the patched version of the file.

## See Also

[MsiEnumapplicationsEx](#)

[MsiGetPatchInfoEx](#)

[MSIPATCHREMOVE](#)

[MsiRemovePatches](#)

[Patch Sequencing](#)

[Patch Uninstall Custom Actions](#)

[Uninstallable Patches](#)

[Uninstalling Patches](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Uninstallable Patches

Whether a patch can be uninstalled depends upon how the patch was authored, the version of Windows Installer used to install the patch, and the changes made by the patch to the application. If a patch is not uninstallable, then the only way to remove the patch is to uninstall the entire application and reinstall without applying the patch being removed.

You can call for the uninstallation of patches applied with Windows Installer version 3.0 by using [Command Line Options](#), the [MsiRemovePatches](#) function, or the [RemovePatches method](#) as described in the [Uninstalling Patches](#) section. The Windows Installer verifies that each of the patches listed for removal in the [MSIPATCHREMOVE](#) property is uninstallable. If the user does not have privileges to remove the patch, the patch is unknown for the product, patch policy prevents removal, or the patch was marked as not uninstallable, the installer returns an error indicating a failed installation transaction.

**Windows Installer 2.0:** Not supported. Patches applied using a version of Windows Installer that is earlier than Windows Installer 3.0 are not uninstallable.

## Patches that are Not Uninstallable

A patch (.msp file) applied to an installed application is not uninstallable in the following cases. The only method to remove a patch that is not uninstallable is to uninstall the patched application and then reinstall the application without reapplying the patch. In this case, you must reapply any patches that you do not wish to be removed from the application.

- Patches applied using a version of Windows Installer that is less than Windows Installer 3.0 are not uninstallable.
- Patches applied to applications installed on a computer that has had the [DisablePatchUninstall](#) policy set by an administrator are not uninstallable. When this [machine policy](#) has been set, no patches on the computer can be uninstalled, even by an administrator.

- Patches that do not have a [MsiPatchMetadata](#) table in their database are not uninstallable.
- Patches that do not include the following row in their MsiPatchMetadata table are not uninstallable. The patch is not uninstallable for other values of Company, Property, and Value.

<b>Company</b>	<b>Property</b>	<b>Value</b>
{Null}	AllowRemoval	1

- The patch has been applied to an application installed in a context for which the user has insufficient privileges to uninstall patches. The words "Not Allowed" in the following table indicate that an administrator or non-administrator user cannot uninstall patches from patched applications installed in this context. The word "Allowed" in this table means that privileges do not prevent an administrator or non-administrator user from uninstalling patches, however for any of the other reasons discussed in this section, it still might not be possible to uninstall the patch.

<b>application Installation Context</b>	<b>Administrator Uninstall of Patch</b>	<b>Non-Administrator Uninstall of Patch</b>
Per-Machine	Allowed	Generally Not Allowed The only exception is if the patch was applied using (LUA) patching. A patch marked as a LUA patch is uninstallable by either administrators or non-administrators. LUA patching is only available for packages installed per-machine from media and require special authoring.
Per-User Non-	Allowed	Allowed

Managed for Current User		
Per-User Non-Managed for Different User	Not Allowed	Not Allowed
Per-User Managed for Current User	Allowed	Not Allowed
Per-User Managed for Different User	Not Allowed	Not Allowed

- A [major upgrade](#) applied by a patch is not uninstallable. Major Upgrades of an application should be performed by installing the upgraded application (.msi file) rather than a patch.
- Patches applied to an administrative installation are not uninstallable. Patching administrative installations is not recommended. The current set of patches should be applied on the user's computer after the user installs the application from the administrative image. This can prevent the [package code](#) cached on the user's computer from becoming different than the package code on the administrative installation. If the package code cached on the user's computer becomes different from that on the administrative installation, reinstall the application from the administrative installation and then patch the client computer.

- When a patch adds new content to any of the tables in the following list, Windows Installer marks the patch as being not uninstallable. An uninstallable patch can add new files, assemblies, registry entries, components, or features to an installation by adding new rows to database tables that are not included in this list.

- [Appld](#)
- [BindImage](#)
- [Class](#)
- [Complus](#)
- [CreateFolder](#)
- [DuplicateFile](#)
- [Environment](#)
- [Extension](#)
- [Font](#)
- [IniFile](#)
- [IsolatedComponent](#)
- [LockPermissions](#)
- [MsiLockPermissionsEx](#)
- [MIME](#)
- [MoveFile](#)
- [MsiServiceConfig](#)
- [MsiServiceConfigFailureActions](#)
- [ODBCAttribute](#)
- [ODBCDataSource](#)
- [ODBCDriver](#)
- [ODBCSourceAttribute](#)
- [ODBCTranslator](#)
- [ProgId](#)
- [PublishComponent](#)

- [RemoveIniFile](#)
- [SelfReg](#)
- [ServiceControl](#)
- [ServiceInstall](#)
- [TypeLib](#)
- [Verb](#)
- **Note** If a patch adds new content to the [RemoveFile](#) or [RemoveRegistry](#) tables, Windows Installer does not mark the patch as being not uninstallable. However, the patch is not uninstallable unless the resource to remove the new content does not already exist in the original installation package. For example, if the patch adds a new row to the RemoveFile table, the removed file cannot be restored by uninstalling the patch if the file is external to the [File table](#). The file must have been authored in the File table of the original package plus applied patches for the patch to be uninstallable.

## See Also

[Patch Sequencing](#)  
[Removing Patches](#)  
[Uninstalling Patches](#)  
[Patch Uninstall Custom Actions](#)  
**[MSIPATCHREMOVE](#)**  
**[MsiEnumapplicationsEx](#)**  
**[MsiGetPatchInfoEx](#)**  
**[MsiRemovePatches](#)**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Uninstalling Patches

Beginning with Windows Installer 3.0, it is possible to uninstall some patches from applications. The patch must be an [uninstallable patch](#). When using a Windows Installer version less than version 3.0, [removing patches](#) requires uninstalling the patch product and reinstalling the product without applying the patch.

**Windows Installer 2.0:** Not supported. Patches applied using a version of Windows Installer that is earlier than Windows Installer 3.0 are not uninstallable.

When you invoke an uninstallation of a patch by any of the following methods, the installer attempts to remove the patch from the first product visible to the application or user requesting the uninstallation. The installer searches for patched products in the following order: per-user managed, per-user unmanaged, per-machine.

## Uninstalling a patch using MSIPATCHREMOVE on a command line

You can uninstall patches from a command by using `msiexec.exe` and the [Command Line Options](#). The following sample command line removes an uninstallable patch, `example.msp`, from an application, `example.msi`, using the **MSIPATCHREMOVE** property and the `/i` command line option. When using `/i`, the patched application can be identified by the path to the application's package (`.msi` file) or the application's [product code](#). In this example, the application's installation package is located at `"\\server\share\products\example\example.msi"` and the application's **ProductCode** property is `"{0C9840E7-7F0B-C648-10F0-4641926FE463}"`. The patch package is located at `"\\server\share\products\example\patches\example.msp"` and the patch code GUID is `"{EB8C947C-78B2-85A0-644D-86CEE8E07C0}"`.

```
Msiexec /i {0C9840E7-7F0B-C648-10F0-4641926FE463}  
MSIPATCHREMOVE={EB8C947C-78B2-85A0-644D-86CEE8E07C0}  
/qb
```

## Uninstalling a patch using the standard command line options

Beginning with Windows Installer version 3.0, you can use the [standard command line options](#) used by Microsoft Windows Operating System Updates (update.exe) to uninstall Windows Installer patches from a command line.

The following command line is the standard command line equivalent of the Windows Installer command line used to uninstall a patch using the **MSIPATCHREMOVE** property. The /uninstall option used with the /package option denotes the uninstallation of a patch. The patch can be referenced by the full path to the patch or by the patch code GUID.

```
Msiexec /package {0C9840E7-7F0B-C648-10F0-4641926FE463} /uninstall {EB8C947C-78B2-85A0-644D-86CEEF8E07C0} /passive
```

**Note** The /passive standard option is not an exact equivalent of the Windows Installer /qb option.

## Uninstalling a patch using the RemovePatches method

You can uninstall patches from script by using the Windows Installer [Automation Interface](#). The following scripting sample removes an uninstalleable patch, example.msp, from an application, example.msi, using the **RemovePatches** method of the **Installer** object. Each patch being uninstalled can be represented by either the full path to the patch package or the patch code GUID. In this example, the application's installation package is located at

"\\server\share\products\example\example.msi" and the application's

**ProductCode** property is "{0C9840E7-7F0B-C648-10F0-4641926FE463}". The patch package is located at

"\\server\share\products\example\patches\example.msp" and the patch code GUID is "{EB8C947C-78B2-85A0-644D-86CEEF8E07C0}".

```
const msiInstallTypeSingleInstance = 2
const PatchList = "{EB8C947C-78B2-85A0-644D-86CEEF8E07C0}"
const Product = "{0C9840E7-7F0B-C648-10F0-4641926FE463}"
```

```
Dim installer
```

```
Set installer = CreateObject("WindowsInstaller.Installer")
installer.RemovePatches(PatchList, Product, msiInstallTypes
```

## Uninstalling a patch using Add/Remove Programs

With Windows XP, you can uninstall patches using Add/Remove programs.

## Uninstalling a patch using the MsiRemovePatches function

Your applications can uninstall patches from other applications by using the [Windows Installer Functions](#). The following code example removes an uninstallable patch, example.msp, from an application, example.msi, using the [MsiRemovePatches](#) function. A patch can be referenced by the full path to the patch package or the patch code GUID. In this example, the application's installation package is located at "\\server\share\products\example\example.msi" and the application's **ProductCode** property is "{0C9840E7-7F0B-C648-10F0-4641926FE463}". The patch package is located at "\\server\share\products\example\patches\example.msp" and the patch code GUID is "{EB8C947C-78B2-85A0-644D-86CEEF8E07C0}".

```
UINT uiReturn = MsiRemovePatches(
    /*szPatchList=*/TEXT("\\server\share\pr
    /*szProductCode=*/ TEXT("{0C9840E7-7F0B-
    /*eUninstallType=*/ INSTALLTYPE_SINGLE_IM
    /*szPropertyList=*/ NULL);
```

## Uninstalling a patch from all applications using MsiRemovePatches function

A single patch can update more than one product on the computer. An application can use [MsiEnumProductsEx](#) to enumerate all the products

on the computer and determine whether a patch has been applied to a particular instance of the product. The application can then uninstall the patch using **MsiRemovePatches**. For example, a single patch can update multiple products if the patch updates a file in a component that is shared by multiple products and the patch is distributed to update both products.

The following example demonstrates how an application can use the Windows Installer to remove a patch from all applications that are available to the user. It does not remove the patch from applications installed per-user for another user.

```
#ifndef UNICODE
#define UNICODE
#endif //UNICODE

#ifndef _WIN32_MSI
#define _WIN32_MSI 300
#endif //_WIN32_MSI

#include <stdio.h>
#include <windows.h>
#include <msi.h>

#pragma comment(lib, "msi.lib")

const int cchGUID = 38;

////////////////////////////////////
// RemovePatchFromAllVisibleapplications:
//
// Arguments:
//     wszPatchToRemove - GUID of patch to remove
//
////////////////////////////////////
//
UINT RemovePatchFromAllVisibleapplications(LPCWSTR wszPatch
{
    if (!wszPatchToRemove)
        return ERROR_INVALID_PARAMETER;

    UINT uiStatus = ERROR_SUCCESS;
```

```

DWORD dwIndex = 0;
WCHAR wszapplicationCode[cchGUID+1] = {0};

DWORD dwapplicationSearchContext = MSIINSTALLCONTEXT_
MSIINSTALLCONTEXT dwInstallContext = MSIINSTALLCONTE

do
{
    // Enumerate all visible applications in all
    // NULL for szUserSid defaults to using the
    uiStatus = MsiEnumProductsEx(/*szapplicatio
    /*szUserSid*/NULL,
    dwapplicationSearchContext,
    dwIndex,
    wszapplicationCode,
    &dwInstallContext,
    /*szSid*/NULL,
    /*pcchSid*/NULL);

    if (ERROR_SUCCESS == uiStatus)
    {
        // check to see if the provided patch
        // registered for this application
        UINT uiPatchStatus = MsiGetPatchInfo
        wszapplicationCode,
        /*szUserSid*/NULL,
        dwInstallContext,
        INSTALLPROPERTY_PATCHSTATE,
        NULL,
        NULL);

        if (ERROR_SUCCESS == uiPatchStatus)
        {
            // patch is registered to t
            wprintf(L"Removing patch %s
            wszPatchToRemove, wszappli

            UINT uiRemoveStatus = MsiRe
            wszPatchToRemove,
            wszapplicationCode,
            INSTALLTYPE_SINGLE_INSTANCE
            L"");

```

```

        if (ERROR_SUCCESS != uiRemo
        {
            // This halts the e
            // you could output
            // enumeration
            return ERROR_FUNC
        }
    }
else if (ERROR_UNKNOWN_PATCH != uiF
{
    // Some other error occurre
    // halts the enumeration ar
    // could output an error ar
    return ERROR_FUNCTION_FAIL
}
// else patch was not applied to th
// (ERROR_UNKNOWN_PATCH returned)
}

    dwIndex++;
}
while (uiStatus == ERROR_SUCCESS);

if (ERROR_NO_MORE_ITEMS != uiStatus)
    return ERROR_FUNCTION_FAILED;

return ERROR_SUCCESS;
}

```

## See Also

[Patch Sequencing](#)  
[Removing Patches](#)  
[Uninstallable Patches](#)  
[Patch Uninstall Custom Actions](#)  
**[MSIPATCHREMOVE](#)**  
**[MsiEnumapplicationsEx](#)**  
**[MsiGetPatchInfoEx](#)**

## **MsiRemovePatches**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patch Uninstall Custom Actions

You can use the [Custom Action Patch Uninstall option](#) to specify that the installer run the custom action only when a patch is uninstalled.

**Windows Installer 4.5 and later:** You can use the Custom Action Patch Uninstall Option to specify that the installer only run the custom action when a patch is uninstalled.

**Windows Installer 4.0 and earlier:**

The Custom Action Patch Uninstall option is not available. There is no method for marking a [custom action](#) within a patch package to be run when the patch is uninstalled because the installer does not apply the patch packages being uninstalled.

To have a custom action run when a particular patch is uninstalled, the custom action must either be present in the original application or be in a patch for the product that is always applied.

Developers can use the [MsiPatchRemovalList](#) property to author a Windows Installer package or patch that performs custom actions on the removal of a patch. The custom action can be authored into the original installation package, a patch that has already been applied to the package, or a patch that is not an [uninstallable patch](#). The custom action can be conditionalized on the **MsiPatchRemovalList** property in the sequence tables. See [Using Properties in Conditional Statements](#) for more information about conditionalizing actions.

The custom action can obtain the GUIDs of patches being removed from the value of the **MsiPatchRemovalList** property. The custom action can determine whether the installation state of the patch is applied, obsolete, or superseded by calling the [MsiGetPatchInfoEx](#) or the [PatchProperty](#) property of the [Patch object](#).

If the custom action requires special metadata from the patch, the patch should contain a custom action that writes the metadata to a registry or file location when the patch is applied. The custom action in the original application or a patch that is always applied can obtain the information needed to remove the patch's changes.

Patches making changes that are difficult to undo correctly should not be

marked as uninstallable patches.

## **See Also**

[Patch Sequencing](#)

[Removing Patches](#)

[Uninstallable Patches](#)

[Uninstalling Patches](#)

**[MSIPATCHREMOVE](#)**

**[MsiEnumapplicationsEx](#)**

**[MsiGetPatchInfoEx](#)**

**[MsiRemovePatches](#)**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patching Per-User Managed Applications

Beginning with Windows Installer 3.0, it is possible to apply patches to an application that has been installed in a per-user-managed context after the patch has been registered as having elevated privileges.

**Windows Installer 2.0:** Not supported. You cannot apply patches to applications that are installed in a per-user managed context using versions of Windows Installer earlier than Windows Installer 3.0.

An application is installed in the per-user-managed state in the following cases.

- The per-user installation of the application was performed using deployment and Group Policy.
- The application was advertised to a specified user and installed by the method described in [Advertising a Per-User Application To Be Installed with Elevated Privileges](#).

Privileges are required to install an application in the per-user-managed context; therefore, future Windows Installer reinstallations or repairs of the application are also performed by the installer using elevated privileges. This means that only patches from trusted sources can be applied to the application.

Beginning with Windows Installer 3.0, you can apply a patch to a per-user managed application after the patch has been registered as having elevated privileges. To register a patch as having elevated privileges, use the [MsiSourceListAddSourceEx](#) function or the [SourceListAddSource](#) method of the [Patch](#) object, with elevated privileges. After registering the patch, you can apply the patch using the [MsiApplyPatch](#) or [MsiApplyMultiplePatches](#) functions, [ApplyPatch](#) or [ApplyMultiplePatches](#) methods of the [Installer Object](#), or the [/p command-line option](#).

**Note** A patch can be registered as having elevated privileges before the application is installed. When a patch has been registered, it remains

registered until the last registered application for this patch is removed.

Patches that have been applied to a per-user managed application cannot be removed without removing the entire application. The patch registrations for a per-user managed application are removed on the removal of the application.

You can also use this method to enable a non-administrator to patch a per-machine application, or you can use least-privilege patching described in [User Account Control \(UAC\) Patching](#).

## Example

The following scripting sample uses the **SourceListAddSource** method to register a patch package located at `\\server\share\products\patches\example.msp` as having elevated privileges. That patch is then ready to be applied to a per-user managed product.

```
const msiInstallContextUserManaged = 1
const msiInstallSourceTypeNetwork = 1

const PatchCode = "{XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX}"
const UserSid = "S-X-X-XX-XXXXXXXXXX-XXXXXXXXXX-XXXXXXXXXX-XXXX"
const PatchPath = "\\server\share\products\patches\"
const PatchPackageName = "example.msp"

Dim installer
Set installer = CreateObject("WindowsInstaller.Installer")

Set patch = installer.Patch(PatchCode, "", UserSid, msiInstallContextUserManaged)
patch.SourceListAddSource msiInstallSourceTypeNetwork, PatchPath, PatchPackageName
patch.SourceListInfo("PackageName") = PatchPackageName
```

## Example

The following code sample uses the **MsiSourceListAddSourceEx** function to register a patch package located at `\\server\share\products\patches\example.msp` as having elevated

privileges. That patch is then ready to be applied to a per-user managed product.

```
#ifndef UNICODE
#define UNICODE
#endif // UNICODE

#ifndef _WIN32_MSI
#define _WIN32_MSI
#endif // _WIN32_MSI

#include <windows.h>
#include <msi.h>

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
// RegisterElevatedPatch
//
// Purpose: register a patch elevated from a network location
//
// Arguments:
//     wszPatchCode <entity type="ndash"/> GUID of patch to register
//     wszUserSid   - String SID that specifies the user account
//     wszPatchPath <entity type="ndash"/> Network location of patch
//     wszPatchPackageName <entity type="ndash"/> Package name
//
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
UINT RegisterElevatedPatch(LPCWSTR wszPatchCode,
LPCWSTR wszUserSid,
LPCWSTR wszPatchPath,
LPCWSTR wszPatchPackageName)
{
// wszUserSid can be NULL
// when wszUserSid is NULL, register patch for current user
// current user must be administrator
if (!wszPatchCode || !wszPatchPath || !wszPatchPackageName)
    return ERROR_INVALID_PARAMETER;

UINT uiReturn = ERROR_SUCCESS;

uiReturn = MsiSourceListAddSourceEx(
/*szPatchCode*/ wszPatchCode,
```

```
/*szUserSid*/ wszUserSid,  
/*dwContext*/ MSIINSTALLCONTEXT_USERMANAGED,  
/*dwOptions*/ MSISOURCETYPE_NETWORK+MSICODE_PATCH,  
/*szSource*/ wszPatchPath,  
/*dwIndex*/ 0);  
if (ERROR_SUCCESS == uiReturn)  
{  
    uiReturn = MsiSourceListSetInfo(  
        /*szPatchCode*/ wszPatchCode,  
        /*szUserSid*/ wszUserSid,  
        /*dwContext*/ MSIINSTALLCONTEXT_USERMANAGED,  
        /*dwOptions*/ MSISOURCETYPE_NETWORK+MSICODE_PATCH,  
        /*szProperty*/ L"PackageName",  
        /*szValue*/ wszPatchPackageName);  
    if (ERROR_SUCCESS != uiReturn)  
    {  
        // Function call failed, return error code  
        return uiReturn;  
    }  
}  
else  
{  
    // Function call failed, return error code  
    return uiReturn;  
}  
  
return ERROR_SUCCESS;  
}
```

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# User Account Control (UAC) Patching

*User Account Control* (UAC) patching enables the authors of Windows Installer installations to identify digitally-signed patches that can be applied in the future by non-administrator users.

If the digital signature does not match the certificate, Windows Vista displays a UAC dialog box requesting administrator authorization before installing the patch. On systems earlier than Windows Vista, the installer will not apply a signed patch that does not match the certificate.

If a non-administrator attempts to apply a patch to an application, and the following conditions have not been met, Windows Vista will notify the user that administrator authorization is required before installing the patch. A non-administrator can continue installing the patch, without needing to obtain additional administrator authorization, provided the following conditions are met.

- The application was installed on Windows Vista or Windows XP using Windows Installer 3.0 or later.

**Windows Server 2008:** Not supported.

- If the application was installed on Windows XP, the application must also have been installed from removable media, such as a CD-ROM or DVD disk. This restriction does not apply if the application was installed on Windows Vista.
- The application was not installed from an [administrative installation](#) source image.
- The application was originally installed per-machine. For information about how to enable non-administrators to apply patches to per-user-managed applications after the patch has been approved as trusted by an administrator, see [Patching Per-User Managed Applications](#).

- The [MsiPatchCertificate table](#) is present in the Windows Installer package (.msi file) and contains the information needed to enable UAC. The table and information may have been included in the original installation package or added to the package by a Windows Installer patch file (.msp file).
- The patches are digitally signed by a certificate listed in the MsiPatchCertificate table. For more information about digital signature certificates, see [Digital Signatures and Windows Installer](#).
- The digital signature on the patch package can be verified against the certificate in the MsiPatchCertificate table. For more information about the use of digital signatures, digital certificates, and **WinVerifyTrust**, see the [Security](#) section of the Microsoft Windows Software Development Kit (SDK).
- The signer certificate used to verify the digital signature on the patch package is valid and has not been revoked.

**Note** Although you cannot sign a patch using an expired certificate, evaluation of a digital signature on a patch does not fail if the certificate has expired. Evaluation uses the current MsiPatchCertificate table, which consists of the MsiPatchCertificate table in the original package and any changes to the table by patches sequenced prior to the current one. A patch can add new certificates to the MsiPatchCertificate table to evaluate patches sequenced after the current patch. A revoked certificate is always rejected.

- Least-privilege patching has not been disabled by setting the [MSIDISABLELUAPATCHING](#) property or the [DisableLUAPatching](#) policy.

A patch that has been applied using UAC patching can also be removed by a non-administrator.

Administrators can apply patches to per-machine installed products regardless of the application's UAC setting.

You can determine whether least-privilege patching is enabled for an application by using the [MsiGetProductInfoEx](#) function to query for the `INSTALLPROPERTY_AUTHORIZED_LUA_APP` property, or by using the [ProductInfo](#) method to query for the "AuthorizedLUApp" property. If the value of either property is 1, the application is enabled for least-privilege user account patching.

An administrator can disable least-privilege patching on the computer by setting the `DisableLUAPatching` policy to 1. You can set the **MSIDISABLELUAPATCHING** property to 1 during the initial installation of an application to prevent least-privilege patching for that application only.

This functionality is available beginning with Windows Installer version 3.0. User Account Control (UAC) patching was called least-privilege user account (LUA) patching in Windows XP. LUA patching is not available on Windows 2000 and Windows Server 2003.

For more information about application compatibility and developing applications that are compatible with User Account Control (UAC), see the UAC information that is provided on [Microsoft Technet](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Database Transforms

See the following sections for more information about database transforms.

- [About Transforms](#)
- [Using Transforms](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# About Transforms

A transform is a collection of changes applied to an installation. By applying a transform to a base installation package, the installer can add or replace data in the installation database. The installer can only apply transforms during an installation.

The installer registers a list of transforms required by the product during the installation. The installer must apply these transforms to the product's installation package when configuring or installing the product. If a listed transform is unavailable, and if the transform source resiliency cannot restore it, the installation fails.

A transform can modify information that is in any persistent table in the [installer database](#). A transform can also add or remove persistent tables in the installer database. Transforms cannot modify any part of an installation package that is not in a database table, such as information in the [summary information stream](#), information in substorages, or files in embedded cabinets.

Transforms have a summary information stream that can contain validation conditions and error conditions. The transform validation and error conditions can be added to the summary information using the [MsiCreateTransformSummaryInfo](#) function. The validation conditions control whether the installer can apply the transform to a given installation database. Validation of the transform can be conditioned upon the values of the [UpgradeCode](#), [ProductCode](#), [ProductVersion](#) and [ProductLanguage](#) properties specified in the transform and those in the installation database. The transform error conditions control which errors get suppressed when the transform is applied. The error conditions included within the transform are overridden by the error conditions specified using the [MsiDatabaseApplyTransform](#) and [ApplyTransform](#) methods.

**Note** Typical customization transforms have no validation conditions or validate against the **ProductCode**. The transforms stored within [patch packages](#) generally have strict validation conditions to ensure that the correct transform is applied to the patch target.

There are three types of Windows Installer transforms:

- [Embedded Transforms](#)
- [Secured Transforms](#)
- [Unsecured Transforms](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Embedded Transforms

Embedded transforms are stored inside the .msi file of the package. This guarantees to users that the transform is always available when the installation package is available. Alternatively, transforms may be provided to users as standalone .mst files.

To add an embedded transform to the transforms list, add a colon (:) prefix to the file name. Because the installer can always obtain the transform from storage in the .msi file, embedded transforms are not cached on the user's computer. Embedded transforms may be used in combination with [secured transforms](#) or [unsecured transforms](#). For more information, see [Applying Transforms](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Secured Transforms

Secured transforms are sometimes necessary for security reasons. Secured transforms are stored locally on the user's computer in a location where, on a secure file system, the user does not have write access. Such transforms are cached in this location during the installation or advertisement of the package. Only administrators and local system have write access to this location. A non-admin user would not be able to modify the transform file. During subsequent [installation-on-demand](#) or [maintenance installations](#) of the package, the installer uses the cached transforms.

To specify secured transform storage, set the [TransformsSecure](#) policy, set the **TRANSFORMSSECURE** property, or pass the @ or | symbol in the transforms list. Note that you cannot include secured and unsecured transforms in the same transforms list. See [Applying Transforms](#).

The removal of the product by any user removes all secured transforms for that product from the user's computer.

If the installer finds that a secured transform is not locally available, it then attempts to restore the transform cache from a source. Secure transforms can be secure-at-source or secure-full-path:

- [Secure-at-source transforms](#) that are missing from the local transform cache are restored from the root of the source of the .msi file.
- [Secure-full-path transforms](#) that are missing from the local transform cache are restored from the original full path specified by the transforms list.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Secure-At-Source Transforms

Secure-at-source transforms must have a source located at the root of the source for the package. When the package is installed or advertised, the installer saves the transforms in a cache where, on a secure file system, the user does not have write access. If the local copy of the transform becomes unavailable, the installer searches for a source to restore the cache. The method is the same as searching the source list for an .msi file. For more information, see [Source Resiliency](#). The removal of a product by any user removes all secure transforms for that product from the user's computer.

To apply secure-at-source transforms at the installation of a package, set the [TransformsSecure policy](#) or the **TRANSFORMSSECURE** property, or make @ the first character passed in the transform list. Each transform must be indicated by a file name and must be located at the root of the source for the package. If any of the transforms are not located at the root of the package source, the installation fails. For more information, see [Applying Transforms](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Secure-Full-Path Transforms

Secure-full-path transforms must have a source located at the full path specified in the **TRANSFORMS** property. When the package is installed or advertised, the installer saves the transforms in a cache where, on a secure file system, the user does not have write access. If the local copy of the transform becomes unavailable, it may only restore the cache using the specified path. The removal of a product by any user removes all secure transforms for that product from the user's computer.

To apply secure-full-paths transforms at the installation of a package, set the **TransformsSecure policy** or the **TRANSFORMSSECURE** property or make | the first character passed in the transform list. The full paths to the source of each transform must be passed in the transforms string. If any relative paths are in the list, the installation fails. The transform source does not have to be located with the source of the package.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Unsecured Transforms

Transforms that have not been secured as described in [Secured Transforms](#) are unsecured transforms by default.

To apply an unsecured transform when installing a package, pass the transform file names in the **TRANSFORMS** property or command line string. Do not begin the string with the @ or | characters or set the [TransformsSecure policy](#) or the **TRANSFORMSSECURE** property. Note that you cannot combine unsecured transforms and [secured transforms](#) in the same transforms list.

If the package is installed or advertised in the per-user [installation context](#), and has unsecured transforms, the installer saves the transform source in the Application Data folder in the user's profile. This enables a user to maintain their customization of a product while moving between computers.

If the package is installed or advertised in the per-machine installation context, and uses unsecured transforms, the installer saves the transform source in the %windir%\Installer folder.

During a first-time installation of the package, the installer first searches for the transform at the source supplied by the **TRANSFORMS** property or command line string. If this source is unavailable, the installer then searches for the transform at the source of the package next to the .msi file.

During a [maintenance installation](#), the installer searches for the transform at the cache location. If the cached copy of the transform becomes unavailable, the installer searches for the transform at the source of the package next to the .msi file.

For more information, see [Applying Transforms](#) and [Source Resiliency](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Transforms

Transforms increase the flexibility of application installation by providing a way of applying changes to a database without altering the original database. For more information, see [Database Transforms](#).

The following sections discuss using transforms:

[Applying Transforms](#)  
[Generating a Transform](#)  
[Using Transforms to Add Resources](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Applying Transforms

The **TRANSFORMS** property contains the list of transforms for an installation package. The installer applies all the transforms in the transforms list at every installation, advertisement, installation-on-demand, or maintenance installation of the package.

The **TRANSFORMS** property is set by specifying the list of transforms on the command line; however, when using either the /jm or /ju **command line option**, the transforms list must be specified using the /t option.

Note that the transforms list cannot be modified once installed and can only be removed by uninstalling the application.

**Note** A Windows Installer package can apply no more than 255 transforms when installing an application or update. When many transforms are necessary, they should be combined and previous obsolete transforms should be eliminated.

The following table provides examples of various transforms strings that could be added to the transforms list.

Transforms string	Description
transform1.mst;;transform2.mst;;transform3.mst	Transform2.mst and transform3.mst are <b>embedded transforms</b> . transform1.mst is a <b>secure-at-source transform</b> only if the <b>TRANSFORMSSECURE</b> property or <b>TransformsSecure policy</b> is set, otherwise transform1 is an <b>unsecured transform</b> .
\\server\share\path\transform1.mst;;transform2.mst	Transform2.mst is an <b>embedded transform</b> . transform1.mst is a <b>secure-full-path transform</b> only if the

	<p><b>TRANSFORMSSECURE</b> property or TransformsSecure policy is set, otherwise transform1 is an <b>unsecured transform</b>.</p>
<p>@:transform2.mst;transform1.mst @transform1.mst;;transform2.mst</p>	<p>Transform2.mst is an embedded transform. transform1.mst is a standalone <b>secure-at-source transforms</b>.</p>
<p> \server\share\path\transform1.mst;;transform2.mst  :transform2.mst;\\server\share\path\transform1.mst</p>	<p>Transform2.mst is an embedded transform. transform1.mst is a Standalone <b>secure-full-path transforms</b>.</p>

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Generating a Transform

The following procedure describes a scenario to generate a transform that permanently hides a feature.

## ► To permanently hide a product feature

1. Copy the original installation package.
2. Alter the copy by setting the value of the Display column in the [Feature table](#) to zero. This specifies hiding the feature.
3. Create a transform from the original installation package to the new installation packages by calling [MsiDatabaseGenerateTransform](#).
4. Create the summary information in the transform by calling [MsiCreateTransformSummaryInfo](#).

The transform is then ready to be applied during installation. For more information, see [Applying Transforms](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Transforms to Add Resources

Resources that are needed for the customization of an application, such as corporate templates, can be deployed with the application by including a transform with the application's installation package. The following rules should be followed when deploying resources, such as files, registry keys, or shortcuts, using a transform.

- The transform should add one or more new components to the installation database to contain the additional resources. The transform should not add resources to a component that already exists in the installation.
- The transform should add one or more new features to the installation database to contain these new components. These new features should not be the parents of any existing features, but new parent and child features may be introduced together. New features should be given names that are unique across all other transforms for this product. No two transforms should ever add a feature to this product that has the same name listed in the Feature column of the [Feature table](#) and containing different components or resources.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Package Validation

It is strongly recommended that you run validation on every new or newly modified installation package before attempting to install the package for the first time.

Package validation includes the following three processes:

- [Internal Validation](#)
- [String-Pool Validation](#)
- [Internal Consistency Evaluators - ICEs](#)

Merge modules should be validated by using the method described in [Validating Merge Modules](#).

Evalcom2.dll provides a COM object that implements validation operations for installation packages and merge modules. The main object implements interfaces for C/C++ programs. For more information, see [Validation Automation](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Internal Validation

When authoring an installation package, you can use the [MsiViewModify](#) function or the View.Modify method to ensure that the data you enter is syntactically correct. For more information, see [Modify method](#). At the lowest level, a database table's column can store integers (short or long), strings, or binary data. However, an installation package requires specific integers or strings in certain tables. These specifications are maintained in the [\\_Validation table](#). For example, the FileName column of the [File table](#) is a string column, but it specifically stores a file name. Therefore, not only should your entry be a string, but it should also follow the requirements for naming files.

The various validation enum values used with the **MsiViewModify** function allow for immediate validation at different levels. The MSIMODIFY\_VALIDATE\_FIELD enum can be used to validate individual fields of a record. It does not validate foreign keys. The MSIMODIFY\_VALIDATE enum validates an entire row and includes foreign key validation. If you are inserting a new row into a table, use the MSIMODIFY\_VALIDATE\_NEW enum to verify that you are adding valid data as well as using unique primary keys. An insert fails if the primary keys are not unique. If a call to **MsiViewModify** with one of the validation enums returns an error, you can make repeated calls to [MsiViewGetError](#) for diagnosing the problem. **MsiViewGetError** indicates the column where the error occurred as well as the enum value to help fix the problem. For more information, see [GetError method](#).

You can also use internal validation to ensure that other authors enter data correctly into your custom table. Add each of the columns of your custom table to the [\\_Validation table](#) using the custom table name and column name as the primary key. Provide a description or purpose of each column in the Description column of the [\\_Validation table](#). Enter the applicable requirements for each column using the Nullable, MinValue, MaxValue, KeyTable, KeyColumn, Category, and Set columns:

- If the column is Nullable, enter a 'Y'. If not, enter a 'N'.
- If the column is an integer column and can contain a range of integers, enter that range using the MinValue and MaxValue

columns.

- Foreign key columns are identified using the KeyTable and KeyColumn columns.
- For string columns, specify a Category such as Filename, GUID, Identifier, etc. For more information, see [column data types](#).
- If the data can only pertain to a specific number of values (string or integer), use the Set column to list the acceptable values.

The following is a list of the columns (in addition to Table, Column, and Description) in the \_Validation table that can be filled in if your column is of the specified type. (Note that you do not have to fill in all the columns.)

Type	Columns
Integer	Nullable, MinValue, MaxValue, KeyTable, KeyColumn, Set
String	Nullable, KeyTable, KeyColumn, Category, Set, MinValue, MaxValue
Binary	Nullable, Category (Category must be "Binary")

Authoring environments may make use of MSIMODIFY\_VALIDATE\_DELETE. This enum assumes that you want to delete the row. No field or foreign key validation is performed. This enum actually performs a reverse foreign key validation. It checks the \_Validation table for references in the KeyTable and KeyColumn columns for the table to which the "deleted" row belongs. If there are columns that list the table containing the "deleted" row as a potential foreign key, it cycles through that column to see if any of the values reference values in the "deleted" row. An error return means that you break the relational integrity of the database by deleting the row.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# String-Pool Validation

The Windows Installer stores all database strings in a single shared string pool to reduce the size of the database and to improve performance. The only means of validating the string pool is to use the MsiInfo tool found in the Windows Installer SDK.

String pool verification consists of two main checks:

- [DBCS string tests](#)
- [Reference count verification](#)

## DBCS String Tests

The DBCS string tests scan each string in the database for two criteria: For packages with a neutral code page marked, if any character is an extended character (greater than 127), the string is flagged and a message is displayed saying that the code page of the database is invalid because these characters require a specific code page to be rendered consistently on all systems.

If the database has a code page, each string is scanned for an invalid DBCS indicator. If a non-neutral string has been improperly marked, the characters will not render correctly. (This is most commonly caused by forcing the code page to a particular value using the `_ForceCodepage` table with non-neutral strings already in the database.) Note that this check requires that the code page of the database be installed on the system.

If there is a code page problem, the user may fix the error by using the `_ForceCodepage` table to force the code page of the database to the appropriate value. For more information, see [Code Page Handling](#).

## Reference Count Verification

To verify the reference counts of all strings, every table is scanned for string values, a count of each distinct string is kept, and the result is compared to the stored reference count in the database string pool.

If there is a string reference count problem, the user should immediately export each table of the database using [MsiDatabaseExport](#), create a new database, and import the tables into the new database using [MsiDatabaseImport](#). The new database then has the same content as the old database, but the string reference counts are correct. Adding or deleting data from a database with a corrupt string pool can increase corruption of the database and loss of data, so taking these steps quickly is important to prevent further data loss.

When rebuilding databases, remember to embed any necessary storages and streams in the new database (see [\\_Streams Table](#) and [\\_Storages Table](#)) and be aware of code page issues. Also remember to set each of the necessary [Summary Information Stream](#) properties.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Internal Consistency Evaluators - ICEs

Internal consistency evaluators, also called ICEs, are custom actions written in VBScript, JScript, or as a DLL or EXE. When these custom actions are executed, they scan the database for entries in database records that are valid when examined individually but that may cause incorrect behavior in the context of the whole database. Note that this is different than the validation done on individual records using [MsiViewModify](#).

For example, the [Component table](#) may list several components that are all valid when tested individually with **MsiViewModify**. However, **MsiViewModify** would not catch the error when two components use the same [GUID](#) as their component code. The custom action [ICE08](#) is designed to validate that the Component table does not contain duplicate component code GUIDs.

ICE custom actions return four kinds of messages:

- **Errors** Error messages report database authoring that cause incorrect behavior. For example, duplicate component GUIDs cause the installer to incorrectly register components.
- **Warnings** Warning messages report database authoring that causes incorrect behavior in certain cases. Warnings can also report unexpected side-effects of database authoring. For example, entering the same property name in two conditions that differ only by the case of letters in the name. Because the installer is case-sensitive, the installer treats these as different properties.
- **Failures** Failure messages report the failure of the ICE custom action. Failure is commonly caused by a database with such severe problems that the ICE cannot even run.
- **Informational** Informational messages provide information from the ICE and do not indicate a problem with the database. Often they are information about the ICE itself, such as a brief description. They can

also provide progress information as the ICE runs.

For more information, see [Using Internal Consistency Evaluators](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Internal Consistency Evaluators

To validate a database, use a special validation tool to merge a .cub file containing the [Internal Consistency Evaluators](#) (ICEs) into your database, execute the ICEs, and report the results. Several such tools are provided in the Microsoft Windows Software Development Kit (SDK). Authoring environments from third-party vendors also may incorporate the ICE validation system into their authoring environment. It is also possible to write your own tool to perform ICE validation. Most ICE validation tools merge the .cub file and your database into a third, temporary database. Windows Installer displays warnings, errors, debugging information, and API errors as it executes each ICE in the .cub file. When the installer finishes executing the ICEs it closes the .msi file, .cub file, and temporary database without saving any changes. The .msi file and .cub file remain unchanged by the validation test.

ICE custom actions communicate to the user by calling [MsiProcessMessage](#) and posting an `INSTALLMESSAGE_USER` message. An ICE message commonly returns information such as the following:

- Name of the ICE that has found an error
- Date the ICE was created
- Author of the ICE
- Date the ICE was last modified.
- Description of the API error causing the ICE to fail
- Description of the error
- A warning to the user
- Name of the database table containing the error or warning
- Name of the table column containing the error or warning
- Primary keys of the table containing the error or warning
- A URL to a HTML file giving debugging suggestions
- A string that can contain other information

Authors of installation packages can write ICE custom actions or use the standard set of ICEs included in the .cub files provided with the SDK. For more information on how to write an ICE, see [Building an ICE](#).

After writing the appropriate ICEs for validation, a developer must collect the custom actions together into an .msi database, called a .cub file, that contains only the ICEs and their required tables. A .cub file cannot be installed and is used only to store and provide access to ICE custom actions. For more information on making .cub files, see [Building an ICE Database](#). Alternatively, developers can validate their installation package using the existing ICEs described in [ICE Reference](#). These ICEs can be obtained from standard .cub files provided with the SDK.

The installation of the database table editor Orca or the validation tool msival2 provides the Logo.cub, Darice.cub, and Mergemod.cub files. The set of ICEs in the Logo.cub file is a subset of those in the Darice.cub file. If your package passes validation using Darice.cub, it will pass with Logo.cub. Mergemod.cub contains a set of ICEs used to validate merge modules. For more information, see [Merge Module ICE Reference](#).

#### ► **To validate an installation package**

1. Obtain or author the appropriate ICE custom actions. You may be able to use one or more of the existing ICEs described in the [ICE Reference](#). If your validation requires an ICE not yet in this list, you can create a new ICE as described in [Building an ICE](#).
2. Prepare an ICE Database containing all the ICE custom actions. See the section [Building An ICE Database](#) for information about preparing a .cub file.
3. Provide the .cub file and the .msi file to a package validation tool such as [Orca.exe](#) or [Msival2.exe](#).

Note that merge modules should be validated as described in [Validating Merge Modules](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Building An ICE

If you are unable to find the [Internal Consistency Evaluators](#) you need among the existing ICE custom actions listed in the [ICE Reference](#), you will need to prepare your own ICE to validate your package.

When authoring ICE custom actions you should do the following:

- Base the ICEs only on custom actions of types listed in the table shown.
- Call [MsiProcessMessage](#) and post an INSTALLMESSAGE\_USER type of message. When authoring your ICE messages follow the message format in the [ICE Message Guidelines](#).
- Write your ICE such that it captures any API errors and always returns ERROR\_SUCCESS. This is necessary to allow subsequent custom actions to run following the failure of an ICE.

ICE custom actions are limited to the following custom action types.

Custom action type	Description
<a href="#">Custom Action Type 1</a>	DLL in binary stream
<a href="#">Custom Action Type 2</a>	EXE in binary stream
<a href="#">Custom Action Type 5</a>	JScript in binary stream
<a href="#">Custom Action Type 6</a>	VBScript in binary stream
<a href="#">Custom Action Type 37</a>	JScript code as string
<a href="#">Custom Action Type 38</a>	VBScript code as string

When authoring an ICE custom action, do not do the following:

- Do not assume that the handle to the engine that the ICE receives is an installation instance of the installer database. If it is not a installation instance, certain properties are not defined, the source and target directories are not resolved, and current feature states are

not defined.

- Do not rely on the prior execution, or non-execution, of any installer action, custom action, or another ICE. Because a prior ICE may have created temporary columns in any table, authors should reference columns by name whenever possible. ICEs should cleanup any temporary columns or tables before they exit.
- Do not assume that authors have access to an image of the source directory of the database.
- Do not assume that changes made to the database do not persist.

## See Also

[Sample ICE in C++](#)

[Sample ICE in VBScript](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE Message Guidelines

ICE custom actions communicate by calling [MsiProcessMessage](#) and posting an `INSTALLMESSAGE_USER` type message.

When authoring a message string for an ICE custom action, format the string as follows.

*Name of ICE <tab> Message Type <tab> Description <tab> Help URL or location <tab> Table Name <tab> Column Name <tab> Primary Key <tab> Primary Key <tab> Primary Key . . . (repeat for as many primary keys as needed)*

The first three fields of the string are required in every message.

The Message Type field specifies whether the ICE reports a Failure, Error, Warning, or Informational message.

Value	Message type
0	Failure message reporting the failure of the ICE custom action.
1	Error message reporting database authoring that cause incorrect behavior.
2	Warning message reporting database authoring that causes incorrect behavior in certain cases. Warnings can also report unexpected side-effects of database authoring.
3	Informational message.

If help is not available, the Help URL field may be the empty string.

Error and Warning messages should provide the Table Name, Column Name, and Primary Key fields. If any of these fields are omitted, all of the fields following the first blank field must be left out of the message. For example, a table name is provided without a column name and primary keys or a table name and a column name is provided with no primary keys. However a column name and primary keys cannot be used without a table name. Multiple primary keys may be listed until all the primary keys in that table have been given values.

## Examples

The first message illustrated by the [Sample ICE in C++](#):

```
"ICE01\t3\tCreated 04/29/1998 by <insert author's name here>."
```

The second message posted by the sample ICE:

```
"ICE01\t3\tLast modified 05/06/1999 by <insert author's name here>."
```

The third message posted by the sample ICE.

```
"ICE01\t3\tSimple ICE to illustrating the ICE concept".
```

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Sample ICE in C++

This sample code is from an ICE custom action ([ICE01](#)). It validates that the ICE mechanism is working by displaying the time. The ICE posts a message giving the time the installer called the ICE. This ICE should never return an error.

```
//
// test of external database access
#include <windows.h>
#include <stdio.h>
#include <tchar.h>
#include <strsafe.h>
#include <MsiQuery.h>
#pragma comment(lib, "msi.lib")

////////////////////////////////////
// ICE01 - simple ICE that does not test anything
UINT __stdcall ICE01(MSIHANDLE hInstall)
{
    // setup the record to describe owner and date created
    PMSIHANDLE hRecCreated = ::MsiCreateRecord(1);
    ::MsiRecordSetString(hRecCreated, 0, TEXT("ICE01\ts"));

    // post the owner message
    ::MsiProcessMessage(hInstall, INSTALLMESSAGE(INSTALLMESSAGE_OWNER));
    // setup the record to describe the last time the ICE was called
    ::MsiRecordSetString(hRecCreated, 0, TEXT("ICE01\ts"));

    // post the last modification message
    ::MsiProcessMessage(hInstall, INSTALLMESSAGE(INSTALLMESSAGE_LASTMODIFICATION));

    // setup the record to describe what the ICE evaluated
    ::MsiRecordSetString(hRecCreated, 0, TEXT("ICE01\ts"));

    // post the description of evaluation message
    ::MsiProcessMessage(hInstall, INSTALLMESSAGE(INSTALLMESSAGE_DESCRIPTION));
    // time value to be sent on
    TCHAR szValue[200];
    DWORD cchValue = sizeof(szValue)/sizeof(TCHAR);
```

```
// try to get the time of this call
if (MsiGetProperty(hInstall, TEXT("Time"), szValue,
    StringCchCopy(szValue, sizeof("(none)"/si

// setup the record to be sent as a message
PMSIHANDLE hRecTime = ::MsiCreateRecord(2);
::MsiRecordSetString(hRecTime, 0, TEXT("ICE01\t3\t0
::MsiRecordSetString(hRecTime, 1, szValue);

// send the time
::MsiProcessMessage(hInstall, INSTALLMESSAGE(INSTAL

return ERROR_SUCCESS; // allows other ICES will cor
}
```

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Sample ICE in VBScript

This sample code is from an ICE custom action ( [ICE08](#)). The ICE validates that every component in the [Component table](#) has a unique [GUID](#). No validation is done if the Component table does not exist.

```
Function ICE08()  
  
    'Give creation data  
    Set recInfo=Installer.CreateRecord(1)  
    recInfo.StringData(0)="ICE08" & Chr(9) & "3"  
        & Chr(9) & "Created 05/21/98 by <insert  
        author's name here>"  
    Message &h03000000, recInfo  
  
    'Give last modification date  
    recInfo.StringData(0)="ICE08" & Chr(9) & "3"  
        & Chr(9) & "Modified 05/21/98 by <insert  
        author's name here>"  
    Message &h03000000, recInfo  
  
    'Give description of test  
    recInfo.StringData(0)="ICE08" & Chr(9) & "3"  
        & Chr(9) & "Checks for duplicate GUIDs  
        in Component table"  
    Message &h03000000, recInfo  
  
    'Is there a Component table in the database?  
    iStat = Database.TablePersistent("Component")  
    If 1 <> iStat Then  
        recInfo.StringData(0)="ICE08" & Chr(9) & "2"  
            & Chr(9) & "Table: 'Component' missing.  
            ICE08 cannot continue its validation."  
            & Chr(9) & "http://mypage2"  
        Message &h03000000, recInfo  
        ICE08 = 1  
    Exit Function  
    End If  
  
    'process component table  
    Set view=Database.OpenView("SELECT
```

```

        `Component`, `ComponentId` FROM
        `Component` ORDER BY `ComponentId`)
view.Execute
Do
Set rec=view.Fetch
If rec Is Nothing Then Exit Do

'compare for duplicate
If lastGuid=rec.StringData(2) Then
rec.StringData(0)="ICE08" & Chr(9)
    & "1" & Chr(9) & "Component: [1]
    has a duplicate GUID: [2]" & Chr(9)
    & "http://mypage2" & Chr(9) &
    "Component" & Chr(9) & "ComponentId" & Chr(9) & "[1]"
Message &h03000000,rec
End If

'set for next compare
lastGuid=rec.StringData(2)
Loop

'Return iesSuccess
ICE08 = 1

End Function

```

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Building an ICE Database

After selecting the appropriate [ICEs](#) for validation, a developer must collect the custom actions together into an ICE database. A .cub file is a standard .msi database that contains only ICEs and their required tables. A .cub file cannot be installed and is used only to store and provide access to ICE custom actions.

A .cub file contains the following database tables.

Table	Description
<a href="#">Binary</a>	The script files, DLLs, and EXEs of the ICE custom actions that are referenced in the CustomAction table.
<a href="#">CustomAction</a>	Each record in this table corresponds to an ICE custom action included in the .cub file.
<a href="#">_ICESequence</a>	This table lists the ICE custom actions included in the .cub file in their execution sequence. The ICE custom actions listed in this table are executed by calling <a href="#">MsiSequence</a> , or individually executed using <a href="#">MsiDoAction</a> .
<a href="#">_Validation</a>	This table contains the .cub file entries that are to be merged into the _Validation table.
<a href="#">_Special</a>	Any special processing tables required by particular ICE custom actions must be included in the .cub file. The name of these tables must have a leading underscore.

See [Sample .cub File](#).

## See Also

[Building An ICE](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Sample .cub File

This sample illustrates the layout of a .cub file containing two **ICEs**. The installer executes the custom actions in the sequence: ICE01 and ICE08.

The custom action ICE01 is a **Custom Action Type 1**. It is an entry point to a DLL that is stored as a stream in the .cub file. This stream is listed in the Binary Table ice.dll.

The custom action ICE08 is a **Custom Action Type 6**. It is an entry point to a function in VBScript that is stored as a stream in the .cub file. This stream is listed in the Binary Table as ice.vbs.

## Binary Table

Name	Data
ice.vbs	<i>Unformatted binary data of ice.vbs</i>
ice.dll	<i>Unformatted binary data of ice.dll</i>

## CustomAction Table

Action	Type	Source	Target
ICE01	1	ice.dll	ICE01
ICE08	6	ice.vbs	ICE02

## \_ICESequence Table

Action	Condition	Sequence
ICE01		10
ICE08		20

## \_Special Table

ICE01 and ICE08 do not require the inclusion of special processing tables. When the .cub file contains special tables they must also be included in the \_Validation Table.

### [\\_Validation Table](#)

<b>Table</b>	<b>Column</b>	<b>Nullable</b>	<b>MinValue</b>	<b>MaxValue</b>	<b>KeyTable</b>	<b>KeyCo</b>
Binary	Name	N				
Binary	Data	N				
CustomAction	Action	N				
CustomAction	Type	N				
CustomAction	Source	Y				
CustomAction	Target	Y				
_ICESequence	Action	N				
_ICESequence	Condition	Y				
_ICESequence	Sequence	Y				

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE Reference

An ICE is used to validate installation packages. The table in this topic identifies each ICE. For information about ICEs used to validate merge modules, see [Merge Module ICE Reference](#).

ICE	Description
<a href="#">ICE01</a>	Simple test of ICE mechanism.
<a href="#">ICE02</a>	Circular reference test for File-Component, Registry-Component KeyPaths.
<a href="#">ICE03</a>	Basic data and foreign key validation.
<a href="#">ICE04</a>	Validates file sequence numbers against the LastSequence numbers of the <a href="#">Media Table</a> .
<a href="#">ICE05</a>	Validates for "required" entries in particular tables.
<a href="#">ICE06</a>	Validates for missing column or tables in the database. Any column defined in the _Validation table must be found in the database.
<a href="#">ICE07</a>	Validates that fonts are installed to the FontsFolder.
<a href="#">ICE08</a>	Checks for duplicate GUIDs in the ComponentId column of the Component table.
<a href="#">ICE09</a>	Validates that the permanent bit is set for every component marked for installation into the SystemFolder.
<a href="#">ICE10</a>	Ensures that advertise feature states among children and parents are compatible.
<a href="#">ICE12</a>	Validates type 35 and type 51 custom actions and their locations in the sequence tables.
<a href="#">ICE13</a>	Validates that dialogs are not listed as actions in the execute sequence tables. Dialog actions are only allowed in the user interface sequence tables.
<a href="#">ICE14</a>	Validates that feature parents do not have the msidbFeatureAttributesFollowParent bit set. Also validates that the entries in the Feature and Feature_Parent columns are not the same in the same record.

ICE15	Validates that a circular reference exists between every entry in the MIME table and the corresponding extension in the Extension table.
ICE16	Validates that the ProductName in the Property table is not greater than 63 characters in length.
ICE17	Validates control type dependencies in the Control table. Covers PushButtons, RadioButtonGroups, ListBoxes, ListViews, and ComboBoxes.
ICE18	Validates the KeyPath column of the Component table when it is null. In this case, the key path is a Directory.
ICE19	Validates the advertising tables: Class, TypeLib, Extension, PublishComponents, and Shortcut.
ICE20	Validates that the required dialogs are in the Dialog table.
ICE21	Validates that all components in the Component table map to a feature in the FeatureComponents table.
ICE22	Validates that the Feature_ and Component_ columns in the PublishComponent table.
ICE23	Validates the tab order of controls in all dialog boxes.
ICE24	Validates certain properties in the Property table.
ICE25	Verifies merge module dependencies and merge module exclusions.
ICE26	Validates required and prohibited actions in the sequence tables.
ICE27	Validates the organization and order of the sequence tables.
ICE28	Validates actions that must not be separated by ForceReboot.
ICE29	Validates that your stream names remain unique if truncated to the 62-character limit.
ICE30	Validates that the installation of components containing the same file never install the file more than one time in the same directory.
ICE31	Validates the text styles listed in the Text column of the control table.
ICE32	Compares the column definitions to validate that keys and foreign

	keys are of the same size and type.
ICE33	Checks the registry table for entries that belong in other tables.
ICE34	Validates that every group of radio buttons has a default.
ICE35	Validates that any files from a cabinet file cannot be set to run from the source.
ICE36	Validates that icons listed in the Icon table are used in the Class, ProgID, or Shortcut tables.
ICE38	Validates that components installed under the user's profile use a registry key under HKCU as their key path.
ICE39	Validates the Summary Information stream of the database.
ICE40	Performs various miscellaneous checks.
ICE41	Validates that entries in the Extension and Class tables refer to components belonging to the referenced feature.
ICE42	Checks that Class table entries do not have .exe files set as InProc values, and that only LocalServer contexts have arguments and DefInProc values.
ICE43	Checks that non-advertised shortcuts are in components with HKCU registry keys as the key paths.
ICE44	Checks that dialog events in the ControlEvent table (NewDialog, SpawnDialog, SpawnWaitDialog) reference valid Dialogs in the Dialog table.
ICE45	Checks for reserved bits that are set.
ICE46	Checks for custom properties that only differ from defined properties by their case.
ICE47	Checks for features with more than 1600 components per feature..
ICE48	Checks for directories that are hard-coded to local paths.
ICE49	Checks for non-REG_SZ default values in the registry table.
ICE50	Checks that advertised shortcuts have correct icons and context menus.
ICE51	Checks that TTC/TTF fonts do not have titles, but that all other

	fonts do.
ICE52	Checks for non-public properties in the AppSearch table.
ICE53	Checks for registry entries that write private installer information or policy values.
ICE54	Checks for components using companion files as their key path file.
ICE55	Checks that LockPermission objects exist and have valid permissions.
ICE56	Validates that the directory structure of the .msi file has a single valid root.
ICE57	Validates that individual components do not mix per-machine and per-user data.
ICE58	Checks that your Media Table does not have more than 80 rows.
ICE59	Checks that advertised shortcuts belong to components that are installed by the target feature of the shortcut.
ICE60	Checks that if a file in the <a href="#">File Table</a> is not a font and has a version, then it also has a language.
ICE61	Checks the <a href="#">Upgrade Table</a> .
ICE62	Performs extensive checks on the <a href="#">IsolatedComponent Table</a> for data that may cause unexpected behavior.
ICE63	Checks for proper sequencing of the RemoveExistingProducts action.
ICE64	Checks that new directories in the user profile are removed in roaming scenarios.
ICE65	Checks that the <a href="#">Environment Table</a> does not have invalid prefix or append values.
ICE66	Uses the tables in the database to determine which schema your database should use.
ICE67	Checks that the target of a non-advertised shortcut belongs to the same component as the shortcut itself, or that the attributes of the target component ensure that it does not change installation locations.

ICE68	Checks that all custom action types needed for an installation are valid.
ICE69	Checks that all substrings of the form [\$componentkey] within a Formatted string do not cross-reference components.
ICE70	Verifies that integer values for registry entries are specified correctly.
ICE71	Verifies that the <a href="#">Media Table</a> contains an entry with DiskId equal to 1.
ICE72	Ensures that the only custom actions used in the <a href="#">AdvtExecuteSequence Table</a> are type 19, type 35, and type 51 custom actions.
ICE73	Verifies that your package does not reuse package codes or product codes of the Windows Installer SDK samples. For more information, see <a href="#">Package Codes</a> and <a href="#">Product Codes</a> .
ICE74	Verifies that the <b>FASTOEM</b> property has not been authored into the <a href="#">Property Table</a> .
ICE75	Verifies that all custom action types that use an installed file as their source are sequenced after the <a href="#">CostFinalize Action</a> .
ICE76	Verifies that no files in the <a href="#">BindImage Table</a> reference SFP (WFP) catalogs.
ICE77	Verifies that in-script custom actions are sequenced after the <a href="#">InstallInitialize Action</a> and before the <a href="#">InstallFinalize Action</a> .
ICE78	Verifies that the <a href="#">AdvtUISequence Table</a> either does not exist or is empty.
ICE79	Validates references to components and features entered in the database fields using the <a href="#">Condition</a> data type.
ICE80	Validates that <b>Template Summary</b> Property and <b>Page Count Summary</b> Property correctly specify the presence of 64-bit components or custom action scripts.
ICE81	Validates the <a href="#">MsiDigitalCertificate Table</a> , <a href="#">MsiDigitalSignature Table</a> and <a href="#">MsiPackageCertificate Table</a> .

ICE82	Validates the <a href="#">InstallExecuteSequence Table</a> .
ICE83	Validates the <a href="#">MsiAssembly Table</a> .
ICE84	Checks the sequence tables to verify that required <a href="#">Standard Actions</a> are not set with conditions.
ICE85	Validates that the SourceName column of the <a href="#">MoveFile Table</a> is a valid long file name.
ICE86	Issues a warning if the package uses the <b>AdminUser</b> property in database column of the <a href="#">Condition</a> type.
ICE87	Validates that the following properties have not been authored in the <a href="#">Property Table</a> .
ICE88	Validates the DirProperty column of <a href="#">IniFile Table</a> .
ICE89	Validates that the value in the Progid_Parent column in <a href="#">ProgId Table</a> is a valid foreign key into the Progid column in ProgId table.
ICE90	Posts a warning if it finds that a shortcut's directory has been specified as a public property.
ICE91	Posts a warning if a file, .ini file, or shortcut file is installed into a per-user profile directory that does not vary based on the <b>ALLUSERS</b> property.
ICE92	Verifies that a component without a Component Id GUID is not also specified as a permanent component. Verifies that no component has both the msidbComponentAttributesPermanent and msidbComponentAttributesUninstallOnSupersedence attributes.
ICE93	Issues a warning if a custom action uses the same name as a standard action.
ICE94	Issues a warning if there are any unadvertised shortcuts pointing to an assembly file in the global assembly cache.
ICE95	Checks the <a href="#">Control Table</a> and <a href="#">BBControl Table</a> to verify that the billboard controls fit onto all the billboards.
ICE96	Verifies that the <a href="#">PublishFeatures Action</a> and the <a href="#">PublishProduct Action</a> are entered in the <a href="#">AdvtExecuteSequence Table</a> .
ICE97	Verifies that two components do not isolate a shared component to

	the same directory.
<a href="#">ICE98</a>	Verifies the description field of the <a href="#">ODBCDataSource Table</a> for an ODBC data source.
<a href="#">ICE99</a>	Verifies that no property name entered in the <a href="#">Directory</a> table duplicates a name reserved for the public or private use of the Windows Installer.
<a href="#">ICE100</a>	Checks the authoring of the <a href="#">MsiEmbeddedUI</a> and <a href="#">MsiEmbeddedChainer</a> tables.
<a href="#">ICE101</a>	Checks that no value in the Feature column of the <a href="#">Feature</a> table exceeds a maximum length of 38 characters.
<a href="#">ICE102</a>	Validates the <a href="#">MsiServiceConfig</a> and <a href="#">MsiServiceConfigFailureActions</a> tables.
<a href="#">ICE103</a>	Validates the <a href="#">MsiPrint</a> and <a href="#">MsiLaunchApp</a> control events.
<a href="#">ICE104</a>	Verifies the <a href="#">MsiLockPermissionsEx</a> and <a href="#">LockPermissions</a> tables.
<a href="#">ICE105</a>	Validates that the package has been authored to be installed in a per-user context.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE01

ICE01 validates that the ICE mechanism is working. This ICE uses the [Time](#) property to get the time and returns either the system time or None.

## Result

ICE01 posts a message giving the time the installer called ICE01. This ICE should never return an error.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE02

ICE02 validates that certain references between the [Component](#), [File](#), and [Registry](#) tables are reciprocal. These references must be reciprocal for the installer to correctly determine the installation state of components.

The installer uses the KeyPath column of the Component table to detect the presence of the component listed in the Component column. The KeyPath column contains a key into the Registry or File tables. Both of these tables have a Component\_ column that contains a key back into the Component table pointing to the component that controls the registry entry or file. These references must be reciprocal.

## Result

ICE02 posts an error message if it finds a reference that should be reciprocal and is not.

## Example

ICE02 would post the following error message for a .msi file containing the database entries shown.

```
File: 'Red_File' cannot be the key file for Component: 'Blue'
```

### Component Table (partial)

Component	KeyPath
Red	Red_File
Blue	Red_File

### File Table (partial)

File Column	Component_
Red_File	Red

Blue_File	Blue
-----------	------

Component Blue references Red\_File, but Red\_File is not controlled by Component Blue and therefore cannot be the KeyPath file. If the installer were called to get the installation state of Blue it would incorrectly check whether Red\_File was installed. Changing the KeyPath field for Blue in the Component Table to Blue\_File fixes the error.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE03

ICE03 validates the data types and foreign keys based on the [\\_Validation table](#) and the database tables in the .msi file.

## Result

ICE03 posts the following messages for the validation errors.

<b>ICE03 error message</b>	<b>Description</b>
Duplicate Primary Key	The primary keys of a new row duplicates the primary keys of an existing row. The Nullable column of the <a href="#">_Validation table</a> shows the primary keys in the database.
Not A Nullable Column	A table column that is not specified as nullable in the Nullable column of the <a href="#">_Validation table</a> contains an entry that is Null.
Not A Valid Foreign Key	A column that is a foreign key into a second table contains an entry that does not exist in the primary key of the second table.
Value exceeds MaxValue	The numeric value of an entry in a database table exceeds the maximum limit specified for this field in the MaxValue column of the <a href="#">_Validation table</a> .
Value below MinValue	The numeric value of an entry in a database table is less than the minimum limit specified for this field in the MinValue column of the <a href="#">_Validation table</a> .
Value not a member of the set	The value of an entry in a database table is not a member of the acceptable set of values specified for this field in the Set column of the <a href="#">_Validation table</a> .
Invalid version string	See the <a href="#">Version</a> data type.
All UPPER case required	See the <a href="#">UpperCase</a> data type.

Invalid GUID string	See the <a href="#">GUID</a> data type.
Invalid file name/usage of wildcards	The database contains an invalid file name or an incorrect wildcard. See the <a href="#">WildcardFilename</a> data type.
Invalid identifier	See the <a href="#">Identifier</a> data type.
Invalid Language Id	The database contains an invalid numeric Language Identifier (LANGID). See the <a href="#">Language</a> data type. See Language Identifier Constants and Strings. For example, 1033 for the U.S. and 0 for language neutral.
Invalid Filename	See the <a href="#">Filename</a> data type.
Invalid full path	See the <a href="#">Path</a> , <a href="#">AnyPath</a> , and <a href="#">Paths</a> data types.
Bad conditional string	The database contains an invalid conditional string. This is a text string that must evaluate to TRUE or FALSE according to the <a href="#">Conditional Statement Syntax</a> . See the <a href="#">Condition</a> data type.
Invalid format string	See the <a href="#">Formatted</a> data type.
Invalid template string	See the <a href="#">Template</a> data type.
Invalid DefaultDir string	See the <a href="#">DefaultDir</a> data type.
Invalid registry path	See the <a href="#">RegPath</a> data type.
Bad CustomSource data	See the <a href="#">CustomSource</a> data type.
Invalid property string	See the <a href="#">Property</a> data type.
Missing data in _Validation table or old Database	There are columns in the database that are not listed in the Column column of the <a href="#">_Validation table</a> . The database and _Validation table do not match

Bad cabinet syntax/name	See the <a href="#">Cabinet</a> data type.
_Validation table: Invalid category string	This is an error in authoring the _Validation table. Validation does not recognize the category string used for this particular column in the _Validation table. See <a href="#">Column Data Types</a> and specify a valid category.
_Validation table: Data in KeyTable column is incorrect	The KeyTable column in the _Validation table references a table that does not exist in the database.
_Validation table: Value in MaxValue column < that in MinValue column	This is an error in authoring the <a href="#">_Validation table</a> . Min must always be less than or equal to Max.
Bad shortcut target	See the <a href="#">Shortcut</a> data type.
String overflow (greater than length permitted in column)	The string's length is greater than the column width specified by the column definition. Note that the installer does not internally limit the column width to the specified value. See <a href="#">Column Definition Format</a> .
Undefined error	Unknown error.
Column cannot be localized	Primary key columns cannot be localized.

## See Also

### [ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE04

ICE04 validates that the sequence number of every file in the [File table](#) is less than or equal to the largest sequence number in the LastSequence column of the [Media table](#).

Each record in the Media table represents a disk on the source media containing all the files with a sequence number less than or equal to the value in the LastSequence column and greater than the LastSequence value in the record of the preceding disk.

## Result

ICE04 posts an error message if there is a file with a sequence number greater than the largest LastSequence number for the source media.

## Example

ICE04 would report the following error for the example shown:

```
File: MyFile, Sequence: 210 Greater Than Max Allowed by Media
```

### File Table (partial)

File	Sequence
MyFile	210

### Media Table (partial)

DiskId	LastSequence
1	100

## See Also

## ICE Reference

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE05

ICE05 validates that certain tables contain required entries. This includes, but is not restricted to, checking the [Property table](#) for the required properties: [ProductName](#), [ProductLanguage](#), [ProductVersion](#), [ProductCode](#), and [Manufacturer](#).

## Result

ICE05 posts an error if a required entry is missing.

## Example

For the example shown, ICE05 would report that the entry: 'ProductVersion' is required in the 'Property' table.

[Property Table](#) (partial)

Property	Value
<a href="#">ProductName</a>	MyProduct

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE06

ICE06 checks every table to validate that all the columns listed in the [\\_Validation table](#) are present in the table. If a table does not exist, any [\\_Validation](#) entries for that table are ignored.

The purpose of ICE06 is to detect instances in which an author tries to use a new [\\_Validation](#) table that reflects a schema change with an old database that has not been updated. ICE06 also detects the reverse case of an old [\\_Validation](#) table being used with an altered database.

Note that the internal validation performed by [ICE03](#) catches the instance of a table column not defined in the [\\_Validation](#) table being listed in the columns catalog. The use of both ICE03 and ICE06 therefore ensures every column in the database is tested.

## Result

ICE06 posts an error when there is a table column defined in the [\\_Validation](#) table that is not listed in the [\\_Columns](#) table.

## Example

For the following example ICE06 posts the message

Column: Version of Table: ModuleSignature is not defined in database.

### [\\_Validation Table \(partial\)](#)

Table	Column
ModuleSignature	ModuleID
ModuleSignature	Version

### [\\_Columns Table \(partial\)](#)

Table	Number	Name
ModuleSignature	1	ModuleID

The Version column of the ModuleSignature table is not in the database or listed in the \_Columns table.

## **See Also**

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE07

ICE07 validates that installation package specifies that fonts be installed into the FontsFolder. If a font is installed to a folder other than the FontsFolder the installer creates a shortcut rather than actually installing the font.

The ICE07 custom action does the following for each font in the [Font table](#).

1. Finds the font file to which each font title belongs using the [Font table](#).
2. Queries the Component\_ column of the [File table](#) for the component that controls each file.
3. Queries the Directory\_ column of the [Component table](#) to obtain a key into the Directory table.
4. Resolves the [Directory table](#) to determine the name of the folder into which the installer is to install the font file
5. Posts an error if the font file is being installed into a folder other than the FontsFolder.

## Result

ICE07 posts an error if it finds that the database specifies that a font file be installed into a folder other than the FontsFolder.

## Example

ICE07 would post the following error message for the example shown.

```
'Tahoma' is a font and must be installed to the FontsFolder
```

## Font Table

File_	FontTitle
Myrtle	Tahoma

---

## File Table (partial)

File	Component_
Myrtle	Myrtle_Beach

## Component Table (partial)

Component	Directory_
Myrtle_Beach	SandBar

In this example, the font Tahoma maps to the font file Myrtle. The file Myrtle belongs to the component Myrtle\_Beach. Resolution of the Directory table shows that all the files belonging to Myrtle\_Beach are to be installed in the Sandbar folder. Because this is not the FontsFolder, ICE07 posts an error message.

Note that if the component Myrtle\_Beach really belongs in the Sandbar folder, and not the FontsFolder, then the font Tahoma may not belong in Myrtle\_Beach. A possible fix for the error would be to include Tahoma in another component that does get installed in the FontsFolder directory.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE08

ICE08 validates that the [Component table](#) contains no duplicate [GUIDs](#). Every component must have a unique GUID. No validation is done if the Component table does not exist.

## Result

ICE08 posts an error message if two or more entries in the ComponentId column of the Component table contain the same GUID.

## Example

For the following example, ICE08 would post a message stating that components Red and Green have duplicate GUIDs.

### [Component Table \(partial\)](#)

Component	ComponentId
Red	{0000000A-0003-0000-0000-624474736554}
Blue	{0000000A-0003-0000-0000-624474736354}
Green	{0000000A-0003-0000-0000-624474736554}

To fix this error, change either of the duplicate GUIDs.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE09

ICE09 validates that the permanent bit is set for every component marked for installation into the SystemFolder. ICE09 posts a warning because you should avoid installing non-permanent system components to the SystemFolder. To prevent this warning, set the Permanent bit in the Attributes column of the [Component table](#) for every component that is marked for installation into the SystemFolder. No validation is done if the database does not have a Component table.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE10

ICE10 validates that the advertise state of child features matches that of its parent feature.

A child feature may not disallow advertisement while its parent feature allows advertisement. The following combination of parent and child attributes is therefore invalid.

```
parent = msidbFeatureAttributesFavorAdvertise  
child = msidbFeatureAttributesDisallowAdvertise
```

This combination is invalid because it would turn off the parent whenever the parent was supposed to be advertised. However, the reverse is allowed. A child can be marked to favor advertisement while the parent is marked to disallow advertisement.

The ICE10 custom action determines the state of parent and child features from the Attributes column of the [Feature](#) table. Note that it is valid to set the state of a feature to 0 and have its parent or child set to favor or disallow advertisement.

## Result

ICE10 posts an error if the Attributes column of the Feature table contains a mismatch in the advertise state.

## Example

ICE10 posts the following error message for the example shown.

```
Conflicting states, one favors, one disallows. Child: Word c  
from Parent: Office.
```

Note for this example that Microsoft Excel and Microsoft Word are child features of Microsoft Office.

[Feature](#) table (partial)

Feature	Feature_Parent	Attributes
---------	----------------	------------

Office	Null	4
Excel	Office	4
Word	Office	8

In the example, Word is set to disallow advertisement, which conflicts with the allow advertisement state of its parent, Office.

In some cases ICE10 posts the following error:

```
Parent feature: 'Parent' not found for child feature: 'Child'
that for the child feature 'Child', the feature 'Parent' is
Feature table.
```

This refers to an invalid foreign key reference. The fix is to have 'Child' point to its correct parent feature, or add an entry for the parent feature 'Parent' to the [Feature](#) table.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE12

ICE12 queries the [CustomAction](#), [Directory](#), [AdminExecuteSequence](#), [AdminUISequence](#), [AdvtExecuteSequence](#), [InstallExecuteSequence](#), and [InstallUISequence](#) tables to validate the following:

- That the [CostFinalize action](#) occurs in any sequence table containing actions of the type [Custom Action Type 35](#) or [Custom Action Type 51](#).
- That every [Custom Action Type 35](#) comes after the [CostFinalize action](#). in the sequence tables.
- That every [Custom Action Type 51](#) that has a foreign key to the [Directory](#) table in the Source column of the CustomAction table comes before the [CostFinalize action](#) in the sequence tables.

Note that ICE12 does not validate the formatted text in the Target column of the CustomAction table.

## Result

ICE12 posts an error message if validation of the custom actions that set a directory property fails.

## Example

ICE12 would post three errors for the example shown.

- For CA1, Folder 'MyFolder' not found in Directory table
- For CA2, Sequence '80' comes before CostFinalize in the InstallExecuteSequence table. It must come after (CF@100)
- For CA3, Sequence '125' comes after CostFinalize in the InstallExecuteSequence table. It must come before (CF@100)

## CustomAction Table (partial)

Action	Type	Source
--------	------	--------

CA1	35	MyFolder
CA2	35	WindowsFolder
CA3	51	WindowsFolder

## Directory Table

Directory	Directory_Parent	DefaultDir
TARGETDIR		SourceDir
WindowsFolder	TARGETDIR	WindowsFolder

## InstallExecuteSequence Table (partial)

Action	Sequence
CostFinalize	100
CA2	80
CA3	125

To fix the error for CA1 change its entry in its Source column in the CustomAction table to an existing entry in the Directory table or add MyFolder to the Directory table.

To fix the error for CA2, change its sequence in the InstallExecuteSequence table such that it comes after the CostFinalize action.

To fix the error for CA3, change its sequence in the InstallExecuteSequence table such that it comes before the CostFinalize action.

## See Also

## ICE Reference

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE13

ICE13 validates that dialogs in sequence tables appear in the [AdminUISequence](#), or [InstallUISequence](#) tables. Dialogs must not be listed in the [InstallExecuteSequence](#), [AdminExecuteSequence](#), or [AdvtExecuteSequence](#) tables.

## Result

ICE13 posts an error message if a dialog appears in an execute sequence table.

## Example

For the following example, ICE13 posts an error message stating that the 'WelcomeDialog' cannot be listed in the 'InstallExecuteSequence' table.

### [InstallExecuteSequence Table \(partial\)](#)

Action
InstallFinalize
CostFinalize
WelcomeDialog

### [InstallUISequence Table \(partial\)](#)

Action
CostFinalize
CostInitialize
BrowseDialog

### [Dialog Table \(partial\)](#)

Dialog
--------

## Dialog

BrowseDialog

WelcomeDialog

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE14

ICE14 validates the following for features:

- That root parent features do not have the `msidbFeatureAttributesFollowParent` bit set in the `Attributes` column of the [Feature table](#). A root feature does not have a parent.
- That no feature has the same entry in the `Feature` and `Feature_Parent` columns of the [Feature table](#). No feature can be its own parent.

## Result

ICE14 posts an error message if it finds a root feature with the `msidbFeatureAttributesFollowParent` bit set or a feature with identical entries in the `Feature` and `Feature_Parent` columns of the `Feature` table.

## Example

ICE14 would return the following errors for the following example:

- The feature `Sport` has the same value in the `Feature` and `Feature_Parent` columns of the `File` table.
- The root feature `Sport` has the `msidbFeatureAttributesFollowParent` bit set.

In the feature tree of this example, `Sport` is the root feature and a parent of the `Swimming` and `Football` features. `Freestyle` is a child feature of `Swimming`. `TouchFootball` is a child feature of `Football`.

[Feature Table](#) (partial)

Feature	Attributes	Feature_Parent
Sport	4	Sport
Swimming	1	Sport

Football	4	Sport
Freestyle	1	Swimming
TouchFootball	4	Football

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE15

ICE15 validates that content type and extension references in the [MIME](#) and [Extension](#) tables are reciprocal. The MIME table must reference a content type to an extension that the Extension table references back to the same content type.

Multiple extensions can reference the same MIME type, as long as the MIME type references back to one of the extensions. Multiple MIME types can reference the same extension, as long as the extension references back to one of the MIME types.

Note that whenever a MIME references an extension, that extension cannot have the MIME\_ column in the Extension table set to Null.

## Result

ICE15 posts an error if the content type and extension references are not reciprocal.

## Example

ICE15 posts two error messages for the example shown:

- The content type test/x-flaps in the MIME table references the extension tst, but the extension tst in the Extension table references flaps/x-flaps. This is not reciprocal.
- The content type flaps/x-flaps references the extension flp, but that extension has a Null entry in the MIME\_ column of the Extension table.

## MIME Table (partial)

ContentType	Extension_
test/x-test	tst
flaps/x-flaps	flp

## Extension Table (partial)

Extension	MIME_
tst	flaps/x-flaps
flp	Null

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE16

ICE16 validates that the value of the **ProductName** property in the [Property table](#) is no greater than 63 characters in length. No limit exists on the length of the registry key for DisplayName.

## Result

ICE16 posts an error message if the ProductName set in the Property table is longer than 63 characters.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE17

ICE17 checks for the situations shown in the example at the end of this topic.

## Result

ICE17 displays an error or warning message for each of the situations in the example. Samples of such messages are shown in the following table.

<b>ICE17 error or warning</b>	<b>Description</b>
PushButton: Button1 of Dialog: MyDialog does not have an event defined in the ControlEvent table. <b>Error</b>	There is a <a href="#">Pushbutton control</a> that is not listed in the <a href="#">ControlEvent table</a> . If ICE17 returns this error on a PushButton for which the <b>Enable Control</b> attribute or the <b>Visible Control</b> attribute is not set in the Attributes column of the <a href="#">Control table</a> , check whether the control also has an entry in the <a href="#">ControlCondition table</a> . The control can unexpectedly become enabled, or visible, if the value in the Condition column changes to True, Enable, or Show.
Bitmap: Bitmap1 of Control: Bitmap1 of Dialog: MyDialog is not in the Binary table. <b>Error</b>	There is a <a href="#">Bitmap control</a> or <a href="#">Icon control</a> , but the corresponding bitmap or icon is not listed in the <a href="#">Binary table</a> . Add the bitmap or icon to the Binary table.
RadioButtonGroup: RadioButton1 of Control: RadioButton1 of Dialog: MyDialog is not in the RadioButton table.	There is a <a href="#">RadioButtonGroup control</a> with values in the Property column and the Attribute column of the Control table; the <a href="#">Indirect</a> bit is not set in the Attributes column. ICE17 posts a warning because the installer uses the property's value as a foreign key into the RadioButton table, but the value is missing from the primary key of that table.

<p><b>Warning</b></p>	<p>If the <a href="#">Indirect</a> bit is set, then the property listed for the control is not used as the property; instead, it is used as the name of the property that is actually used.</p> <p>This warning can be ignored if the control is created at runtime. For example, the <a href="#">ListBox control</a> for on the <a href="#">FilesInUse Dialog</a> is only created at runtime if there are files in use during the installation.</p>
<p>ListBox: ListBox1 of Control: ListBox1 of Dialog: MyDialog is not in the ListBox table.</p> <p><b>Warning</b></p>	<p>There is a <a href="#">ListBox control</a> with a value in the Property column of the <a href="#">Control table</a> and for which the <a href="#">Indirect</a> bit is not set in the Attributes column. ICE17 posts a warning because the installer uses the property's value as a foreign key into the <a href="#">ListBox table</a>, but the value is missing from the primary key of that table.</p> <p>If the <a href="#">Indirect</a> bit is set, the control changes the value of a property having a name that is the value of the property associated with this control.</p> <p>This warning can be ignored if the control is created at runtime. For example, the <a href="#">ListBox control</a> for on the <a href="#">FilesInUse Dialog</a> is only created at runtime if there are files in use during the installation.</p>
<p>ComboBox: ComboBox1 of Control: ComboBox1 of Dialog: ByDialog is not in the ComboBox table</p> <p><b>Warning</b></p>	<p>There is a <a href="#">ComboBox control</a> with a value in the Property column of the <a href="#">Control table</a> and for which the <a href="#">Indirect</a> bit is not set in the Attributes column. ICE17 posts a warning because the installer uses the property's value as a foreign key into the <a href="#">ComboBox table</a>, but the value is missing from the primary key of that table.</p> <p>If the <a href="#">Indirect</a> bit is set, the control changes the value of a property having a name that is the value of the property associated with this control.</p> <p>This warning can be ignored if the control is created at runtime. For example, the <a href="#">ListBox control</a> for on the <a href="#">FilesInUse Dialog</a> is only created at runtime if there are files in use during the installation.</p>
<p>ListView: ListView1 of</p>	<p>There is a <a href="#">ListView control</a> with a value in the Property column of the <a href="#">Control table</a> and for which the <a href="#">Indirect</a></p>

<p>Control: ListView1 of Dialog: MyDialog is not in the ListView table. <b>Warning</b></p>	<p>bit is not set in the Attributes column. ICE17 posts a warning because the installer uses the property's value as a foreign key into the <a href="#">ListView table</a>, but the value is missing from the primary key of that table. If the <a href="#">Indirect</a> bit is set, the control changes the value of a property having a name that is the value of the property associated with this control.  This warning can be ignored if the control is created at runtime. For example, the <a href="#">ListBox control</a> for on the <a href="#">FilesInUse Dialog</a> is only created at runtime if there are files in use during the installation.</p>
<p>Bitmap: 'Bitmap2' for Control: 'Button2' of Dialog: 'MyDialog' not found in Binary table <b>Error</b></p>	<p>There is a <a href="#">Pushbutton Control</a> or <a href="#">Checkbox Control</a> for which the Text column of the <a href="#">Control table</a> does not contain a foreign key into the record of the <a href="#">Binary table</a> containing the bitmap or icon.</p>
<p>Bitmap: 'Bitmap3' for Control: 'RadioButton2' of Dialog: 'MyDialog' not found in Binary table or Icon: 'Icon1' for Control: 'RadioButton3' of Dialog: 'MyDialog' not found in Binary table <b>Error</b></p>	<p>There is a <a href="#">RadioButtonGroup</a> control for which the Text column of the <a href="#">RadioButton table</a> does not contain a foreign key into the record of the <a href="#">Binary table</a> containing the bitmap or icon.</p>
<p>Picture control: 'Button3' of Dialog:</p>	<p>There is a <a href="#">PushButton</a>, <a href="#">CheckBox</a>, or <a href="#">RadioButtonGroup</a> control with both the <a href="#">Icon</a> bit or</p>

'MyDialog' has both the Icon and Bitmap attributes set  
**Error**

[Bitmap](#) bit set in the Attributes column of the [Control table](#). You cannot set both attributes together.

## Example

### Control Table (partial)

Dialog_	Control	Type	Attributes	Property	Text
MyDialog	Button1	PushButton	0		OK
MyDialog	Bitmap1	Bitmap	0		Bitm
MyDialog	RadioButton1	RadioButtonGroup	0	RadioButton1	
MyDialog	Listbox1	Listbox	0	Listbox1	
MyDialog	ComboBox1	ComboBox	0	ComboBox1	
MyDialog	Listview1	Listview	0	Listview1	
MyDialog	Button2	Pushbutton	262144		Bitm
MyDialog	RadioButton2	RadioButtonGroup	262144		Prop
MyDialog	RadioButton3	RadioButtonGroup	524288		Prop
MyDialog	Button3	Pushbutton	786432		Amt

### RadioButton Table (partial)

Property_	Order	Text
Property2	1	Bitmap3
Property3	2	Icon1

The following tables are empty:

- [ControlEvents Table](#)
- [ListBox Table](#)
- [ComboBox Table](#)
- [ListView Table](#)
- [Binary Table](#)

## **See Also**

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE18

ICE18 validates that any empty directories used as a key path for a component are listed in the [CreateFolder table](#).

If the KeyPath column of the [Component table](#) is Null, this means that directory listed in the Directory\_ column is the key path for that component. Because folders created by the installer are deleted when they become empty, this folder must be listed in the [CreateFolder table](#) to prevent the installer from attempting to install every time.

Do not make the SystemFolder directory the key path of a component. Because this folder is present on every operating system, the installer always detects the key path whether or not the component is present. In this case, the key path should be a file, registry entry, or ODBC data source.

When performing a validation ICE18 first checks whether the following are all true:

- The KeyPath column of the [Component table](#) contains a Null value.
- That there are no files listed for the component in the [File table](#).
- That there are no files for the component listed in the [RemoveFile table](#) and that the value in the DirProperty is the same as the Directory\_ column of the [Component table](#).
- That there are no files for the component listed in the [DuplicateFile table](#) and that the value in the DestFolder is the same as the Directory\_ column of the [Component table](#).
- That there are no files for the component listed in the [MoveFile table](#) and that the value in the DestFolder is the same as the Directory\_ column of the [Component table](#).

If these are all true then ICE18 validates the following:

- That the Component\_ column of the [CreateFolder table](#) has the same value as the Component column of the [Component table](#).

- That the Directory\_ column of the CreateFolder table has the same value as the Directory\_ column of the [Component table](#).

## Result

ICE18 posts an error message if the installation package specifies a directory as the key path for component that is not listed in the [CreateFolder table](#).

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE19

ICE19 validates that advertised components reference a file in the KeyPath column of the [Component table](#) and that an advertised shortcut references a directory in this column.

ICE19 validates that advertised components or shortcuts have a ComponentId. Components in the [PublishComponent table](#), which are not advertised in another table, are only checked to see whether they have a ComponentId.

## Result

ICE19 posts an error message if the KeyPath column of the Component table does not reference a file in the case of an advertised component or a directory in the case of an advertised shortcut. ICE19 posts an error message if any advertised components or shortcuts do not have a ComponentId.

## Example

ICE19 posts the following error messages for the example shown:

- Extension flp references the component Comp1 which does not have a ComponentId specified in the [Component table](#).
- Extension exe references the component Comp4 which references a directory as its KeyPath. The KeyPath is Null in the Component table.
- Shortcut Shortcut2 references the component Comp3 which references a Registry entry as the key path. The value of the Attributes column in the Component table is 4.

## Component Table (partial)

Component	ComponentId	Attributes	KeyPath
Comp1	Null	0	File1

Comp2	{00000002-0003-0000-0000-624474736554}	0	File2
Comp3	{00000003-0003-0000-0000-624474736554}	4	Reg3
Comp4	{00000004-0003-0000-0000-624474736554}	0	Null

### Extension Table (partial)

Extension	Component_
flp	Comp1
tst	Comp2
exe	Comp4

### Shortcut Table (partial)

Shortcut	Component_	Feature_
Shortcut1	Comp4	ProductFeature
Shortcut2	Comp3	ProductFeature

### Feature Table (partial)

Feature
ProductFeature

**Note** If the extension flp and exe both reference the same component, the EXE or COM server that opens them must be the same. This EXE is normally the KeyPath for the Component. For OFFICE, the extensions

doc and xls cannot reference the same component because the same EXE does not open both extensions. You need winword.exe to open doc extensions and you need excel.exe to open xls extensions.

## **See Also**

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE20

ICE20 validates the standard dialogs and dialog requirements of an installation package having an authored user interface. Installation packages with an authored user interface have a [Dialog table](#) and the **LIMITUI** property is not set.

ICE20 validates the following tables and requirements.

Standard dialog	Dialog requirements
<a href="#">FilesInUse Dialog</a>	<p>A <a href="#">ListBox table</a>.</p> <p>A <a href="#">ListBox control</a> with the Property column of the <a href="#">Control table</a> set to FileInUseProcess.</p> <p>A <a href="#">PushButton control</a> with EndDialog entered into the Event column of the <a href="#">ControlEvent table</a> and Ignore in the Argument column.</p> <p>A <a href="#">PushButton control</a> with EndDialog entered into the Event column of the <a href="#">ControlEvent table</a> and Exit in the Argument column.</p> <p>A <a href="#">PushButton control</a> with EndDialog entered into the Event column of the <a href="#">ControlEvent table</a> and Retry in the Argument column.</p>
<a href="#">Error Dialog</a>	<p>The <a href="#">Error Dialog Style bit</a> must be set to specify the dialog, with any name, is an Error Dialog.</p> <p>A Text control named ErrorText.</p> <p>ErrorText in the Control_First column of the <a href="#">Dialog table</a>.</p> <p>A Pushbutton control, named A, with EndDialog in the Event column of the <a href="#">ControlEvent table</a> and ErrorAbort in the Argument column.</p> <p>A Pushbutton control, named C, with EndDialog in the Event column of the <a href="#">ControlEvent table</a> and ErrorCancel in the Argument column.</p> <p>A Pushbutton control, named I, with EndDialog in the Event</p>

	<p>column of the <a href="#">ControlEvent table</a> and ErrorIgnore in the Argument column.</p> <p>A Pushbutton control, named N, with EndDialog in the Event column of the <a href="#">ControlEvent table</a> and ErrorNo in the Argument column.</p> <p>A Pushbutton control, named O, with EndDialog in the Event column of the <a href="#">ControlEvent table</a> and ErrorOk in the Argument column.</p> <p>A Pushbutton control, named R, with EndDialog in the Event column of the <a href="#">ControlEvent table</a> and ErrorRetry in the Argument column.</p> <p>A Pushbutton control, named Y, with EndDialog in the Event column of the <a href="#">ControlEvent table</a> and ErrorYes in the Argument column.</p> <p>The width, height, and Y coordinates should be the same for all of the buttons. The X coordinates are determined by the installer.</p> <p>If a control named ErrorIcon exists, it must have Icon in the Type column of the <a href="#">Control table</a>.</p>
<a href="#">Exit Dialog</a>	There must be either a Dialog or Custom Action in the <a href="#">AdminUISequence table</a> and <a href="#">InstallUISequence table</a> with a value -1 in the Sequence Column.
<a href="#">UserExit Dialog</a>	There must be either a Dialog or Custom Action in the <a href="#">AdminUISequence table</a> and <a href="#">InstallUISequence table</a> with a value -2 in the Sequence Column.
<a href="#">FatalError Dialog</a>	There must be either a Dialog or Custom Action in the <a href="#">AdminUISequence table</a> and <a href="#">InstallUISequence table</a> with a value -3 in the Sequence Column.

## Result

ICE20 posts an error message if it cannot validate that the standard dialogs and requirements are present in the installation package.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE21

ICE21 validates that every component in the [Component table](#) belongs to a feature. It uses the [FeatureComponents table](#) to check for this mapping.

## Result

ICE21 posts an error message if the installation package contains a component that does not belong to a feature.

## Example

For the following example, ICE21 posts an error message stating that the component Comp1 does not map to any feature.

### Component Table (partial)

Component
Comp1
Comp2

### FeatureComponents Table (partial)

Feature	Component
Feature1	Comp2

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE22

ICE22 uses the [FeatureComponents table](#) to validate the mapping of features and components referenced in the [PublishComponent table](#).

## Result

ICE22 posts an error message if the features and components are mapped incorrectly in the [PublishComponent table](#).

## Example

For the following example ICE22 posts an error that {00000003-0004-0000-0000-624474732465} does not have the correct mapping for the Feature\_ and Component\_ columns.

### [PublishComponent Table \(partial\)](#)

ComponentId	Feature_	Component_
{00000002-0003-0000-0000-624474736554}	Feat1	Comp1
{00000003-0004-0000-0000-624474732465}	Feat1	Comp2

### [FeatureComponents Table \(partial\)](#)

Feature_	Component_
Feat1	Comp1

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE23

ICE23 validates the control tab order for each dialog box.

ICE23 validates the following in the [Dialog table](#) and [Control table](#):

- That every record in the Dialog table specifies a control in the Control\_First column that exists in the dialog box specified by the Dialog column.
- That every record in the Control table specifies a control in the Control\_Next column that is on the same dialog as the control listed in the Control column, or Control\_Next contains the Null value.
- That following the Control\_Next entries from control to control in the Control table makes a single, closed, loop that comes back to the initial control. Not every control needs to be in the loop, but the loop must pass through every control that has an entry in the Control\_Next column.

## Result

ICE23 posts an error message if the tab order of controls does not form a single closed loop in the dialog box.

## Example

ICE23 would post the following error messages for the example shown.

- Dialog1 has no Control\_First.
- Control\_First of dialog Dialog2 refers to nonexistent control ControlX.
- Dialog3 has dead-end tab order at control ControlB.
- Dialog4 has malformed tab order at control ControlC
- Dialog5 has malformed tab order at control ControlC.
- Control\_Next of control Dialog6.ControlC links to unknown control.

## Dialog Table (partial)

Dialog	Control_First
Dialog1	
Dialog2	ControlX
Dialog3	ControlA
Dialog4	ControlA
Dialog5	ControlA

## Control Table (partial)

Dialog	Control	Control_Next
Dialog1	ControlA	
Dialog1	ControlB	ControlA
Dialog2	ControlA	ControlB
Dialog2	ControlB	ControlA
Dialog3	ControlA	ControlB
Dialog3	ControlB	
Dialog4	ControlA	ControlB
Dialog4	ControlB	ControlC
Dialog4	ControlC	ControlB
Dialog5	ControlA	ControlB
Dialog5	ControlB	ControlC
Dialog5	ControlC	ControlA
Dialog5	ControlD	ControlA
Dialog6	ControlA	ControlB
Dialog6	ControlB	ControlC
Dialog6	ControlC	ControlX

Dialog6	ControlD	ControlA
---------	----------	----------

To fix these errors note the following in the above tables and make the indicated changes.

Not every row in the Dialog table has a control specified in the Control\_First column. Change the Control\_First column of the Dialog1 record in the Dialog table to a control that exists in Dialog1.

Not every row in the Dialog table has a control specified in the Control\_First column that exists on the dialog box. Change the Control\_First column of the Dialog2 to a control that exists in Dialog2.

Following the Control\_Next entries in the Control table from control to control does not make a closed loop in every case. Change the Control\_Next column for ControlB in Dialog3 to ControlA.

Following the Control\_Next entries in the Control table from control to control does not lead back to the initial control in every case. Change the Control\_Next column for ControlC in Dialog4 to refer to ControlA.

Following the Control\_Next entries in the Control table from control to control does not pass through every control in the dialog box having an entry in the Control\_Next column. Change the Control\_Next column for ControlC in Dialog5 to ControlD.

Control\_Next does not refer to a valid control that is in the same dialog as the control listed in the Control column. Change the Control\_Next column for ControlC in Dialog6 to refer to ControlD.

## See Also

### [ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE24

ICE24 validates the following properties in the [Property table](#):

- That the [ProductCode](#) Property is a valid [GUID](#) data type.
- That the [ProductVersion](#) Property is a valid product version.
- That the [ProductLanguage](#) Property is a valid [Language](#) data type.

## Result

ICE24 posts an error message if any of these properties are not in the form of a valid data type.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE25

ICE25 validates that a .msi file satisfies all of its internal [merge module dependencies](#) and exclusions. ICE validates the following:

- That all merge module dependencies indicated in the .msi file's [ModuleDependency table](#) are satisfied by at least one merge module listed in the [ModuleSignature table](#).
- That none of the excluded merge modules in the [ModuleExclusion table](#) are incompatible with the merge modules listed in the [ModuleSignature table](#).

## Result

ICE25 posts an error message if .msi file has previously been merged with an incompatible merge module or if it has not been merged with a necessary merge module.

## Example

ICE25 posts the following errors for the example shown.

```
Dependency failure: Need ModuleX@0 v2.0  
Module ModuleB@1033 v1.0 is excluded.
```

## ModuleSignature Table

ModuleID	Language	Version
ModuleA	0	1.0
ModuleB	1033	1.0

## ModuleDependency Table

ModuleID	ModuleLanguage	RequiredID	RequiredLanguage	RequiredVersion
ModuleA	0	ModuleX	0	2.0

## ModuleExclusion Table

ModuleID	ModuleLanguage	ExcludedID	ExcludedLanguage	ExcludedM
ModuleA	0	ModuleB	1033	

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE26

ICE26 validates that each of the following [sequence tables](#) contain the actions that are required by the table and does not contain any actions disallowed in the table:

- [AdminUISequence table](#)
- [AdminExecuteSequence table](#)
- [InstallUISequence table](#)
- [InstallExecuteSequence table](#)

## Result

ICE26 posts an error message if the installation package has a sequence table that lacks a required action or that contains an action that is disallowed for the table.

## Example

ICE26 error	Description
Action: 'Action1' is required in the InstallExecuteSequence table.	A required action is missing from the indicated sequence table. See the template.msi or the suggested sequence tables in <a href="#">Using a Sequence Table</a> .
Action: 'Action2' is prohibited in the InstallExecuteSequence table.	This action cannot be in the indicated sequence table. Remove this action from the sequence table.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE27

ICE27 validates the [sequence tables](#) of an installation package for valid actions, action sequence restrictions, and organization in Search, Costing, Selection, and Execution sections.

The ICE27 custom action validates the following:

- That the actions listed in the Action column of the sequence tables are a [standard actions](#), a custom action listed in the [CustomAction table](#), or a dialog box listed in the [Dialog table](#).
- That actions subject to sequencing restrictions are in the correct relative order to each other in the action sequence. Sequencing restrictions result when one action is dependent on another.
- That actions restricted to a particular section of the sequence are located where they belong. ICE27 validates the following organization of the sequence tables. Note that not every sequence table has every section. See the suggested sequence tables in [Using a Sequence Table](#).

Sequence table section	Range in action sequence	Actions belonging to section
Search	{start} to <a href="#">CostInitialize</a>	Actions that search for existing applications. <a href="#">AppSearch</a> <a href="#">CCPSearch</a>
Costing	<a href="#">CostInitialize</a> to <a href="#">CostFinalize</a> action	Actions that do <a href="#">file costing</a> . <a href="#">CostInitialize</a> <a href="#">FileCost</a> <a href="#">CostFinalize</a>
Selection	<a href="#">CostFinalize</a> to	Actions that set folders or feature states. <a href="#">SetODBCFolders</a> action

	<a href="#">InstallValidate</a>	
Execution	<a href="#">InstallValidate</a> to <a href="#">InstallFinalize</a>	Script actions, such as Registration, Publication, Installation (where you copy files). Note the <a href="#">InstallFinalize action</a> must be in the table if and only if there are actions in the Execution section.
PostExecute	<a href="#">InstallFinalize</a> to {end}	<a href="#">RemoveExistingProducts</a>

ICE27 validates the following tables:

- [AdvtExecuteSequence](#)
- [AdminUISequence](#)
- [AdminExecuteSequence](#)
- [InstallUISequence](#)
- [InstallExecuteSequence](#)

## Result

ICE27 posts an error message if there are sequence tables in the package with invalid action sequencing or organization.

## Example

<b>ICE27 error</b>	<b>Description</b>
Unknown action: 'Action1' of InstallExecuteSequence table. Not a standard action and not found in CustomAction or Dialog tables	There is an action listed in the sequence table indicated that is not a <a href="#">standard actions</a> , a custom action listed in the <a href="#">CustomAction table</a> , or a dialog box listed in the <a href="#">Dialog table</a> .
'Action2' in InstallExecute table	There is an action in a sequence table that

<p>in wrong place. Current: Search, Correct: Costing</p>	<p>is incorrectly placed with respect to the sequence number in the Sequence column. "Current" indicates the current placement of the action in the Search, Costing, Selection, or Execution sections of the indicated sequence table.</p> <p>"Correct" indicates in which section the action belongs.</p> <p>To fix this error, change the sequence number of the action to inside the correct section. Note that some action can be located in more than one section.</p>
<p>'InstallFinalize' Action in InstallExecuteSequence table can only be called when script operations exist to be executed</p>	<p>There is an <a href="#">InstallFinalize action</a> in a sequence table that does not contain any script operations in the Execution section of the table.</p> <p>Add actions to the Execution section or remove the InstallFinalize action from the table.</p>
<p>InstallFinalize must be called in InstallExecuteSequence table as script operations exist to be executed</p>	<p>There is a sequence table containing actions in the Execution section that does not include the <a href="#">InstallFinalize action</a>.</p> <p>Add the InstallFinalize action to this sequence table and give it the greatest sequence number to place it last in the action sequence.</p>
<p>Action: 'Action3' in InstallExecuteSequence table must come before the 'Action5' action. Current seq#: 1200. Dependent seq#: 1100</p>	<p>There is an action in the indicated sequence table that is sequenced after a dependent action.</p> <p>Change the sequence number on the dependent action so that it comes before the action.</p>
<p>Action: 'Action4' in InstallExecuteSequence table</p>	<p>There is an action in the indicated sequence table that is sequenced before an</p>

must come after the 'Action6' action.

action that it is dependent upon.  
Change the sequence number on the action so that it comes after its dependent action.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE28

ICE28 is commonly used to validate that the ForceReboot action is placed before or after, and never within, a specific group of actions in the action sequence tables. See the [ForceReboot action](#) topic to determine the actions that comprise this group.

ICE28 validates actions in the following sequence tables:

[AdminUISequence table](#)

[AdminExecuteSequence table](#)

[InstallUISequence table](#)

[InstallExecuteSequence table](#)

## Result

ICE28 posts an error message if ForceReboot is sequenced within the specified action group.

## Example

For the example shown, ICE28 posts the following error message:

```
Action: 'ForceReboot' of table InstallExecuteSequence is not
```

### InstallExecuteSequence Table

Action	Sequence
ForceReboot	10
RegisterMIMEInfo	5
RegisterProgIdInfo	15

The sequence number of 10 given to ForceReboot breaks generates the error, because it comes between the sequence numbers of RegisterMIMEInfo and RegisterProgIdInfo.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE29

ICE29 validates that truncated stream names remain unique. Any table having a Binary or Object column is validated. See the [Binary](#) column data type.

Handling of streams by the Win32 OLE structured storage implementation limits stream names. See [OLE Limitations on Streams](#). The installer can compress stream names up to 62 characters in length. Names longer than this are truncated.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE30

ICE30 validates that the installation of components containing the same file never installs the file more than once in the same directory.

ICE30 goes to every component in the [Component table](#) and then determines the component's target directory from the [Directory table](#). It then checks to see which of these components install to the same target directory. Finally, it uses the [File table](#) to verify that none of the files in these components have the same name.

ICE30 checks both long file names (LFN) and short file names (SFN).

ICE30 does not evaluate properties in the resolution of directories because these properties can change at runtime and alter the directory resolution scheme. This means ICE30 can detect file collisions due to directories with the same property in their paths, but does not detect collisions resulting from two properties having the same value.

## Result

ICE30 posts an error message for each pair of components that installs the same file to the same directory.

## Example

The example shown returns each of the following errors twice.

ICE30 error or warning	Description
ERROR: The target file 'README.1st' is installed in 'TARGETDIR\PRODUCT' by two different components on an SFN system: 'Component1' and 'Component2'. This breaks component reference counting.	Component1 and Component2 both have a file named 'README.1st'. When using short file names, the installer installs both Dir1 and Dir2 to the same directory, TARGETDIR\PRODUCT. ICE30 generate two errors, one for each file. In an

	authoring environment that displays error locations, the first error is at one file's entry in the <a href="#">File Table</a> , and the second at the location of the other file.
ERROR: Installation of a conditionalized component would cause the target file 'README.1st' to be installed in 'TARGETDIR\COMMON TOOLS' by two different components on an LFN system: 'Component3' and 'Component4'. This would break component reference counting.	Component4 has an entry in the Condition column of the <a href="#">Component table</a> and Component3 does not. If <b>VersionNT</b> is True, Component4 is installed, and there a collision with the Readme.1st always installed by Component3.  ICE30 generates 4 errors, one pair for SFN, one for LFN.
WARNING: The target file 'README.1st' might be installed in 'TARGETDIR\COMMON TOOLS' by two different conditionalized components on an SFN system: 'Component4' and 'Component5'. If the conditions are not mutually exclusive, this will break the component reference counting system.	Because Component4 and Component5 both have entries in the Condition column of the <a href="#">Component table</a> this file collision may not occur. ICE30 only posts a warning because the conditions must be determined at the time of the installation.  ICE30 generates 4 warnings, one pair for SFN, one for LFN.

### Component Table (partial)

Component	Directory	Condition

Component1	Dir1	
Component2	Dir2	
Component3	Dir3	
Component4	Dir3	VersionNT
Component5	Dir3	Version9X

## Directory Table

Directory	Parent_Directory	DefaultDir
SOURCEDIR		TARGETDIR
Dir1	SOURCEDIR	Product Component1 Product:.
Dir2	SOURCEDIR	Product:.
Dir3	SOURCEDIR	Common Common Tools:

## File Table (partial)

File	Component_	FileName
File1	Component1	README.1st
File2	Component2	README.1st
File3	Component3	README.1st
File4	Component4	README.1st
File5	Component5	README.1st

## See Also

### [ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE31

ICE31 validates any predefined font styles used in [controls](#) that display text. It also validates that the **DefaultUIFont** property refers to a valid font style.

Controls can have a predefined font style as described in [Adding Controls and Text](#). To set the font and font style of a text string, prefix the string of displayed characters with `{style}` or `&style`. Where style is an identifier listed in the TextStyle column of the [TextStyle table](#). If neither of these are present, but the **DefaultUIFont** property is defined as a valid text style, that font will be used.

ICE31 checks the Text column for each control in the [Control Table](#) to verifies that a valid entry exist in the TextStyle table.

ICE31 ignores the [ScrollableText Control](#).

## Results

ICE31 posts an error message for undefined styles, style names that are too long, a missing TextStyle table, and style tags with no closing brace.

ICE31 posts a warning if the style tag is not at the beginning of the line, or if a control has multiple style tags.

## Example

ICE31 posts the following errors for the example shown:

- Control DialogB.Control1 uses undefined TextStyle BadStyle.
- Control DialogB.Control2 uses undefined TextStyle BadStyle.
- Control DialogB.Control6 is missing closing brace in text style.
- Control DialogB.Control3 specifies a text style that is too long to be valid.

ICE31 posts the following warning for the example shown:

- Text Style tag in DialogB.Control4 has no effect. Do you really want it

to appear as text?

### Control Table (partial)

Dialog	Control	Text
DialogA	Control0	{\OKStyle}This is the text to display.
DialogA	Control1	{&OKStyle}This is the text to display.
DialogB	Control1	{&BadStyle}This is the text to display.
DialogB	Control2	{\BadStyle}This is the text to display.
DialogB	Control3	{&Style that is over 72 chars and therefore cannot possibly be a style even if somehow you did manage to get it in the TextStyle table}This is the text to display.
DialogB	Control4	Warning {\OKStyle}This is the text to display.
DialogB	Control5	{\OKStyle}{&OKStyle}This is the text to display.
DialogB	Control6	{\OKStyle This is the text to display.

### TextStyle table (partial)

TextStyle
OkStyle

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE32

ICE32 validates that keys and foreign keys in the .msi file are of the same size and column definition types. This ICE custom action makes the comparison using the [\\_Validation table](#) and using the definition types that are returned by [MsiViewGetColumnInfo](#). For more information, see [Column Definition Format](#).

## Result

ICE32 posts errors if the .msi file contains any foreign keys to keys of a different column length or column data type.

## Example

ICE32 posts two errors for the example shown:

- There is a foreign key and key defined that differ in size.
- There is a foreign key and key defined that differ in their definition type.

### [\\_Validation Table \(partial\)](#)

Table	Column	KeyTable	KeyColumn
File	Version	File	1
Flap	Column8	Flap	1

### Column Definitions (partial)

Table	Column	Type	Size
File	File	s	72
File	Version	S	32
Flap	Column1	i	2
Flap	Column8	S	32

The Version column of the File table can be a foreign key to another file in the File table. This occurs with companion files. However, the Version column only allows a string length 32, whereas the File column allows a string length 72. To fix this error change the string lengths to match.

There is a foreign key and key defined that differ in their definition types. Column8 of the Flap Table is listed as a foreign key to Column1. Column8 is a string column and Column1 is an integer column. The foreign key and key pairs must be defined so that their data types match.

## **See Also**

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE33

ICE33 processes entries in the [Registry table](#) and may issue a warning for each table entry that registers Classes, Filename Extensions, ProgIDs, Shell Verbs, Remote Server AppIDs, MIME types, or Typelibs. The warnings are obsolete and can be ignored.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE34

ICE34 validates that each radio button on every [RadioButtonGroup Control](#) has a property in the Property column of the [RadioButton table](#) that specifies its radio button group. ICE34 validates that this property exists and is set to a default value in the [Property table](#) which is equal to one of the group's radio button values in the Value column of the [RadioButton table](#).

A radio button group must have a default for users to be able to select a choice using the TAB key. This is required for proper user accessibility.

ICE34 reports missing tables.

## Result

ICE34 post an error message if there is a radio button that specifies an invalid property.

## Example

ICE34 reports the following errors for the example shown.

ICE34 error	Description
Control DialogA.Control2 must have a property because it is of type RadioButtonGroup.	There is a <a href="#">RadioButtonGroup control</a> , without the <a href="#">Indirect control</a> bit set in the Attributes column of the <a href="#">Control table</a> , that does not have a property listed in the Property column.
Maybe is not a valid default value for the RadioButtonGroup using property Property3. The value must be listed as an option in the RadioButtonGroup table.	There is a default value for a property specified in the Value column of the <a href="#">Property table</a> that is not one of the values for the radio button group specified in the Value column of the <a href="#">RadioButton table</a> .
Property PropertyB must be defined because it is an indirect property of a RadioButtonGroup	The property referenced by this RadioButton group is an indirect property, and the value of the indirect

control DialogA.Control4	property is not one of the choices for the RadioButton group.
Maybe is not a valid default value for the property PropertyA. The property is an indirect RadioButtonGroup property of control DialogA.Control5 (via property Property5).	The value of the indirect property referenced via the control is not one of the default values for that RadioButtonGroup.

### Control Table (partial)

Dialog	Control	Type	Attributes	Property
DialogA	Control1	RadioButtonGroup	0	Property1
DialogA	Control2	RadioButtonGroup	0	
DialogA	Control3	RadioButtonGroup	0	Property3
DialogA	Control4	RadioButtonGroup	8	Property4
DialogA	Control5	RadioButtonGroup	8	Property5

### Property Table (partial)

Property	Value
Property1	Yes
Property3	Maybe
Property4	PropertyB
Property5	PropertyA
PropertyA	Maybe

### RadioButton Table (partial)

---

Property	Order	Value
Property1	1	Yes
Property1	2	Now
Property2	1	Yes
Property2	2	No
Property3	1	Yes
Property3	2	No
Property4	1	Yes
Property4	2	No
PropertyA	1	Yes
PropertyA	2	No
PropertyB	1	Yes
PropertyB	2	No

To fix the errors reported by this ICE, check the following:

- That every RadioButton control entry without the indirect attribute set has a property listed in the Property column:
- That every such property has at least one corresponding entry in the RadioButton table.
- That every such property is defined in the Property table, with a value that is one of the choices from the RadioButton table.
- That every property referenced in the Property column of a RadioButton control with the indirect attribute set is defined in the Property table.

**See Also**

## ICE Reference

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE35

ICE35 validates that components containing compressed files stored in a [cabinet file](#) are not set to run from source. With Windows Installer 2.0 or later, this restriction has been removed.

ICE35 queries the Cabinet column of the [Media table](#) to determine which files are compressed and stored in a cabinet file. It queries the [File table](#) to determine which components contain these files. Finally, it checks the [Component table](#) to determine whether the run-from-source bits are set in the Attributes column.

## Result

ICE35 posts an error message if there is a compressed file stored in a cabinet file belonging to a component with the `msidbComponentAttributesSourceOnly` bit set in the Attributes column of the [Component table](#). With Windows Installer 2.0 or later, this is changed from an error to a warning message. A package that supports only Windows Installer 2.0 and later has the `PID_PAGECOUNT` property of the Summary Information Stream set to a value of at least 200.

ICE35 posts warning message if there is a compressed file stored in a cabinet file belonging to a component with the `msidbComponentAttributesOptional` bit set in the Attributes column of the [Component table](#). This warning message has been removed with Windows Installer 2.0 and later.

If multiple files in a component are in a cabinet file, ICE35 reports errors for each file that has the run from source bit set.

## Example

ICE35 reports the following errors and warnings for the example shown using a version earlier than Windows Installer version 2.0.

ICE35 Error	Description
ERROR: Component Component3 cannot be Run	There is a compressed file stored in a cabinet file and this file belongs to a

<p>From Source only, because its member file 'File3' is compressed.</p>	<p>component with the SourceOnly bit set in the Attributes column of the <a href="#">Component table</a>. To fix this error change the lower 2 bits of Component2's Attributes value to "00", meaning Local only, or remove File4 from the CAB file.</p>
<p>ERROR: Component Component3 cannot be Run From Source only, because its member file 'File3' is compressed.</p>	<p>There is a compressed file stored in a cabinet file and this file belongs to a component with the SourceOnly bit set in the Attributes column of the <a href="#">Component table</a>. Because the files in a component do not all have to originate from the same media, ICE35 reports errors for each file in the component that is in a cabinet.  To fix this error change the lower 2 bits of Component2's Attributes value to "00", meaning Local only, or remove File4 from the CAB file.</p>

### Media Table (partial)

DiskID	LastSequence	Cabinet
1	2	
2	4	One.cab
3	5	#Two.cab

### File Table (partial)

File	Component_	Sequence
File1	Component1	1
File2	Component2	2

File3	Component2	3
File4	Component3	4
File5	Component3	5

### Component Table (partial)

Component	Attributes
Component1	0
Component2	2
Component3	1

### Shortcut Table (partial)

Shortcut	Icon_
Shortcut1	Icon2

Note that files can also be marked as compressed using the **Word Count Summary** Property of the [Summary Information stream](#).

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE36

ICE36 validates that every icon in the Icon table is listed at least once in the **ARPPRODUCTICON** property or the **Class**, **ProgId**, or **Shortcut** tables.

During advertisement, the installer installs all the icons listed in the **Icon table** on the user's computer. Having unused icons in the Icon table does not prevent the installation from running, however it does unnecessarily increase the size of the .msi file and the time and space required to advertise a feature.

If an icon is not referenced in the property or table and there is no UI provided to create a reference at run time, you should remove the icon to achieve better performance.

## Result

ICE36 posts a message if there is a icon in the Icon table that is not referenced in the Class, ProgId, or Shortcut tables and if there is no UI provided to create such a reference at run time.

## Example

ICE36 reports the following error for the example shown.

Icon Bloat. Icon Icon4 is not used in the Class, Shortcut, c

**Icon Table** (partial)

Name	Data
Icon1	Control1
Icon2	Control2
Icon3	Control3
Icon4	Control4

## ProgID Table (partial)

<b>ProgID</b>
Property1

## Class Table (partial)

<b>CLSID</b>
{3E469ABA-3644-11d2-8892-00A0C981B015}

## Shortcut Table (partial)

<b>Shortcut</b>	<b>Icon_</b>
Shortcut1	Icon2

## See Also

### [ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE38

ICE38 validates that every component being installed under the current user's profile also specifies a registry key under the **HKEY\_CURRENT\_USER** root in the KeyPath column of the [Component table](#).

## Result

ICE38 posts an error if a component installed under the user's profile does not specify a HKCU registry key.

## Example

ICE38 reports the following errors for the sample shown.

ICE38 error	Description
Component Component1 installs to user profile. It must use a registry key under HKCU as its KeyPath, not a file.	The value of the attributes column of Component1 is 0, meaning that the component must use a file as its KeyPath. This causes difficulties when multiple users install the component on the same computer. To fix this error on Component1, set the RegistryKeyPath bit in the Attributes column of the <a href="#">Component table</a> and change the entry in the KeyPath column to a value listed in the Registry column of the <a href="#">Registry table</a> .
Component Component2 installs to user profile. It must use a registry key under HKCU as its KeyPath. The KeyPath is currently NULL.	Component2 has the RegistryKeyPath bit set in the Attributes column of the <a href="#">Component table</a> . The KeyPath field must therefore contain a key to the Registry column of the <a href="#">Registry Table</a> but the KeyPath column is Null. To fix this error, change the KeyPath value to a valid entry into the Registry table.
Component	Component3 has the RegistryKeyPath bit set in the

<p>Component3 installs to user profile. It's KeyPath registry key must fall under HKCU.</p>	<p>Attributes column of the <a href="#">Component table</a> but the root of the registry entry specified in the Root column of the Registry table specifies <b>HKEY_LOCAL_MACHINE</b> rather than <b>HKEY_CURRENT_USER</b>. To fix this error, use a valid registry entry under <b>HKEY_LOCAL_MACHINE</b> as the KeyPath for this component or change the value in the Root column of the <a href="#">Registry table</a> to -1 or 1.</p>
<p>The KeyPath registry entry for component Component4 does not exist.</p>	<p>Component4 has the RegistryKeyPath bit set in the Attributes column of the <a href="#">Component table</a> but the entry in the KeyPath column does not exist in the <a href="#">Registry Table</a>. To fix this error, add an entry for Reg4 to the Registry table that is a under <b>HKEY_CURRENT_USER</b>.</p>
<p>The Registry Entry Reg5 is set as the KeyPath for component Component5, but that registry entry doesn't belong to Component5.</p>	<p>The Registry entry referenced in the KeyPath column of the component was found and lies under the HKCU tree, but the registry entry's Component_ column does not refer back to the same component that listed it as the KeyPath. This means that the registry entry used as the KeyPath of the component would only be created when some other component was installed. To fix this error change the KeyPath value to refer to a registry entry that belongs to the component, or change the registry entry to belong to the component using it as a KeyPath.</p>

### Directory Table (partial)

Directory	Directory_Parent	DefaultDir
Dir1		StartMenuFolder
Dir2		DesktopFolder
Dir3	Dir3	AppData
Dir4	Dir3	SubDir

## Component Table (partial)

Component	Directory_	Attributes	KeyPath
Component1	Dir1	0	File1
Component2	Dir2	4	
Component3	Dir3	4	Reg3
Component4	Dir4	4	Reg4
Component5	Dir5	4	Reg5

## Registry Table (partial)

Registry	Root	Value	Component_
Reg3	2		Component3
Reg5	0		Component4

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE39

ICE39 validates the [Summary Information Stream](#) of the database.

ICE39 checks the format of the following properties:

- [Word Count Summary](#)
- [Page Count Summary](#)
- [Template Summary](#)
- [Revision Number Summary](#)
- [Create Time/Date Summary](#)
- [Last Saved Time/Date Summary](#)
- [Last Printed Summary](#)

If the [Word Count Summary](#) Property specifies that the source is compressed, ICE39 posts a warning if any files are also marked as compressed in the Attributes column of the [File table](#). See [Using Cabinets and Compressed Sources](#).

ICE39 posts a warning if the [Word Count Summary](#) Property specifies that the package is UAC compliant and the [MsiPackageCertificate Table](#) is not empty.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE40

ICE40 does miscellaneous validation.

## Result

ICE40 posts warnings on the following:

- The **REINSTALLMODE** property has been overridden.
- The **RemoveIniFile** table has a Delete Tag entry with no value.
- The .msi file is missing the **Error** table and the **Page Count Summary** Property is less than or equal to 100. This ICE warning is obsolete because Windows Installer does not require the package to have an Error table. Error messages can be retrieved using Msimsg.dll.

## Example

### Property Table

Property	Value
<b>REINSTALLMODE</b>	A

### RemoveIniFile Table

RemoveIniFile	Action	Value
<b>REINSTALLMODE</b>	4	

## Results

ICE40 would report the following errors.

ICE40 error	Description
-------------	-------------

<p><b>REINSTALLMODE</b> is defined in the Property table. This may cause difficulties.</p>	<p>Defining the <b>REINSTALLMODE</b> property in .msi file can lead to unexpected behavior. To fix this error, don't define this property.</p>
<p>RemoveIniFile entry Remove1 must have a value, because the Action is "Delete Tag" (4).</p>	<p>There is a Delete Tag action in the in the RemoveIniFile column of the <a href="#">RemoveIniFile table</a> without specifying a tag to delete in the Value column.</p>
<p>Error Table is missing. Only numerical error messages will be generated.</p>	<p>This ICE warning is obsolete because Windows Installer does not require the package to have an Error table. Error messages can be retrieved using Msimsg.dll.</p> <p>This warning means that the .msi file is missing the <a href="#">Error table</a> and the <a href="#">Page Count Summary Property</a> is less than or equal to 100.</p> <p>To fix this error, use a current version of the Windows Installer, or add an Error table to the installation package and author formatting templates in the Message column for error messages.</p>

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE41

ICE41 validates that the entries in the [Class](#) and [Extension](#) tables refer to entries in the [Component](#) table that implement the class object or extension of the component.

## Result

ICE41 posts an error if there is a feature that does not contain the component implementing the class object or extension.

## Example

ICE41 reports the following errors for the example shown.

ICE41 error	Description
Class {00000000-0000-0000-0000-000000000000} references feature Feature2 and component Component1, but the that Component is not associated with that Feature in the FeatureComponents table.	There is a feature that does not contain the component implementing the class object. This means that the installer does not install the component with the feature and that advertising may not work as expected. To fix this error, change the entry in the Feature_ column of the <a href="#">Class table</a> entry to reference a feature that installs component listed in the Component_ column or change the feature and component associated in the <a href="#">FeatureComponents table</a> .
Extension .yip references feature Feature1 and component Component2, but the that Component is not associated with that Feature in the FeatureComponents table.	There is a feature that does not contain the component implementing the extension. This means that the installer does not install the component with the feature and that advertising may not work as expected. To fix this error, change the entry in the Feature_ column of the <a href="#">Extension table</a>

entry to reference a feature that installs the component listed in the Component\_ column or change the feature and component associated in the [FeatureComponents table](#).

### FeatureComponents Table (partial)

Feature_
Feature1
Feature2

### Class Table (partial)

CLSID	Component_	Feature_
{00000000-0000-0000-0000-000000000000}	Component1	Feature2

### Class Table (partial)

Extension	Component_	Feature_
.yip	Component2	Feature1

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE42

ICE42 validates that InProc servers are not linked to EXE files in the [Class table](#). It also validates that only LocalServer and LocalServer32 classes have arguments and DefInProc values.

## Result

ICE42 posts an error if there are InProc servers linked to EXE files in the Class table.

## Example

ICE42 would report the following errors for the example shown.

ICE42 error	Description
CLSID '{GUID1}' is an InProc server, but the implementing component 'Component1' has an EXE ('test.exe') as its KeyFile.	There is an executable file specified as an InProc server. EXE files cannot be InProc servers.
CLSID '{GUID1}' in context 'InProcServer32' has an argument. Only LocalServer contexts can have arguments.	To fix this error, remove the argument.
CLSID '{GUID1}' in context 'InProcServer32' species a default InProc value. Only LocalServer contexts can have default InProc values.	There is an object with a default InProc value that is not an object operating in the LocalServer or LocalServer32 contexts. To fix this error, remove the DefInProc value or change the context of the class.

## Class Table (partial)

CLSID	Context	Component_	DefInProcHandler	Argument

{GUID1}	InProcServer32	Component1	InProcServer	Arg
---------	----------------	------------	--------------	-----

### Component Table (partial)

Component	KeyPath
Component1	File1

### File Table (partial)

File	Filename
File1	test.exe

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE43

ICE43 validates that shortcuts that do not reference a feature as their Target (non-advertised shortcuts) are in components having a HKCU registry entry as their key path.

## Result

ICE43 posts an error message if a non-advertised shortcut is in a component that does not have a HKCU registry entry as its key path.

## Example

ICE43 would report the following errors for the example shown.

<b>ICE43 error</b>	<b>Description</b>
Component Component1 has non-advertised shortcuts. It must use a registry key under HKCU as its KeyPath, not a file.	The attributes column of Component1 is 0, meaning that the component uses a file as its KeyPath. This causes non-advertised shortcuts in this component to be installed for the first user on the computer ONLY. Users who install the component later do not see the shortcuts because the component appear to the installer as already existing on the computer. To fix this error, set the RegistryKeyPath bit of the attributes to switch the Component to a Registry entry, then change the KeyPath value to a valid entry in the Registry table.
Component Component2 has non-advertised shortcuts. It must use a registry key under HKCU as its KeyPath. The KeyPath is currently null.	The Attributes column is set to use the registry, but the KeyPath is null. The KeyPath must refer to an entry in the Registry Table. To fix this error, change the KeyPath value to a valid entry in the Registry table.

Component Component3 has non-advertised shortcuts. Its KeyPath registry key must fall under HKCU.	The Attributes column is set to use the registry, but the referenced registry entry is not under HKCU. To fix this error, either switch to a different registry entry as the KeyPath for this component, or change the Root value of the Registry entry to either -1 or 1.
The KeyPath registry entry for component Component4 does not exist.	The Registry entry referenced in the KeyPath column of the component is not in the Registry Table. To fix this error, create an entry.
The Registry Entry Reg5 is set as the KeyPath for component Component5, but that registry entry doesn't belong to Component5.	There is a Registry entry referenced in the KeyPath column of the component that lies under the HKCU tree, but the registry entry's Component_ column does not refer back to the same component that listed it as the KeyPath. This means that the registry entry used as the KeyPath of the component is only created if some other component was installed. To fix this error, change the KeyPath value to refer to a registry entry that belongs to the component or change the registry entry to belong to the component using it as a KeyPath.

### Component Table (partial)

Component	Attributes	KeyPath
Component1	0	File1
Component2	4	
Component3	4	Reg3
Component4	4	Reg4
Component5	4	Reg5

## Registry Table (partial)

Registry	Root	Value	Component_
Reg3	2		Component3
Reg5	0		Component4

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE44

ICE44 validates that the [NewDialog](#), [SpawnDialog](#), and [SpawnWaitDialog](#) ControlEvents reference valid dialog boxes in the [Dialog table](#).

## Result

ICE44 posts an error message if there is a dialog control event that does not reference a dialog box listed in the Dialog table.

## Example

ICE44 would report the following errors for the example shown.

ICE44 error	Description
Control Event for Control 'Dialog1'. 'Control1' is of type SpawnDialog, but its argument Dialog2 is not a valid entry in the Dialog Table.	There is a SpawnDialog, SpawnWaitDialog, or NewDialog actions that has an argument that does not refer to a dialog box in the Dialog table. To fix this error, add an argument that is a key in the Dialog Table.

## Dialog Table (partial)

Dialog	Title
Dialog1	

## ControlEvent Table (partial)

Dialog_	Control_	Type	Argument
Dialog1	Control1	SpawnDialog	Dialog2

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE45

ICE45 validates that bit field columns in the database do not set any reserved bits to 1.

Reserved bits provide no functionality in current versions of the installer, but might in future versions. They should be set to 0 to be compatible with future versions of Windows Installer.

## Result

ICE45 posts an error message if any of the following tables contains a bit field with a reserved bit set to a value of 1.

- [BBControl table](#)
- [Dialog table](#)
- [Feature table](#)
- [File table](#)
- [MoveFile table](#)
- [ModuleConfiguration table](#)
- [ODBCDataSource table](#)
- [Patch table](#)
- [RemoveFile table](#)
- [ServiceControl table](#)
- [ServiceInstall table](#)
- [TextStyle table](#)

ICE45 posts one of two warning messages if the [Control Table](#) contains a bit field with a reserved bit set to a value of 1.

## Example

ICE45 reports the following error for the example shown.

```
Row 'File1' in table 'File' has bits set in the 'Attributes'
```

column that are reserved. They must be 0 to ensure compatibility with future installer versions.

ICE45 reports the following warning for the example shown.

Row 'Dialog1.Edit2' in table 'Control' has bits set in the column that are reserved. They should be 0 to ensure compatibility with future installer versions.

#### File Table (partial)

File	Attributes
File1	128

#### Control Table (partial)

Dialog	Control	Attributes
Dialog1	Edit1	2097152
Dialog1	Edit2	1048576

## See Also

### [ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE46

ICE46 checks for custom properties in conditions, formatted text, and other locations that differ from a system defined property only by the case of one or more characters.

## Result

ICE46 posts an informational message if there is a custom property in a condition, formatted text, and other location that differs from a system defined properties only in the case of one or more characters.

## Example

ICE46 reports the following errors for the example shown.

ICE46 error	Description
Property ReinstallMode defined in property table differs from another defined property only by case.	The system defined property name <b>REINSTALLMODE</b> differs from the custom property by case only. Properties are case sensitive, so custom property is not the same as the system property. This is a common cause of errors.
Property 'Myproperty' referenced in column 'InstallExecuteSequence'. 'Condition' of row 'InstallFinalize' differs from a defined property by case only.	The property table defines the table MyProperty, but the referenced property is Myproperty. Properties are case sensitive, so the two properties are NOT the same. This is a common cause of errors.

## Property Table

Property	Value
ReinstallMode	omus

MyProperty

a value

### InstallExecuteSequence Table (partial)

Action	Condition
InstallFinalize	Myproperty

### See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE47

ICE47 checks the [Feature](#) and [FeatureComponents](#) tables for features with 1600 or more components.

## Result

ICE47 posts an error message if a feature exceeds the maximum limit of 1600 components per feature.

## Example

ICE47 would report the following warning:

```
Feature 'Feature1' has 1600 components. This could cause
  problems on Win9X systems. You should try to have less
  than 800 components per feature."
```

### Feature Table (partial)

Feature
Feature1

### FeatureComponents Table (partial)

Action	Condition
Feature1	Component1
Feature1	Component1600

To fix this warning, try splitting the feature into several features

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE48

ICE48 checks for directories that are hard-coded to local paths in the [Property table](#).

Do not hard-code directory paths to local drives because computers differ in the setup of the local drive. This practice may be acceptable if deploying an application to a large number of computers on which the relevant portions of the drives are all the same.

## Result

ICE48 posts an error message if there is a directory that is hard-coded to a local path in the Property table.

## Example

ICE48 would report the following warning for the example shown.

```
Directory 'Dir1' appears to be hardcoded in the property table to a local drive.
```

### Directory Table (partial)

Directory	Directory_Parent	DefaultDir
Dir1		SourceDir

### Property Table (partial)

Property	Value
Dir1	d:\source

## See Also

## ICE Reference

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE49

ICE49 checks for default registry entries that are not a **REG\_SZ** type.

## Result

ICE49 posts an warning if there is a default registry entry that is not a **REG\_SZ** type.

## Example

ICE49 reports the following warning for the example shown.

```
Reg Entry 'Reg1' is not of type REG_SZ. Default types must be used on Win95 Systems. Make sure the component is conditional to never be installed on Win95 machines.
```

The value '#123' is a **DWORD** registry value.

[Registry Table](#) (partial)

Registry	Name	Value
Reg1		#123

To fix this warning, change the value to type **REG\_SZ**.

Components with non-**REG\_SZ** are valid.

## See Also

[Conditional Statement Syntax](#)

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE50

ICE50 checks that shortcut icons are specified to display correctly and match their target file's extension.

## Result

ICE50 posts an error message if the extension of the icon and target files do not match. ICE50 posts a warning if icons are stored in files that do not have an .exe or .ico extension.

## Example

ICE50 reports the following error for the example shown.

ICE50 error or warning	Description
The extension of Icon 'Icon2.dat' for Shortcut 'Shortcut2' does not match the extension of the Key File for component 'Component2'.	If the extensions of the icon and the target file do not match, the shortcut will not have the correct context menu when the component is advertised. To fix this error, rename the icon to match the extension of the target file.
The extension of Icon 'Icon1.bat' for Shortcut 'Shortcut1' is not "exe" or "ico". The Icon will not be displayed correctly.	Not all versions of the shell correctly display icons stored in files that do not have extensions of "exe" or "ico". To fix this warning, rename the icon have an extension of "exe" or "ico".

## File Table (partial)

File	FileName
File1	File1.bat
File2	File2.pl

### Feature Table (partial)

Feature
Feature1

### Component Table (partial)

Component	KeyPath
Component1	File1
Component2	File2

### Icon Table

Name	Data
Icon1.bat	[Binary Data]
Icon2.dat	[Binary Data]

### Shortcut Table (partial)

Shortcut	Component	Target	Icon_
Shortcut1	Component1	Feature1	Icon1.bat
Shortcut2	Component2	Feature1	Icon2.dat

### See Also

## ICE Reference

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE51

ICE51 checks that a title has been provided for font resource files.

You must provide a title for a font resource that does not have an embedded name. Only .ttc, .ttf, and .otf font resource files do not require a title, because these files contain an embedded name. Do not provide a title for a font resource that contains an embedded name because the system then registers the font twice.

**Windows Installer 4.5 and earlier:** ICE51 does not check .otf font resource files.

## Result

ICE51 posts an error if there are any .ttc, .ttf, and .otf font resource files with titles. ICE51 posts a warning if there are any other font resource files without a title.

## Example

ICE51 would report the following warning for the example shown.

```
Font 'Font1' is a TTC\TTF\OTF font, but also has a title.
```

ICE51 would report the following error message for the example shown.

```
Font 'Font2' does not have a title. Only TTC\TTF\OTF fonts c
```

## File Table (partial)

File	FileName
Font1	font1.ttf
Font2	font2.fon

## Font Table



<b>File_</b>	<b>FontTitle</b>
Font1	A Really Cool Font
Font2	

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE52

ICE52 checks for private properties in the [AppSearch table](#). See [About Properties](#).

When using Windows 2000 all properties set in the Property column of the AppSearch table must be public properties.

## Result

ICE52 posts a warning if there is a private property in the AppSearch table.

## Example

ICE52 posts the following warning for the example shown.

```
Property 'Property2' in AppSearch row 'Property2'. 'Signature' is not public. It should be all uppercase.
```

## AppSearch Table

Property	Signature
PROPERTY1	Signature1
Property2	Signature2

To fix this warning change to the custom public property: PROPERTY2.

## See Also

### [ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE53

ICE53 checks for entries in the Registry table that write private installer information or policy values to the system registry.

## Result

ICE53 posts a warning if the Registry table specifies writing internal or policy information to the registry.

## Example

ICE53 posts the following warning for the example shown.

```
Registry Key 'Registry1' writes Installer internal or policy
```

[Registry Table](#) (partial)

Registry	Root	Key
Registry1	1	Software\Policies\Microsoft\Windows\Installer\DisableRol

Registry table row 'Registry1' writes a system policy value in the registry that affects the installation of all packages. Depending on the package, it may be possible to disable rollback for this package alone by setting the **DISABLEROLLBACK** property in the [Property table](#). See [Rollback Installation](#).

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE54

ICE54 checks for components that use a companion file as their key path.

The key path file of a component must not derive its version from a different file. This can cause some files to fail to install. See the [File table](#) for more information about companion files.

## Result

ICE54 posts a warning if any component has a key path file that derives its version from another file.

## Example

ICE54 returns the following warning for the example shown.

```
Component 'Component1' uses file 'File1' as its KeyPath, but
```

## Component Table (partial)

Component	Attribute	KeyPath
Component1	0	File1

## File Table (partial)

File	Version	Language
File1	File2	
File2	1.0.0.0	1033

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE55

ICE55 validates that all LockPermission objects exist and have valid permission values.

## Result

ICE55 post an error if a LockObject listed in the [LockPermissions table](#) does not exist or if no privilege level is specified in the Permission column.

## Example

ICE55 would report the following errors for the example.

```
LockObject 'File1'. 'File'. ''.'guest' in the LockPermissions
has a null Permission value.
Could not find item 'File3' in table 'File' which is referer
in the LockPermissions table.
```

### LockPermissions Table (partial)

LockObject	Table	Domain	User	Permission
File1	File		guest	
File3	File		guest	1

### File Table (partial)

File	Version	Language
File1	File2	
File2	1.0.0.0	1033

The object File1 has a null in the Permission column. Each row must

have a value in the Permissions column. To fix this error specify a numeric value in this column. If no privileges are needed for this object then you should remove the row.

The object File3 described in the LockPermissions table is not listed in the File table. To fix this error refer to a valid object.

## **See Also**

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE56

ICE56 validates that the directory structure of the .msi file has a single root directory, that the root is the **TARGETDIR** property, and that the **SourceDir** property value is in the DefaultDir column of the [Directory table](#).

If a .msi file has multiple roots or specifies a root other than **TARGETDIR**, an [administrative installation](#) does not create a correct administrative image.

Note that empty directories are not checked by ICE56. The directory structure passes validation with multiple root directories if the extra directories are empty.

## Result

ICE56 posts an error if the .msi does not have a single root, **TARGETDIR**, or if **SourceDir** is not specified in the DefaultDir column of the [Directory table](#).

## Example

ICE56 reports the following errors for the example shown.

```
Directory 'TARGETDIR' has a bad DefaultDir value.  
Directory 'Root2' is an invalid root Directory.
```

### Directory Table

Directory	Directory_Parent	DefaultDir
TARGETDIR		Temp
Root2	Root2	SourceDir

To fix the first error, the **TARGETDIR** root should have a DefaultDir value of **SourceDir**. SOURCEDIR is also accepted. It may be possible to make **TARGETDIR** the parent of the second root, and use the '.' value in the

DefaultDir column. See the [Directory table](#) for more information.

To fix the second error, the Directory structure should have only one root called **TARGETDIR**.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE57

ICE57 validates that individual components do not mix per-machine and per-user data. This ICE custom action checks registry entries, files, directory key paths, and non-advertised shortcuts.

Mixing per-user and per-machine data in the same component could result in only partial installation of the component for some users in a multi-user environment.

See the [ALLUSERS](#) property.

## Result

ICE57 posts an error if it finds any component that contains both a per-machine and per-user registry entries, files, directory key paths, or non-advertised shortcuts.

## Example

ICE57 reports the following errors for the example shown.

```
Component 'Component1' has both per-user and per-machine
data with a per-machine KeyPath.

WARNING: Component 'Component2' has both per-user and
per-machine data with an HKCU Registry KeyPath.

Component 'Component3' has a registry entry that
can be either per-user or per-machine and a per-machine

Component 'Component4' has both per-user data and
a keypath that can be either per-user or per-machine.
```

### Component Table (partial)

Component	Directory	Attributes	KeyPath
Component1	DirectoryA	0	FileA
Component2	DirectoryA	4	RegKeyB

Component3	DirectoryA	0	FileC
Component4	DirectoryA	4	RegKeyD

### Registry Table (partial)

Registry	Root	Component_
RegKeyA	1	Component1
RegKeyB	1	Component2
RegKeyC	-1	Component3
RegKeyD	-1	Component4

### File Table (partial)

File	Component_
FileA	Component1
FileB	Component2
FileC	Component3
FileD	Component4

### Directory Table

Directory	Directory_Parent	DefaultDir
TARGETDIR		SourceDir
DirectoryA	TARGETDIR	DirectoryA

To fix the errors, reorganize the application such that each component contains only per-user or per-machine resources, and not both.

The first error message is posted because Component1 contains FileA (per-machine) and the HKCU registry key RegKeyA (per user).

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE58

ICE58 checks that your [Media table](#) does not have more than 80 rows.

## Result

Warnings reported by ICE58 cause the installation to fail unless the package is installed with Windows Installer 2.0 or later. Beginning with Windows Installer 2.0, the restriction to more than 80 media table entries is removed. No warning is issued if the package's [Page Count Summary](#) Property is greater than or equal to 150. Packages of schema 200 or higher can only be installed by Windows Installer 2.0 or later.

## Example

ICE58 reports the following errors and warnings for the example shown.

This package has 81 media entries. Packages are limited to 80

To fix this error, eliminate any unused media table entries, consolidate media table entries that refer to the same media, and repackage your application to reduce the media required.

[Media Table](#) (partial)

DiskId	LastSequence_
1	10
2	20
...	...
81	810

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE59

ICE59 checks that advertised shortcuts belong to components that are installed by the target feature of the shortcut.

Errors reported by ICE59 generally lead to the following behavior:

1. The advertised shortcut will launch the Windows Installer to install the feature listed in the Target column.
2. But because the [FeatureComponents table](#) does not map the target feature to the component containing the shortcut, the keyfile of the component (which is activated by the shortcut) is not installed.
3. Therefore the shortcut is broken and will not do anything.

## Result

ICE59 posts an error if an advertised shortcut does not belong to the components that are installed by the target feature of the shortcut.

## Example

ICE59 reports the following error for the example shown:

```
The shortcut ShortcutB activates component ComponentB and ac
```

In this case, ShortcutB advertises FeatureA, and when activated, starts the key file of ComponentB. Yet ComponentB is never installed by FeatureA, so even after the installation-on-demand phase completes, the target of the shortcut does not exist.

To fix this error, add a row to the [FeatureComponents table](#) that associates FeatureA and ComponentB.

### Shortcut Table (partial)

Shortcut	Target	Component_
ShortcutB	FeatureA	ComponentB

---

## FeatureComponents Table

Feature_	Component_
FeatureA	ComponentA

## Feature Table (partial)

Feature	Level
FeatureA	10

## Component Table (partial)

Component	KeyPath
ComponentA	FileA
ComponentB	FileB

## File Table (partial)

File	Component_	Sequence
FileA	ComponentA	1
FileB	ComponentB	2

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE60

ICE60 checks that files in the [File table](#) meet the following condition:

- If the file is not a font and has a version, then it must have a language.
- ICE60 checks that no versioned files are listed in the [MsiFileHash table](#).

Failure to fix a warning reported by ICE60 generally leads to a file being needlessly reinstalled when a product repair is done. This happens because the file to be installed in the repair and the existing file on disk have the same version (they are the same file) but different languages. The file table lists the language as null, but the file itself has a language value in the resource. Based on the [file versioning rules](#), the installer favors the file to be installed, so it is recopied needlessly.

## Result

ICE60 posts a warning or an error if a file in the [File table](#) that is not a font and has a version, does not have a language.

ICE60 posts the following error if a file listed in the MsiFileHash table is versioned.

```
ERROR: "The file [1] is Versioned. It cannot be hashed"
```

## Example

ICE60 reports the following error and warning for the example shown. (File B is a font; the other files are not.)

```
WARNING: The file FileE is not a Font, and its version is not
```

FileA has both a version and a language; therefore no warning or error is generated.

FileB has a version but no language. No warning or error is generated,

however, because it is a font.

FileC is a companion reference, so it does not have to have a language. No warning or error is generated.

FileD has no version, so it does not need to have a language. No warning or error is generated.

FileE has a version but no language. Therefore a warning is generated.

To fix this warning, add a language to FileE.

Files should have language values stored in the version resource whenever possible. If a file is language neutral, use the [LANGID 0](#).

**File Table** (FileB is a font; the other files are not.)

File	Version	Language
FileA	1.0	1033
FileB	1.0	
FileC	FileA	
FileD		
FileE	1.0	

## Font Table

File	FontTitle
FileB	Font Title

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE61

ICE61 checks the upgrade table to ensure that the following conditions are true:

- All ActionProperty properties are not pre-authored in the Property table.
- All ActionProperty properties are Public Properties.
- All ActionProperty properties are included in the [SecureCustomProperties](#) property.
- All ActionProperty properties are unique to each record in the Upgrade table.
- All VersionMax values are not less than the corresponding VersionMin values.
- VersionMin and VersionMax values are valid product versions. See the [ProductVersion](#) property for the valid product version format.
- No attempt is made to remove a newer or equal version of the current product.

Failure to fix a warning or error reported by ICE61 generally leads to problems in upgrading your application. Depending on the exact error, this could be anything from leaving files from the older version behind, deleting files from the older version even though they are needed by the new application, or complete failure of the upgrade.

## Result

ICE61 posts a warning or error if any of the above conditions are not true.

## Example

ICE61 reports the following errors and warning for the examples shown.

```
This product should remove only older versions of itself. Th
```

In this case, the first row would try to remove a product of the same version. (VersionMax column is equal to the product version in the Property table).

To fix this error, use a version in the VersionMax column lower than the current version specified in the Property table. Remove the msidbUpgradeAttributesVersionMaxInclusive bit from the Attributes column if the VersionMax is equal to the current version. If the intent is only to detect the product and not remove it, adding the msidbUpgradeAttributesOnlyDetect bit to the Attributes column will also fix this error.

```
Upgrade.ActionProperty EnglishAPPFFOUND must be added to the
```

Unless the property is listed in the [SecureCustomProperties](#) property, the property is not passed to the server side of the install when the property is found.

To fix this error, add the property to [SecureCustomProperties](#).

```
Upgrade.ActionProperty EnglishAPPFFOUND must not contain lower
```

Upgrade properties must be public properties for the result to be passed to the server side of the installation.

To fix this error, use all uppercase letters in the property name.

```
Upgrade.ActionProperty OLDAPPFFOUND may be used in only one r
```

A property can only be used in one row of the Upgrade table.

To fix this error, use a different property for the second row.

```
Upgrade.VersionMax cannot be less than Upgrade.VersionMin. (
```

The minimum version must be less than the maximum version.

To fix this error, check your version numbers for typos. If they are correct and you want to use the range between the two versions, switch them so that VersionMin is less than VersionMax.

## Property Table

Property	Value
<b>UpgradeCode</b>	{61AA4C55-E17F-11D2-93BB-0060089A76DB}
<b>ProductVersion</b>	2.01.0000
<b>SecureCustomProperties</b>	OLDAPPFOUND

## Upgrade Table

UpgradeCode	VersionMin	VersionMax	Language	Attributes	Remove
{61AA4C55-E17F-11D2-93BB-0060089A76DB}		2.01.0000		513	
{61AA4C55-E17F-11D2-93BB-0060089A76DB}	2.01.0001	2.01.0000			
{C6CB4596-D8E8-D5A4-635F-9FE456D682EB}	1.00.0000	2.00.0000	1033		[AppFea

## See Also

### [ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE62

ICE62 performs extensive checks on the [IsolatedComponent table](#) for data that may cause unexpected behavior.

Failure to fix an error reported by ICE62 can result in a failure of the isolated component system in a wide variety of ways. For example, if the SharedDllRefCount bit is not set for a shared component, the registration for the component could be removed when another application uses that ComponentId and is uninstalled.

## Result

ICE62 posts a warning or error when it finds data in the IsolatedComponent table that may produce unexpected behavior.

## Example

ICE62 reports the following errors and warning for the examples shown.

```
The component 'Component2' is listed as an isolated application component in the IsolatedComponent table, but the key path is
```

Component2 is listed as the application component for the isolation of component1. However, Component2 has a registry key path, and does not provide a valid executable path to use to isolate the component.

To fix this error, use a different component as the application for the isolated component Component1.

```
The component 'Component1' is listed as an isolated shared component in the IsolatedComponent table, but is not marked with the SharedDll
```

Component1 is listed as an isolated shared component, but does not have the SharedDllRefCount bit set. This could result in the lifetime of the component being incorrect. If another application uses this component (isolated or not) and is uninstalled, the registration for the component is removed but this application's isolated copy remains. This causes repair and uninstall problems.

To fix this error, set the SharedDllRefCount bit for the component.

```
The isolated shared component 'Component1' is not installed  
(or a parent feature of) its isolated application component
```

Component1 and Component2 are installed by different features. Component1 is installed by Feature1, and Component2 is installed by Feature2. Feature1 is not a parent of Feature2, hence it is possible for the application to be installed but not the isolated component, breaking the isolation.

To fix this error, add an entry to the FeatureComponents table linking Component1 to the same feature as (or a parent feature of) the feature that installs Component2.

```
WARNING: The isolated shared component 'Component1' (referer  
is conditionalized. Isolated shared component conditions sho
```

Component1 has a condition in the Component table. If this condition ever changes from TRUE to FALSE during the lifetime of an installation on a computer, the isolated component could be orphaned without registration information.

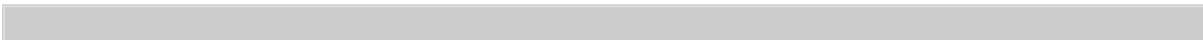
To fix this warning, remove the condition, or author the condition so that it can never change from TRUE to FALSE.

```
WARNING: The isolated shared component 'Component1' is share  
(including 'Component2') that are installed to the directory  
WARNING: The isolated shared component 'Component1' is share  
(including 'Component3') that are installed to the directory
```

Component1 is isolated for both Component2 and Component3, and the two components are also installed to the same directory. The applications share an isolated component, but if one application is removed the shared component is removed as well causing the other applications to lose the isolated component.

To fix this warning, install the applications to different directories or check whether some of the applications truly require an isolated component.

## IsolatedComponent Table



<b>Component_Shared</b>	<b>Component_Application</b>
Component1	Component2
Component1	Component3

### Component Table

<b>Component</b>	<b>ComponentId</b>	<b>Directory_</b>	<b>Attributes</b>	<b>Condition</b>	<b>Ke</b>
Component1		Dir1	0	MYCONDITION	Fi
Component2		TARGETDIR	4		Re
Component3		TARGETDIR	0		Fi

### FeatureComponentsTable

<b>Feature_</b>	<b>Component_</b>
Feature1	Component1
Feature2	Component2
Feature1	Component3

### Feature Table (partial)

<b>Feature</b>	<b>Feature_Parent</b>
Feature1	
Feature2	

**See Also**

## ICE Reference

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE63

ICE63 checks for proper sequencing of the [RemoveExistingProducts action](#). The RemoveExistingProducts action may be placed:

1. Between InstallValidate and InstallInitialize
2. Immediately after InstallInitialize, or after InstallInitialize if the actions between InstallInitialize and RemoveExistingProducts do not generate any script actions.
3. Immediately after InstallExecute or InstallExecuteAgain and before InstallFinalize (the same restriction as above applies).
4. After InstallFinalize.

Failure to fix a warning or error reported by ICE63 leads to failure of the upgrade.

## Result

ICE63 posts a warning or error if the sequencing of the RemoveExistingProducts action is not correct.

## Example

ICE63 reports the following error for the example shown.

```
WARNING: Some action falls between InstallInitialize and RemoveExistingProducts
```

The action 'MyCustomAction' occurs between InstallInitialize and RemoveExistingProducts. If MyCustomAction generates any actions in the script, this causes problems in the installation.

To fix this error, verify that MyCustomAction does not generate any script actions or resequence the actions.

## InstallExecuteSequence Table

Action	Condition	Sequence

InstallInitialize		1000
MyCustomAction		1010
RemoveExistingProducts		1020

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE64

ICE64 checks that new directories in the user profile are removed correctly in roaming scenarios.

Failure to fix a warning or error reported by ICE64 generally leads to problems in completely cleaning the computer during an uninstallation. When a roaming user who has installed the application logs on to a computer for the first time, all of the profile is copied down onto the computer. On advertisement (which takes place after the roaming profile download), the Installer verifies that the directory is already there and therefore does not delete it on uninstallation. This leaves the directory on the user's computer permanently.

## Result

ICE64 posts a warning or an error in a roaming situation if a new directory in the user profile that should be removed is not removed.

## Example

ICE64 reports the following error for the example shown.

The directory 'MyOtherFolder' is in the user profile but is

The folder 'MyOtherFolder' is a custom profile folder. Because it is not listed in the RemoveFile table, it is not removed in some scenarios.

To fix this error, create a row for the folder in the RemoveFile table.

## Directory Table

Directory	Directory_Parent	DefaultDir
AppDataFolder	TARGETDIR	
MyFolder	AppDataFolder	DataFolder
MyOtherFolder	AppDataFolder	DataFolder2

## RemoveFile Table

FileKey	Component_	FileName	DirProperty	InstallMode
Key1	Component1		MyFolder	2

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE65

ICE65 checks that the [Environment table](#) does not have invalid prefix or append values.

Failure to fix a warning or error reported by ICE65 generally leads to problems in install, uninstall, or repair of the environment variable. For example, only some values of a particular variable may be removed if one or more of the values for that variable have a trailing separator.

## Result

ICE65 posts a warning or an error if the environment table has invalid prefix or append values.

## Example

ICE65 reports the following error and warning for the example shown.

```
The environment variable 'Var3' has a separator beginning or
```

The trailing null at the end of the value ([~]) marks this value to be prepended to any existing value. The character immediately before the null (a semicolon) becomes the separator for this value. This value has a semicolon at the beginning of the string as well.

To fix this error, simply delete the leading semicolon.

```
WARNING: The environment variable 'Var2' has an alphanumeric
```

The leading null in the value ([~]) marks this value to be appended to any existing value. The character immediately after the null becomes the separator for this value. In this case, that character is the letter "e", which also occurs in the middle of the string to be appended. This condition (having a separator that is the same as a character within the string to be appended) can cause unpredictable results.

The letter "e", being a common letter, is likely to be found in the value. A better choice would be ";" or some other non-alphanumeric character. (However, if the value is a path, then ":" and "\" and "." are risky choices.)

To fix this warning, use a different separator character.

## Environment Table

<b>Component</b>	<b>Directory</b>	<b>Attributes</b>	<b>KeyPath</b>
Var1	TestVar	[~];AppendThis	TestComponent
Var2	TestVar	[~]eAppendThis	TestComponent
Var3	TestVar	;PrependThis;[~]	TestComponent

## See Also

### [ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE66

ICE66 uses the tables in the database to determine which schema your database should use.

Some functionality may only be available if the package is installed on a system with a current Windows Installer version.

## Result

ICE66 posts a warning if your database is using the wrong schema.

## Example

ICE66 reports the following warning for the example shown.

```
WARNING: Complete functionality of the IsolatedComponents table
```

This warning can be ignored if you want your package to be installed using a current Windows Installer version. For example, if you want your package to be installable only on version 2.0 or later, change your package schema (PID\_PAGECOUNT) to 200.

## IsolatedComponent Table

Component_Shared	Component_Application
Component1	Component2

## Summary Information Stream

PIDt	Value
PID_PAGECOUNT	100

**Table used during execution:**

[\\_Columns table](#)

## **See Also**

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE67

ICE67 checks that the target of a non-advertised shortcut belongs to the same component as the shortcut itself, or that the attributes of the target component ensure that it does not change installation locations.

Failure to fix a warning or error reported by ICE67 can cause the shortcut to be invalid if the target component changes state and the source component does not. For example, when the target file's component is set to run from source, a reinstallation that changes the component to local results in the component containing the shortcut not being reinstalled. Thus the shortcut points to an invalid location.

Note that in some cases, using a different component for the shortcut is unavoidable. For example, if the shortcut is created in the user profile and the file is installed to a non-profile directory, you may not be able to use the same component for both pieces of data. (This results in failures in multi-user scenarios – such as those described in [ICE57](#)). In this case, you may be able to use advertised shortcuts to achieve the behavior you want, or you can simply ensure that the target component cannot change from run-from-source to local.

## Result

ICE67 returns an error or a warning if the target of a non-advertised shortcut does not belong to the same component as the shortcut itself, or if the attributes of the target component do not ensure that the installation locations will not change.

## Example

ICE67 reports the following warning and errors for the example shown.

The shortcut 'Shortcut1' is a non-advertised shortcut with a

Shortcut1 is installed by Component2, but its target file, File1, is installed by component1. The target component is marked optional (meaning that it can be local or run-from-source). One possible situation that would cause a problem is if Component1 changes from run-from-source to

local. This would cause Shortcut1 to point to an invalid location.

To fix this warning, install the shortcut as part of Component1, or mark Component1 as LocalOnly or SourceOnly.

### **File Table (partial)**

<b>File</b>	<b>Component_</b>
File1	Component1

### **Shortcut Table (partial)**

<b>Shortcut</b>	<b>Component_</b>	<b>Target</b>
Shortcut1	Component2	[#File1]

### **Component Table (partial)**

<b>Component</b>	<b>Attributes</b>
Component1	2

## **See Also**

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE68

ICE68 checks that all custom action types needed for an installation are valid. Failure to fix the error reported by ICE68 causes an installation that attempts to execute the action to fail. ICE68 issues a warning if the `msidbCustomActionTypeNoImpersonate` attribute is set without also setting the `msidbCustomActionTypeInScript` attribute.

## Result

ICE68 returns an error if an action type needed for an installation is invalid.

## Example

ICE68 posts the following warning if a custom action has the `msidbCustomActionTypeNoImpersonate` bit set in the `Type` field of the `CustomAction` table without the `msidbCustomActionTypeInScript` also set.

```
Even though custom action '[2]' is marked to be elevated (with attribute msidbCustomActionTypeNoImpersonate), it will not be elevated because it's not deferred (with attribute msidbCustomActionTypeInScript).
```

To fix this warning, include `msidbCustomActionTypeInScript` (0x400) if the custom action includes `msidbCustomActionTypeNoImpersonate` (0x800). Otherwise the installer ignores the `msidbCustomActionTypeNoImpersonate` attribute. For more information, see [Custom Action In-Script Execution Options](#).

ICE68 reports the following error for the example shown:

```
Invalid custom action type for action 'Action1'.
```

1027 is not a valid action type.

To fix this error, choose a valid custom action type.

[CustomAction Table](#) (partial)

Action	Type	Source	Target
--------	------	--------	--------

Action1	1027	Argument	Component1
---------	------	----------	------------

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE69

ICE69 checks that all substrings of the form [[\\$componentkey](#)] within a [formatted](#) string do not cross-reference components. A cross-component reference occurs when the [[\\$componentkey](#)] property of a formatted string refers to a component other than the component stored in the `Component_` column of your tables.

Problems with cross-component referencing arise from the way formatted strings are evaluated. If the component referenced with the [[\\$componentkey](#)] property is already installed and is not being changed during the current installation (for example, being reinstalled, moved to source, and so forth), the expression [[\\$componentkey](#)] evaluates to null, because the action state of the component in [[\\$componentkey](#)] is null. Similar problems can occur during upgrade and repair operations.

## Result

ICE69 returns an error if a [[\\$componentkey](#)] substring within a formatted string cross-references a component in another feature. ICE69 returns a warning if a [[\\$componentkey](#)] substring within a formatted string cross-references a component in the same feature. (The [FeatureComponents](#) table is used to determine this mapping. It must map to the same feature for the warning. Referencing components in parent features or referencing components in child features is considered to be an error.)

ICE69 reports an error if the [[#FileKey](#)] substring within a formatted string references a file which is not specified in the [File](#) table as belonging to the same component.

## Example

ICE69 reports the following for the examples shown.

```
WARNING: "Mismatched component reference. Entry 'Test' of th  
ERROR: "Mismatched component reference. Entry 'Shortcut2' of
```

To fix this error, do not cross-reference components. Change the [[\\$componentkey](#)] to match the component of the shortcut.

### Shortcut Table (partial)

Shortcut	Component_	Argument
Test	QuickTest	-v [\$Test]
Shortcut2	QuickTest	[\$Test2]

The [Verb](#) and [Extension](#) tables are special cases in that the Verb table references an extension that belongs to a component. An Extension, however, can belong to multiple components because the primary key for the extension table is made up of the Extension and Component\_ columns. You can logically have the following situation.

### Verb Table (partial)

Extension	Verb_	Argument
tst	open	-v [\$comp1][\$comp2]

### Extension Table (partial)

Extension	Component_
tst	comp1
tst	comp2

### FeatureComponents Table

Feature_	Component_
Feature1	QuickTest
Feature1	Test
Feature2	Test2

In this case, you must ensure that at least one of the [\$componentkey] properties evaluates to a non-null value. However, every [\$componentkey] property in the Argument column of the Verb table ([\$comp1] and [\$comp2] in the above example) must reference a possible component included with the extension associated with the verb. A reference like [\$comp3] would result in a warning from ICE69.

The [AppId table](#) has a similar situation to the Verb table. It uses the [Class table](#) for its component reference. In this case, the AppId table is validated in the same way as the Verb-Extension validation (now AppId-Class).

The Class table's Argument column is validated like the Shortcut, [Registry](#), and similar tables.

## **Table used during execution (only if found)**

[IniFile](#)

[RemoveIniFile](#)

[Registry](#)

[RemoveRegistry](#)

[ServiceControl](#)

[ServiceInstall](#)

[Shortcut](#)

[Verb](#)

[Extension](#)

[Class](#)

[AppId](#)

[Environment](#)

## **See Also**

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE70

ICE70 verifies that integer values for registry entries are specified correctly. Values of the form `##str`, `##%unexpanded str` are not validated. Values of the form `#xhex`, `#Xhex`, `#integer`, and `#[property]` are validated. The following table provides a brief overview.

Value	Validation
<code>##str</code>	valid
<code>##%unexpanded str</code>	valid
<code>#xHex,#XHex</code>	Validate for valid hex characters (0-9,a-f,A-F). Properties are allowed here.
<code>#+int, #-int, #int</code>	Validate for valid numeric characters (0-9). Properties are allowed here.

The syntax for an integer value to be entered into the registry is `#integer` where `integer` is numerical.

## Result

ICE70 reports an error if integer values for registry entries are not specified correctly.

## Example

ICE70 reports the following errors for the given example.

```
The value #12xz34 is an invalid numeric value for registry e
```

To fix this error: If you want the value to be numeric, change the value to use all numeric characters. If you want the value to be a string, it must be preceded by two '#' (`##`) instead of just one.

```
The value #xz34 is an invalid hexadecimal value for registry
```

To fix this error: Valid hexadecimal characters are 0-9, A-F, and a-f. Only these characters can follow the #x (or #X).

### Registry table (partial)

Registry	Value
Reg1	#12xz34
Reg2	#xz34

### Remarks

- #[myproperty] is valid.
- #[myproperty is not valid (missing ending bracket).
- #[myprop1] [myprop2 is valid. (Even though the last one is missing the ending bracket, myprop1 could evaluate to #str so you would have ##str [myprop2, which is valid
- #]myproperty[ is not valid
- Any embedded property in a value string cannot be in [\$compkey], [#filekey], or [!filekey] form because these are not numeric. However, there is one exception, #[myproperty] [\$compkey] (or [#filekey] or [!filekey]) is valid because, as with the preceding, [myproperty] can evaluate to #str.

### See Also

#### [ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE71

ICE71 verifies that the **Media table** contains an entry with DiskId equal to 1. (Windows Installer assumes that the .msi package is on disk 1.)

## Result

ICE71 returns an error if the Media table does not contain an entry with DiskId equal to 1.

## Example

ICE71 reports the following error for the example shown.

```
The Media table requires an entry with DiskId=1. First DiskId
```

To fix this error, change the DiskId of the entry where the package is stored to 1.

### Media Table (partial)

DiskId
2

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE72

ICE72 verifies that non-built-in custom actions are not used in the [AdvtExecuteSequence table](#). Specifically, only type 19, type 35, and type 51 custom actions are allowed in the AdvtExecuteSequence table. If other custom actions are used, advertisement may not behave as expected.

## Result

ICE72 returns an error if the AdvtExecuteSequence table uses custom actions other than type 35, type 51, and type 19.

## Example

ICE72 reports the following error for the example shown:

```
Custom Action 'CA1' in the AdvtExecuteSequence table is not
```

To fix the error, remove 'CA1' from the AdvtExecuteSequence table.

### [AdvtExecuteSequence Table \(partial\)](#)

Action
CA1
CA35

### [CustomAction Table \(partial\)](#)

Action	Type
CA1	1
CA35	35

## Tables used during execution

[AdvtExecuteSequence Table](#)

[CustomAction Table](#)

## See Also

[Custom Action Type 19](#)

[Custom Action Type 35](#)

[Custom Action Type 51](#)

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE73

ICE73 verifies that your package does not reuse package codes, upgrade codes, or product codes of the Windows Installer SDK samples. Packages should never reuse the package, upgrade, or product codes of another product.

## Result

ICE73 outputs a warning if your product's package reuses a package or product code of a Windows Installer SDK sample.

## Example

ICE73 reports the following errors for the example shown:

```
This package reuses the '{80F7E030-A751-11D2-A7D4-006097C99860}'
This package reuses the '{000C1101-0000-0000-C000-000000000047}'
This package reuses the '{8FC7****-88A0-4b41-82B8-8905D4AA904C}'
```

**Note** The asterisks (\*\*\*\*) in the GUID represent the range of GUIDs reserved for subsequent Windows Installer SDK packages.

To fix the errors, generate a new unique GUID for your package's product and package codes. You will also need a new unique GUID for your package's upgrade code.

### Summary Information Stream (partial)

Property	Value
PID_REVNUMBER	{000C1101-0000-0000-C000-000000000047}

### Property Table (partial)

Property	Value
ProductCode	{80F7E030-A751-11D2-A7D4-006097C99860}
UpgradeCode	{8FC70000-88A0-4b41-82B8-8905D4AA904C}

## See Also

[Package Codes](#)

[Product Codes](#)

[Revision Number Summary Property](#)

[UpgradeCode Property](#)

[ProductCode Property](#)

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE74

ICE74 verifies that the **FASTOEM** property has not been authored into the [Property table](#).

The **FASTOEM** property enables OEMs to reduce the time required to install Windows Installer applications for the first time. It cannot be used after the first install. The **FASTOEM** property must not be authored in the Property table because this interferes with subsequent installations for the maintenance, removal, or repair of the application.

ICE74 also verifies that the **UpgradeCode** property is authored into the [Property table](#), and that its value is not a null GUID, {00000000-0000-0000-0000-000000000000}.

## Result

ICE74 can post the following errors.

ICE74 error	Description
The <b>FASTOEM</b> property cannot be authored in the Property table.	The <b>FASTOEM</b> property has been set in the Property table.
'[2]' is not a valid <b>UpgradeCode</b> .	A null GUID has been entered for the <b>UpgradeCode</b> property in the Property table.

ICE74 can post the following warning.

ICE74 warning	Description
The <b>UpgradeCode</b> property is not authored in the Property table. It is strongly recommended that authors of installation packages specify an <b>UpgradeCode</b> for their application.	The <b>UpgradeCode</b> property is not authored in the Property table.

## Example

ICE74 reports the following error if the **FASTOEM** property is set: The FASTOEM

property cannot be authored in the Property table.

[Property Table](#) (partial)

Property	Value
<b>FASTOEM</b>	1

To fix this error remove the **FASTOEM** property from the Property Table.

## See Also

[FASTOEM property](#)

[Property table](#)

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE75

ICE75 verifies that all [Custom Action Type 17 \(DLL\)](#), [Custom Action Type 18 \(EXE\)](#), [Custom Action type 21 \(JScript\)](#), and [Custom Action Type 22 \(VBScript\)](#) custom actions are sequenced after the [CostFinalize action](#). These types of custom action use an installed file as their source. ICE75 checks the [InstallUISequence Table](#), [InstallExecuteSequence Table](#), [AdminUISequence Table](#), and [AdminExecuteSequence Table](#). Note that the CostFinalize action is required in these sequence tables.

## Result

ICE75 posts an error if it finds a custom action using an installed file as a source file that is not sequenced after the CostFinalize action.

## Example

ICE75 reports the following errors for the example shown:

```
CostFinalize is missing from 'AdminUISequence'. CA_FileExe is a custom action whose source is an installed file. It must be sequenced after the CostFinalize action.
```

```
CA_FileDLL is a custom action whose source is an installed file. It must be sequenced after the CostFinalize action in the AdminExecuteSequence table
```

### CustomAction Table (partial)

Action	Type	Source
CA_FileExe	18	FileExe
CA_FileDLL	17	FileDLL

### AdminUISequence Table (partial)

Action	Sequence
CA_FileExe	1100

---

## AdminExecuteSequence Table (partial)

Action	Sequence
CA_FileDLL	800
CostFinalize	1000

To fix the errors, sequence the custom actions after the CostFinalize action.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE76

ICE76 verifies the use of the SFP (WFP) catalog within Windows Installer packages for Windows Me. This ICE also verifies that no files in the BindImage [table](#) reference SFP catalogs.

Windows File Protection requires an exact match between the file and the signature embedded in the catalog file. Files that reference a SFP catalog must not be listed in the BindImage table because the effect of the [BindImage action](#) on these files differs between computers. Files referenced by SFP catalogs must be in components that are permanent or installed locally.

## Result

ICE76 posts an error for each file in the [BindImage table](#) that is also in the [FileSFPCatalog table](#).

ICE76 outputs an error if a file in the FileSFPCatalog table belongs to a component with any of the following true:

- msidbComponentAttributesPermanent is not set in the Attributes column of the [Component table](#).
- msidbComponentAttributesSourceOnly is set in the Attributes column of the Component table.
- msidbAttributesOptional is set in the Attributes column of the Component table.

## Example

ICE76 reports the following error for the example:

File 'File1' references a SFP catalog. Therefore it cannot be installed.

[FileSFPCatalog Table](#) (partial)

<a href="#">File_</a>	<a href="#">SFPCatalog_</a>

File1	Catalog1.Cat
-------	--------------

### BindImage Table (partial)

File_
File1

To fix this do not enter any files that reference SFP catalogs into the BindImage table.

### See Also

[BindImage Table](#)

[Component Table](#)

[FileSFPCatalog Table](#)

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE77

ICE77 verifies that custom actions with the `msidbCustomActionTypeInScript` bit set are sequenced after the [InstallInitialize action](#) and before the [InstallFinalize action](#). ICE77 checks the sequence in the [InstallExecuteSequence table](#) and [AdminExecuteSequence table](#).

## Result

ICE77 posts an error if an in-script custom action is sequenced before the `InstallInitialize` action or after the `InstallFinalize` action.

ICE77 posts an error if the `InstallInitialize` action or the `InstallFinalize` action is missing.

## Example

ICE77 reports the following errors for the example:

```
InstallFinalize is missing from 'InstallExecuteSequence'.  
CA_InScriptInstall is a in-script custom action. It must be  
before the InstallFinalize action.
```

```
CA_InScriptAdmin is a in-script custom action. It must be s  
in between the InstallInitialize action and the InstallFinal  
in the AdminExecuteSequence Sequence table.
```

### CustomAction Table (partial)

Action	Type
CA_InScriptInstall	1025
CA_InScriptAdmin	1026

### InstallExecuteSequence Table (partial)

Action	Sequence
--------	----------

CA_InScriptInstall	2000
InstallInitialize	1500

### [AdminExecuteSequence Table \(partial\)](#)

<b>Action</b>	<b>Sequence</b>
CA_InScriptAdmin	1400
InstallInitialize	1500
InstallFinalize	6600

To fix the errors, sequence the in-script custom actions after the InstallInitialize action and before the InstallFinalize action. The InstallInitialize and InstallFinalize actions must be present in the InstallExecuteSequence table and the AdminExecuteSequence table.

## **See Also**

### [ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE78

ICE78 verifies that the [AdvtUISequence table](#) either does not exist or is empty. This is required because no user interface is allowed during advertising.

## Result

ICE78 posts an error if the AdvtUISequence table exists and is not empty.

## Example

ICE78 reports the following error for the example:

Action 'Action1' found in AdvtUISequence table. No UI is allowed during advertising. Therefore AdvtUISequence table must be empty or not present.

### [AdvtUISequence table](#)(partial)

Action	Condition	Sequence
Action1	TRUE	1

To fix the error, either remove "Action1" from the table, or remove the table.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE79

ICE79 validates the references to components and features entered in the database fields using the [Condition](#) data type.

## Result

ICE79 posts two warnings.

ICE79 warning	Description
Database is missing _Validation table. Could not completely check property names.	Database is missing <a href="#">_Validation table</a> .
Error retrieving values from column [2] in table [1]. Skipping Column.	Error retrieving value.

ICE79 posts two errors.

ICE79 error	Description
Component '%ls' referenced in column '%s'. '%s' of row %s is invalid.	An invalid component reference was found.
Feature '%ls' referenced in column '%s'. '%s' of row %s is invalid.	An invalid feature reference was found

## Example

ICE79 reports the following errors for the example:

```
Component 'NoSuchComponent' referenced in column
'InstallExecuteSequence'. 'Condition' of row Custom2 is inval
Feature 'NoSuchFeature' referenced in column
'InstallExecuteSequence'. 'Condition' of row Custom1 is inval
```

In this example, NoSuchComponent is absent from the [Component table](#)

and NoSuchFeature is absent from the [Feature table](#).

### **InstallExecuteSequence Table (partial)**

<b>Action</b>	<b>Condition</b>
Custom1	TESTACTION=1046 AND &NoSuchFeature>2
Custom2	TESTACTION=146 AND \$NoSuchComponent>2

To fix these errors, enter valid records for NoSuchFeature and NoSuchComponent in the Feature and Component tables.

### **See Also**

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE80

ICE80 validates that the value of the **Template Summary** Property (PID\_TEMPLATE) correctly specifies "Intel64", "x64", or "Intel" depending on the presence of 64-bit components or custom action scripts. ICE80 checks the **Component Table** for any components with the msidbComponentAttributes64bit attribute and checks the **CustomAction Table** for any scripts with the msidbCustomActionType64BitScript attribute. ICE80 verifies that a package with "Intel64" or "x64" in its **Template Summary** Property also has a **Page Count Summary** Property (PID\_PAGECOUNT) of at least 150.

ICE80 also validates that the language ID specified by the **ProductLanguage** property must be contained in the **Template Summary** Property.

For more information, see [Windows Installer on 64-bit Operating Systems](#).

## Result

ICE80 posts the following errors.

Error	Description
This package contains 64 bit component '[1]' but the <b>Template Summary</b> Property does not contain Intel64 or x64.	The Component Table contains a component with the msidbComponentAttributes64bit attribute and the <b>Template Summary</b> Property does not contain Intel64 or x64.
This package contains 64 bit custom action script '[1]' but the <b>Template Summary</b> Property does not contain Intel64 or x64.	CustomAction Table contains a script custom action with the msidbCustomActionType64BitScript but the <b>Template Summary</b> Property does not contain Intel64 or x64.
Bad value in Summary Information Stream for %s.	Returned for PID_TEMPLATE property if that property is an empty string or not a VT_LPSTR type. Returned for PID_PAGECOUNT if that

	property it is not a VT_I4 type.
This package is marked with Intel64 or x64 but it has a schema less than 150.	The PID_TEMPLATE property of the package is Intel64 or x64, but its PID_PAGECOUNT property is less than 150.
This 32Bit Package is using 64 bit property [1]	A 32-bit package is using a 64-bit property.
This 32Bit Package is using 64 bit Locator Type in RegLocator table entry [1]	A 32-bit package contains msidbLocatorType64bit in the Type field of the <a href="#">RegLocator table</a> .
This 64BitComponent [1] uses 32BitDirectory [3]	A 64-bit component is using a 32-bit directory.
This 32BitComponent [1] uses 64BitDirectory [3]	A 32-bit component is using a 64-bit directory.
The ' <b>ProductLanguage</b> ' property in the Property table has a value of '[2]', which is not contained in the Template Summary Property stream.	The value of the <b>ProductLanguage</b> property is not listed in the <a href="#">Template Summary</a> property.

## See Also

[ICE Reference](#)

[Windows Installer on 64-bit Operating Systems](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE81

ICE81 validates the [MsiDigitalCertificate table](#), [MsiDigitalSignature table](#), [MsiPatchCertificate table](#), and [MsiPackageCertificate Table](#). This ICE custom action posts warnings for digital certificates that are unused or unreferenced, and it posts an error when the signed object does not exist or when the signed object's cabinet does not point to external data.

Note that ICE03 verifies that the entry in the Table column in the MsiDigitalSignature table is "Media."

## Result

ICE81 posts the following warnings for unused or unreferenced Digital Certificates.

ICE81 warning	Description
No reference to any of the records in the MsiDigitalCertificate table could be found in MsiDigitalSignature, MsiPackageCertificate, or MsiPatchCertificate tables.	This warning is returned if all records are unused.
No reference to the Digital Certificate [1] could be found in MsiDigitalSignature, MsiPackageCertificate, or MsiPatchCertificate tables.	This warning is returned if some records, but not all, are unused.

ICE81 posts the following errors.

ICE81 error	Description
Media Table does not exist. Hence all the entries in MsiDigitalSignature are incorrect	The signed object does not exist. This error is returned if the Media table does not exist but MsiDigitalSignature has entries.
Missing signed object [2] in Media Table	The signed object [2] does not exist. This error is returned if the Media table exists, but this entry in

	MsiDigitalSignature is not present in Media table.
The entry in table [1] with key [2] is signed. Hence the cabinet should point to an object outside the package (the value of Cabinet should NOT be prefixed with #)	The signed object's cabinet does not point to external data. [1] is table name. [2] is key in the Media table.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE82

ICE82 validates that the [RegisterProduct Action](#), [RegisterUser Action](#), [PublishProduct Action](#), and [PublishFeatures Action](#) are all present in the [InstallExecuteSequence table](#). The package is validated if all the actions are present.

ICE82 posts a warning if there are two actions with the same sequence number listed in the [InstallExecuteSequence](#), [InstallUISequence](#), [AdminExecuteSequence](#), [AdminUISequence](#), or [AdvtExecuteSequence tables](#) .

## Result

ICE82 posts the following warnings.

<b>ICE82 warning</b>	<b>Description</b>
The <a href="#">InstallExecuteSequence table</a> does not contain the set of actions mentioned below: Actions Missing:  Publish Features  Publish Product  Register Product  Register User	ICE82 custom action posts a warning if all four actions are absent.
This action [1] has duplicate sequence number [2] in the table [3].	ICE82 posts a warning if there are two actions with the same sequence number listed in the <a href="#">InstallExecuteSequence</a> , <a href="#">InstallUISequence</a> , <a href="#">AdminExecuteSequence</a> , <a href="#">AdminUISequence</a> , or <a href="#">AdvtExecuteSequence tables</a> .

ICE82 posts the following errors.

---

ICE82 error	Description
<p>The InstallExecuteSequence should either contain all of the actions mentioned below or none of them</p> <p>Actions Present</p> <p>&lt;List of actions present&gt;</p> <p>Actions Missing:</p> <p>&lt;List of actions missing&gt;</p>	<p>ICE82 posts an error if some of the four actions are present and others are absent.</p>

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE83

ICE83 validates the [MsiAssembly table](#). This ICE custom action posts an error if the key path for a component containing a Win32 assembly is set to the manifest file. Explicitly the error is posted if the value entered in the KeyPath field of the [Component table](#) equals the value entered in the File\_Manifest field of the MsiAssembly table. This ICE custom action posts an error if there is at least one record in the MsiAssembly table and the [InstallExecuteSequence table](#) does not contain both the [MsiPublishAssemblies Action](#) and [MsiUnpublishAssemblies Action](#).

## Result

ICE83 posts the following errors.

ICE83 error	Description
The key path for Win32 SXS Assembly (Component_[1]) SHOULD NOT be its manifest file	ICE83 posts this error when the KeyPath field for a Win32 Assembly is set to its manifest file (Component.KeyPath == MsiAssembly.File_Manifest). [1] is KeyPath in Component table
Both MsiPublishAssemblies AND MsiUnpublishAssemblies actions MUST be present in InstallExecuteSequence table.	ICE83 posts this error when there is at least one entry in the MsiAssembly table but the InstallExecuteSequence table does not contain both the MsiAssemblyPublish action and the MsiAssemblyUnpublish action.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE84

ICE84 checks the [AdvtExecuteSequence table](#), [AdminExecuteSequence table](#), and the [InstallExecuteSequence table](#) to verify that the following [standard actions](#) have not been set with conditions in the Condition field.

- [CostInitialize action](#)
- [CostFinalize action](#)
- [FileCost action](#)
- [InstallValidate action](#)
- [InstallInitialize action](#)
- [InstallFinalize action](#)
- [ProcessComponents action](#)
- [PublishFeatures action](#)
- [PublishProduct action](#)
- [RegisterProduct action](#)
- [UnpublishFeatures action](#)

If conditions are found, ICE84 posts a warning.

## Result

ICE84 posts the following warning.

<b>ICE84 error</b>	<b>Description</b>
Action '[1]' found in %s table is a required action with a condition.	A required action has been authored with a condition.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE85

ICE85 validates that the SourceName column of the [MoveFile table](#) is a valid long file name. This field may contain wildcard characters (\* and ?).

## Result

ICE85 posts the following errors.

ICE85 error	Description
SourceName '[2]' found in the MoveFile table is of bad format.	The SourceName field in the MoveFile table is not a valid long file name

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE86

ICE86 issues a warning if the package uses the [AdminUser](#) property in database column of the [Condition](#) type. Package authors should use the [Privileged](#) property in conditional statements.

## Result

ICE86 posts the following warning.

ICE86 warning	Description
Property `%s` found in column `%s`. `%s` in row %s. `Privileged` property is often more appropriate.	<b>AdminUser</b> property was used in a Condition field.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE87

ICE87 validates that the following properties have not been authored in the [Property Table](#). These properties should instead be set on a command line.

- [ADDLOCAL](#) property
- [REMOVE](#) property
- [ADDSOURCE](#) property
- [ADDDEFAULT](#) property
- [REINSTALL](#) property
- [ADVERTISE](#) property
- [COMPADDLOCAL](#) property
- [COMPADDSOURCE](#) property
- [FILEADDLOCAL](#) property
- [FILEADDSOURCE](#) property
- [FILEADDDEFAULT](#) property

## Result

ICE87 posts the following warning.

ICE87 warning	Description
The property '[1]' shouldn't be authored into the Property table. Doing so might cause the product to not be uninstalled correctly	The specified property should not be set in the <a href="#">Property table</a> . Set the property on a command line instead.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE88

ICE88 validates that the directory referenced in the DirProperty column of the [IniFile table](#) exists in the Windows Installer package. ICE88 issues a warning if the DirProperty value does not represent a property in the Directory, AppSearch, or Property tables, certain [system folder properties](#), or a property set by a type 51 custom action.

ICE88 scans the following tables and properties.

- [Directory Table](#)
- [AppSearch Table](#)
- [Property Table](#)
- [CustomAction Table](#) , where the custom action is a [Custom Action Type 51](#)
- [ProgramFilesFolder Property](#)
- [CommonFilesFolder Property](#)
- [SystemFolder Property](#)
- [ProgramFiles64Folder Property](#)
- [CommonFiles64Folder Property](#)
- [System64Folder Property](#)

## Result

ICE88 posts the following warning.

ICE88 Warning	Description
In the IniFile table entry (IniFile=) [3] the DirProperty=[1] is not found in Directory/Property/AppSearch/CA-Type51 tables and it is not one of the installer properties.	For each record in the IniFile table, ICE88 reads the value in the DirProperty column. ICE88 scans for the value in the <a href="#">Directory Table</a> , <a href="#">AppSearch Table</a> , and <a href="#">Property Table</a> and also scans the list of properties set by the installer. ICE88 posts this warning if the value cannot be found in

the scan.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE89

ICE89 validates that the value in the ProgId\_Parent column in [ProgId table](#) is a valid foreign key into the ProgId column in ProgId table. Every ProgId parent should have a record in the ProgId table.

## Result

ICE89 posts the following errors.

<b>ICE89 error</b>	<b>Description</b>
The ProgId_Parent '[1]' in the ProgId table is not a valid ProgId.	There is a ProgId parent specified that is not listed in the ProgId table.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE90

ICE90 posts a warning if it finds that a shortcut's directory has been specified as a public property. The names of [Public Properties](#) are written in uppercase letters. A shortcut specified by a public property may not work if the value of the [ALLUSERS](#) property changes.

This ICE custom action validates the Shortcut table and uses the Directory table. If the Directory table is not present, it returns without validating the Shortcut table and posts no errors or warnings.

## Result

ICE90 posts the following warning.

ICE90 error	Description
The shortcut '[1]' has a directory that is a public property (ALL CAPS) and is under user profile directory. This results in a problem if the value of the <a href="#">ALLUSERS</a> property changes in the UI sequence.	A shortcut's directory has been specified as a public property.

## Example

ICE90 reports the following warning for the example:

```
The shortcut 'Shortcut1' has a directory that is a public property and is under user profile directory. This results in a problem if the ALLUSERS property changes in the UI sequence.
```

In this example, MYDIR is under a users profile. ICE90 posts a warning because the location of the target directory is specified by a public property, MYDIR. A user may change MYDIR or **ALLUSERS** property. If **ALLUSERS** is set for the per-machine [installation context](#), and MYDIR is under a users profile, the shortcut file in MYDIR are copied under the "All Users" profile and not a particular user's profile. If **ALLUSERS** is set for the per-user installation context, the shortcut file in MYDIR is copied into

a particular user's profile and is not available to other users.

#### [Shortcut Table](#) (partial)

<b>Shortcut</b>	<b>Directory_</b>
Shortcut1	MYDIR

#### [Directory Table](#) (partial)

<b>Directory</b>	<b>Directory_Parent</b>
MYDIR	ProgramMenuFolder

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE91

ICE91 posts a warning if a file, .ini file, or shortcut file is installed into a per-user only directory. These warnings are harmless if the package is only used for installation in the per-user [installation context](#) and never used for per-machine installations.

Files, .ini files, or shortcuts in per-user only directories are installed into a particular user's profile. Even if the user sets the **ALLUSERS** property for a per-machine installations, files, .ini files, or shortcuts in per-user only directories are not copied in to the "All Users" profile and are not available to other users. The per-user only directories do not vary with the **ALLUSERS** property. The following is a list of the per-user only directories:

- AppDataFolder
- FavoritesFolder
- NetHoodFolder
- PersonalFolder
- PrintHoodFolder
- RecentFolder
- SendToFolder
- MyPicturesFolder
- LocalAppDataFolder

## Result

ICE91 posts the following warnings.

ICE91 warning	Description
The file '[1]' will be installed to the per user directory '[2]' that doesn't vary based on <b>ALLUSERS</b> value. This file won't be copied to each user's profile even if a per machine installation is desired.	The file is installed into a per-user only directory. It is not installed into each user's profile during a per-machine installation.

The IniFile '[1]' will be installed to the per user directory '[2]' that doesn't vary based on **ALLUSERS** value. This file won't be copied to each user's profile even if a per machine installation is desired.

The .ini file is installed into a per-user only directory. It is not installed into each user's profile during a per-machine installation.

The shortcut '[1]' will be installed to the per user directory '[2]' that doesn't vary based on **ALLUSERS** value. This file won't be copied to each user's profile even if a per machine installation is desired.

The shortcut is installed into a per-user only directory. It is not installed into each user's profile during a per-machine installation.

## Example

ICE91 reports the following warnings for the example:

```
The file 'File1' will be installed to the per user directory
The IniFile 'IniFile1' will be installed to the per user dir
The shortcut 'Shortcut1' will be installed to the per user c
```

### File Table (partial)

File	Component_
File1	C1

### Component Table (partial) (partial) (partial) (partial)

Component	Directory_
C1	MyDir

### IniFile Table

IniFile	DirProperty
IniFile1	MyIniDir

---

## Shortcut Table

Shortcut	Directory_
Shortcut1	MyShortcutDir

## Directory Table

Directory	Directory_Parent
MyDir	FavoritesFolder
MyIniDir	LocalAppDataFolder
MyShortcutDir	PersonalFolder

## See Also

### [ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE92

ICE92 verifies that a component without a Component Id GUID is not also specified as a permanent component. This ICE custom action checks the [Component Table](#) for components without a [GUID](#) specified in the ComponentId field and verifies that the msidbComponentAttributesPermanent flag has not been set in the Attributes field. ICE92 also verifies that no component has both the msidbComponentAttributesPermanent and msidbComponentAttributesUninstallOnSupersedence attributes.

If the ComponentId column is null, the installer does not register the component and the component cannot be removed or repaired by the installer.

## Result

ICE92 posts the following error.

ICE92 error	Description
The Component '[1]' has no ComponentId and is marked as permanent.	The entry for this component in the Component table has null in the ComponentId column and has msidbComponentAttributesPermanent in the Attributes column.

ICE92 posts the following warning.

ICE92 warning	Description
The Component '[1]' is marked as permanent and uninstall-on-supersedence. The uninstall-on-supersedence attribute will be ignored because the	The entry for this component in the Component table has both the msidbComponentAttributesPermanent and msidbComponentAttributesUninstallOnSupersedence attributes specified.

component is permanent.	
-------------------------	--

## Example

ICE92 reports the following error for the example:

The Component 'Component1' has no ComponentId and is marked

[Component Table](#) (partial)

Component	ComponentId	Directory_	Attributes	KeyPath
Component1		DirectoryA	16	FileA

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE93

ICE93 issues a warning if a custom action uses the same name as a standard action. For a list of all standard action names, see [Standard Actions Reference](#).

## Result

ICE93 posts the following warning.

<b>ICE93 warning</b>	<b>Description</b>
The Custom action '[1]' uses the same name as a standard action.	There is an identifier in the Action column of the CustomAction table that is the name of a Windows Installer standard action.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE94

ICE94 checks the [Shortcut table](#), [Feature table](#), and [MsiAssembly table](#) and posts a warning if there are any unadvertised shortcuts pointing to an assembly file in the global assembly cache. If the entry in the Target field of the Shortcut table is not a feature in the Feature table, the shortcut is unadvertised. If the entry in the Component\_ field of the Shortcut table is also listed in the MsiAssembly table, the shortcut points to an assembly file. If the entry in the File\_Application field in the MsiAssembly table is empty, the assembly file is in the global assembly cache.

## Result

ICE94 posts the following warning.

ICE94 warning	Description
The non-advertised shortcut '[2]' points to an assembly file in the global assembly cache.	An unadvertised shortcut is pointing to an assembly file in the global assembly cache.

## Example

ICE94 reports the following error for the example:

The non-advertised shortcut 'shortcut1' points to an assembly file.

### Shortcut Table (partial)

Shortcut	Component_	Target
shortcut1	c1	[file1]
shortcut2	c2	feature1
shortcut3	c3	[file2]

## Feature Table (partial)

Feature
feature1

## MsiAssembly Table (partial)

Component_	File_Application
c1	
c2	
c3	fa1

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE95

ICE95 checks the [Control table](#) and [BBControl table](#) to verify that the billboard controls fit onto all the billboards.

## Result

If any of the following are true, a billboard control fails to fit on a billboard. ICE95 posts the following warnings.

ICE95 warning	Description
The BBControl item '[1],[2]' in the BBControl table does not fit in all the billboard controls in the Control table. The X coordinate exceeds the boundary of the minimum billboard control width %s	The control's X coordinate is outside the width of the billboard.
The BBControl item '[1],[2]' in the BBControl table does not fit in all the billboard controls in the Control table. The Y coordinate exceeds the boundary of the minimum billboard control height %s	The control's Y coordinate is below the bottom of the billboard.
The BBControl item '[1],[2]' in the BBControl table does not fit in all the billboard controls in the Control table. The X coordinate and the width combined together exceeds the minimum billboard control width %s	The control's X coordinate plus the control's width is outside the width of the billboard.
The BBControl item '[1],[2]' in the BBControl table does not fit in all the billboard controls in the Control table. The Y coordinate and the height combined together exceeds the minimum billboard control height %s	The control's Y coordinate plus the control's height is below the bottom of the billboard.

## Example

ICE95 reports the following warnings for the example:

```
The BBControl item 'billboard1.bbcontrol1' in the BBControl
```

The BBControl item 'billboard1.bbcontrol2' in the BBControl  
The BBControl item 'billboard1.bbcontrol3' in the BBControl  
The BBControl item 'billboard1.bbcontrol4' in the BBControl

Control Table (partial) (partial)

<b>Control</b>	<b>Type</b>	<b>Width</b>	<b>Height</b>
control1	Billboard	300	100
Control2	Billboard	100	300

BBControl table

<b>Billboard_</b>	<b>BBControl</b>	<b>X</b>	<b>Y</b>	<b>Width</b>	<b>Height</b>
billboard1	bbcontrol1	200	0	100	100
billboard1	bbcontrol2	0	200	100	100
billboard1	bbcontrol3	50	0	100	50
billboard1	bbcontrol4	0	50	50	100

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE96

ICE96 verifies that the [PublishFeatures action](#) and the [PublishProduct action](#) are entered in the [AdvExecuteSequence table](#). A package cannot be advertised without these actions.

## Result

ICE96 posts the following warnings.

<b>ICE96 warning</b>	<b>Description</b>
The PublishFeatures action is required in the AdvExecuteSequence table.	A package cannot be advertised without the PublishFeatures action included in the AdvExecuteSequence table.
The PublishProduct action is required in the AdvExecuteSequence table.	A package cannot be advertised without the PublishProduct action included in the AdvExecuteSequence table.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE97

ICE97 verifies that two components do not isolate a shared component to the same directory.

## Result

ICE97 posts the following warnings.

ICE97 Warning	Description
This component [1] installs the Shared component into the same directory [2] as another, which breaks component rules if both (or more) components are selected for install.	Two components must not isolate a shared component to the same directory.

For example, Component1 and Component2, which share ComponentShared, are installed to the same directory. Both specify ComponentShared as an isolated component. Because of the isolation, the files in ComponentShared are copied twice into the Directory\_ reference for Component1 and Component2. The components now have one reference count on the copy of files. This is in violation of the Installer component rules. If Component1 is uninstalled, the isolated components files are removed and Component2 is broken.

## See Also

[ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE98

ICE98 verifies the description field of the [ODBCDataSource Table](#) for an ODBC data source. It uses the [SQLValidDSN](#) function to check that only valid characters are used, and that the description does not exceed the maximum allowed length.

## Result

ICE98 posts the following warnings.

<b>ICE98 warning</b>	<b>Description</b>
The data source name is invalid.	The value in the Description column of the ODBCDataSource Table either contains invalid characters or is too long, which means that it exceeds the SQL_MAX_DSN_LENGTH of 32.

## Example

ICE98 reports the following warnings for the example:

```
The data source name is invalid: !  
The data source name is invalid: <String of length > 32>
```

ODBCDataSource Table (partial)

<b>DataSource</b>	<b>Description</b>
BadChar	!
TooLong	<String of length > 32>

## See Also

[ICE Reference](#)

## ODBCDataSource Table

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE99

ICE99 verifies that no property name entered in the [Directory](#) table duplicates a name reserved for the public or private use of the Windows Installer.

## Result

ICE99 posts the following error.

ICE99 error	Description
The directory name: [1] is the same as one of the MSI Public Properties and can cause unforeseen side effects.	The value in the Directory column of the Directory table duplicates a property name reserved by the Windows Installer.

## Example

ICE99 reports the following error for the example:

```
CustomActionData is the same as one of the MSI Public Properties
```

[Directory](#) (partial)

Directory	Directory_Parent	DefaultDir
CustomActionData		

To correct this warning you should change the name of CustomActionData.

## See Also

[ICE Reference](#)  
[Directory Table](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE 100

ICE100 checks the authoring of the [MsiEmbeddedUI](#) table and the [MsiEmbeddedChainer](#) table.

## Result

ICE100 posts the following errors.

ICE100 error	Description
Too many DLLs marked as UI: [1]	The Attributes column of the MsiEmbeddedUI table contains too many DLLs marked as a user interface DLL. Only one DLL should be marked as a user interface DLL.
UI DLL cannot have null/zero message filter: [1]	A DLL marked as a user interface DLL in the Attributes column of the MsiEmbeddedUI table has null in the MessageFilter field. If a row references a user interface DLL, the value in MessageFilter field should not be null.
Resource DLLs must have null/zero message filters: [1]	A DLL not marked as a user interface DLL in the MsiEmbeddedUI table must have null in the MessageFilter field. This field should be null if a row references a resource file and the value of Attributes is null.
Multiple DLLs marked with this filename: [2] (index: [1])	Multiple DLLs in the MsiEmbeddedUI table have the same file name. The FileName column should not contain duplicate names.

ICE100 posts the following warning.

ICE100 warning	Description
Multiple chainers exist in MsiEmbeddedChainer table. Please ensure	There are multiple entries in the MsiEmbeddedChainer table. Only one embedded chainer can be launched. If multiple entries are both conditioned to run, it is undefined which entry will

that only one is  
conditioned to run.

run. If the user is sure that only one entry's condition  
can resolve to true, this warning can be ignored.

## See Also

[ICE Reference  
Directory Table](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE 101

Verifies that no value in the Feature column of the [Feature](#) table exceeds a maximum length of 38 characters.

**Windows Installer 4.5 or earlier:** Not supported. This ICE is available beginning with Windows Installer 5.0.

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE 102

Validates the [MsiServiceConfig](#) and [MsiServiceConfigFailureActions](#) tables.

Verifies that the value in the Event column in the MsiServiceConfig and MsiServiceConfigFailureActions tables is msidbServiceConfigEventInstall, msidbServiceConfigEventUninstall, or msidbServiceConfigEventReinstall. Verifies that the value in the Component\_ column in the MsiServiceConfig and MsiServiceConfigFailureActions tables is a valid key into the [Component Table](#).

Verifies that the values in the ConfigType column of the MsiServiceConfig table is one of the allowed values.

Verifies that the values in the Actions and the Delayed Actions columns of the MsiServiceConfigFailureActions table are separated by [~]. Verifies that the number of values listed in the Actions column and the number of values listed in the Delayed Actions column in each row of the MsiServiceConfigFailureActions table are the same.

If the value in the ConfigType field in the MsiServiceConfig table is SERVICE\_CONFIG\_DELAYED\_AUTO\_START the value in the Argument field must be 0 or 1.

If the value in the ConfigType field in the MsiServiceConfig table is SERVICE\_CONFIG\_REQUIRED\_PRIVILEGES\_INFO the value in the Argument field must contain a valid string of [Privilege Constants](#).

If the value in the ConfigType field in the MsiServiceConfig table is SERVICE\_CONFIG\_SERVICE\_SID\_INFO the value in the Argument field must be SERVICE\_SID\_TYPE\_NONE, SERVICE\_SID\_TYPE\_RESTRICTED, or SERVICE\_SID\_TYPE\_UNRESTRICTED.

If the value in the ConfigType field in the MsiServiceConfig table is SERVICE\_CONFIG\_PRESHUTDOWN\_INFO the value in the Argument field must be positive or empty. If the value in the ConfigType field in the MsiServiceConfig table is

SERVICE\_CONFIG\_FAILURE\_ACTIONS\_FLAG the value in the Argument field must contain 0 or 1.

**Windows Installer 4.5 or earlier:** Not supported. This ICE is available beginning with Windows Installer 5.0.

## Result

ICE102 posts the following errors.

ICE102 error	Description
In the MsiServiceConfig table entry (MsiServiceConfig = ) [1], ConfigType = %d is not a valid parameter. It should be between 3 and 7.	The Config field of the MsiServiceConfig table does not contain one of the allowed values.
In the MsiServiceConfig table entry (MsiServiceConfig = ) [1], Argument = %s is not a valid %s parameter. It should be %s.	The Argument field of the MsiServiceConfig table does not contain one of the allowed values.
In the MsiServiceConfigFailureActions table entry (MsiServiceConfigFailureActions = ) [1], Actions=[3] is not a valid parameter. It should be a list of null-separated non-negative integers.	The Actions field of the MsiServiceConfigFailureActions does not contain an array of positive integers. Separate the values in the array by [~].
In the MsiServiceConfigFailureActions table entry (MsiServiceConfigFailureActions = ) [1], DelayActions=[4] is not a valid parameter. It should be a list of null-separated non-negative integers.	The DelayActions field of the MsiServiceConfigFailureActions does not contain an array of positive integers. Separate the values in the array by [~].
In the MsiServiceConfigFailureActions table entry (MsiServiceConfigFailureActions = ) [1], number of Actions (= %d) is not	The number of elements in the Actions and the DelayActions fields of the MsiServiceConfigFailureActions table are not equal. The number of

equal to the number of DelayActions (= %d). They should be equal.

elements in these arrays should be the same.

ICE102 posts the following warnings.

<b>ICE104 warning</b>	<b>Description</b>
In the MsiServiceConfig table entry (MsiServiceConfig = )[1], Argument field is left blank. Default preshutdown value of 180000 will be used	The time delay is 180000 milliseconds because the Argument field of the MsiServiceConfig table is blank.
In the MsiServiceConfigFailureActions table entry (MsiServiceConfigFailureActions = )[1], ResetPeriod is left blank. It will be replaced with INFINITE.	The failure count is never be reset because the ResetPeriod field of the MsiServiceConfigFailureActions is blank.

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE 103

ICE 103 verifies the [MsiPrint](#) and [MsiLaunchApp](#) control events.

ICE 103 verifies that the MsiPrint control event is used only on a dialog box with a [ScrollableText](#) control.

**Windows Installer 4.5 or earlier:** Not supported. This ICE is available beginning with Windows Installer 5.0.

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE 104

ICE 104 verifies that only one of the two tables: [MsiLockPermissionsEx](#) and [LockPermissions](#) is present in the package.

ICE 104 verifies the syntax used in the LockObject, Table, and SDDLText fields in the MsiLockPermissionsEx table. ICE 104 does not verify that the value in the SDDLText field in the MsiLockPermissionsEx table is a valid security descriptor.

**Windows Installer 4.5 or earlier:** Not supported. This ICE is available beginning with Windows Installer 5.0.

## Result

ICE 104 posts the following errors.

<b>ICE104 error</b>	<b>Description</b>
Both tables LockPermissions and MsiLockPermissionsEx exist in the database. Only one such table should be present.	A package cannot contain both the MsiLockPermissionsEx and LockPermissions tables.
Could not find item '[2]' in table '[3]' which is referenced in the MsiLockPermissionsEx table entry (MsiLockPermissionsEx = )[1]	Windows Installer is unable to find and secured an item specified in the MsiLockPermissionsEx table.
In the MsiLockPermissionsEx table entry (MsiLockPermissionsEx = )[1], SDDLText = %s appears to be an invalid FormattedSDDLText value	The value in the SDDLText field of the MsiLockPermissionsEx table is required to be a valid <b>FormattedSDDLText</b> data type.

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICE 105

Validates the package against a set of requirements for installation in the per-user context. ICE 105 can also validate dual-purpose packages. A dual-purpose package enables a user to select whether to install the application in the per-user context or per-machine context. For information about developing a dual-purpose package, see [Single Package Authoring](#).

ICE 105 performs the following validation of the package.

- Checks that the [CustomAction](#) table contains no custom actions that have been marked to run with elevated privileges. For more information about elevated custom actions, see [Custom Action Security](#).
- Checks that the [Directory](#) table does not include any of the following system folder properties.
  - [AdminToolsFolder](#)
  - [CommonAppDataFolder](#)
  - [FontsFolder](#)
  - [System16Folder](#)
  - [System64Folder](#)
  - [SystemFolder](#)
  - [TempFolder](#)
  - [WindowsFolder](#)
  - [WindowsVolume](#)
- Verifies that the package does not install a common language runtime assembly to the global assembly cache (GAC.) For more information about installing assemblies to the global assembly cache, see [Adding Assemblies to a Package](#) and [Installation of Common Language Runtime Assemblies](#).
- Checks the [ODBCDataSource](#) table to verify that the package does

not install any data sources.

- Checks the [ServiceInstall](#) table to verify that the package does not install any services.
- Checks that the [Registry](#) table writes no entries under the HKEY\_LOCAL\_MACHINE key.

**Windows Installer 4.5 or earlier:** Not supported. This ICE is available beginning with Windows Installer 5.0.

## See Also

[Single Package Authoring](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Validation Automation

You can use the Evalcom2.dll to implement validation operations for installation packages and merge modules.

The following sections of this documentation contain information about using Windows Installer validation from within authoring tools:

- [Using Evalcom2](#)
- [EvalCom2 Interfaces](#)

It is recommended that Evalcom2.dll be installed using the Windows Installer. The Orca.msi package provided with the Microsoft Windows Software Development Kit (SDK) for Windows XP with Service Pack 2 (SP2) or greater installs Evalcom2.dll. The component ID for the component that contains the COM interface is {53315473-B8D6-470A-9951-D9AE5C11FC91}.

For information about obtaining the Windows SDK, see [Windows SDK Components for Windows Installer Developers](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Evalcom2

Evalcom2.dll can be used to implement validation operations for installation packages and merge modules using [Internal Consistency Evaluators - ICEs](#). The main object implements interfaces for C/C++ programs.

The main object also implements [Evalcom2 interfaces](#) for C/C++ programs. The CLSID required to obtain the interface from **CoCreateInstance** is {6E5E1910-8053-4660-B795-6B612E29BC58}. The REFIID is {E482E5C6-E31E-4143-A2E6-DBC3D8E4B8D3}.

You can use the following procedure to implement validation operations.

## ► To implement validation operations

1. Initialize COM on the calling thread using **CoInitialize**.
2. Obtain the pointer to the **IValidate** interface using **CoCreateInstance**.
3. Open the installation package or merge module using the **OpenDatabase** method.
4. Open the evaluation file using the **OpenCUB** method.
5. Set the display callback function using the **SetDisplay** method.
6. Set the status callback function using the **SetStatus** method.
7. Perform the validation using the **Validate** method.
8. Close the .cub file using the **CloseCUB** method.
9. Close the database using the **CloseDatabase** method.
10. Release the **IValidate** interface.
11. Uninitialize COM using **CoUninitialize**.

## See Also

[Evalcom2 Interfaces](#)  
[Validation Automation](#)  
[Validation Callback Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# EvalCom2 Interfaces

The following are the EvalCom2 interfaces.

## **IValidate**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IValidate Interface

The **IValidate** interface enables authoring tools to validate a Windows Installer package against a set of [Internal Consistency Evaluators](#).

## Methods

The **IValidate** interface inherits the methods of the **IUnknown** interface.

In addition, **IValidate** defines the following methods.

Method	Description
<b>OpenDatabase</b>	Opens an installation package or merge module.
<b>OpenCUB</b>	Opens an internal consistency evaluator database (.cub file).
<b>CloseDatabase</b>	Closes the installation package or merge module.
<b>CloseCUB</b>	Closes the internal consistency evaluator database (.cub file).
<b>SetDisplay</b>	Registers a callback function to receive ICE display messages (info, warning, and error messages).
<b>SetStatus</b>	Enables an authoring tool to receive information about the progress of validation through a registered callback function.
<b>Validate</b>	Validates the current installation package or merge module against the internal consistency evaluator database.

## Requirements

<b>Version</b>	Evalcom2.dll version 3.0.3790.371 or later
<b>DLL</b>	Evalcom2.dll
<b>IID</b>	IID_IValidate is defined as E482E5C6-E31E-4143-A2E6-DBC3D8E4B8D3

## See Also

[Using Evalcom2  
Validation Callback Functions  
IValidate](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IValidate::OpenDatabase Method

The **OpenDatabase** method opens a Windows Installer installation package or merge module for validation.

## Syntax

```
C++HRESULT OpenDatabase(  
    [in] LPCOLESTR szDatabase  
);
```

## Parameters

*szDatabase* [in]

The fully qualified path to the installation package or merge module to be opened. The *szDatabase* parameter cannot be NULL.

## Return Value

Return code	Description
S_OK	The method succeeded.
E_POINTER	The value of <i>szDatabase</i> is invalid.

This method can also return one or more of the errors returned by the [MsiOpenDatabase](#) function. The error is converted to **HRESULTS** using the [HRESULT\\_FROM\\_WIN32](#) function.

## Remarks

The **OpenDatabase** method can also accept a handle to an opened database. The handle to the opened database can be provided in the form "#nnnn" where nnnn is the database handle in string form. For example, for an opened database handle 123, the method can accept #123 for the value of *szDatabase* instead of the path to the package.

## Requirements

<b>Version</b>	Evalcom2.dll version 3.0.3790.371 or later
<b>Header</b>	Evalcom2.h
<b>DLL</b>	Evalcom2.dll
<b>IID</b>	IID_IValidate is defined as E482E5C6-E31E-4143-A2E6-DBC3D8E4B8D3

## See Also

### [IValidate](#)

[Using Evalcom2](#)

[Validation Callback Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IValidate::OpenCUB Method

The **OpenCUB** method opens an [Internal Consistency Evaluator \(ICE\)](#) file that is to be used for validation.

## Syntax

```
C++HRESULT OpenCUB(  
    [in] LPCOLESTR szCUBFile  
);
```

## Parameters

*szCUBFile* [in]

The fully qualified path to the Internal Consistency Evaluator (ICE) file to be used for validation.

## Return Value

The method can return one of the following values.

Return code	Description
S_OK	The method succeeded.
E_POINTER	The value of <i>szDatabase</i> is invalid.
E_OUTOFMEMORY	Failed to allocate memory.
E_FAIL	The method failed.

## Remarks

The Internal Consistency Evaluator (ICE) file typically has a .cub file name extension.

## Requirements

<b>Version</b>	Evalcom2.dll version 3.0.3790.371 or later
<b>Header</b>	Evalcom2.h
<b>DLL</b>	Evalcom2.dll
<b>IID</b>	IID_IValidate is defined as E482E5C6-E31E-4143-A2E6-DBC3D8E4B8D3

## See Also

### [IValidate](#)

[Using Evalcom2](#)

[Validation Callback Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IValidate::CloseDatabase Method

The **CloseDatabase** method closes the currently open Windows Installer package or merge module. Windows Installer packages or merge modules can be opened by using the [OpenDatabase](#) method.

## Syntax

```
C++HRESULT CloseDatabase();
```

## Parameters

This method has no parameters.

## Return Value

Return code	Description
S_OK	The method succeeded.

This method can also return one or more of the errors returned by the [MsiCloseHandle](#) function. The error is converted to **HRESULTS** using the [HRESULT\\_FROM\\_WIN32](#) function.

## Requirements

<b>Version</b>	Evalcom2.dll version 3.0.3790.371 or later
<b>Header</b>	Evalcom2.h
<b>DLL</b>	Evalcom2.dll
<b>IID</b>	IID_IValidate is defined as E482E5C6-E31E-4143-A2E6-DBC3D8E4B8D3

## See Also

## **IValidate**

Using Evalcom2

Validation Callback Functions

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IValidate::CloseCUB Method

The **CloseCUB** method closes an open [Internal Consistency Evaluator \(ICE\)](#) .cub file. Internal Consistency Evaluator (ICE) .cub files can be opened using the [OpenCUB](#) method.

## Syntax

```
C++HRESULT CloseCUB();
```

## Parameters

This method has no parameters.

## Return Value

The method can return one of the following values.

Return code	Description
S_OK	The method succeeded.
S_FALSE	The method failed.

## Remarks

The method returns S\_FALSE if no .cub file has been opened using the **OpenCUB** method.

## Requirements

<b>Version</b>	Evalcom2.dll version 3.0.3790.371 or later
<b>Header</b>	Evalcom2.h
<b>DLL</b>	Evalcom2.dll

## **IID**

IID\_IValidate is defined as E482E5C6-E31E-4143-A2E6-DBC3D8E4B8D3

## **See Also**

### **IValidate**

[Using Evalcom2](#)

[Validation Callback Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IValidate::SetDisplay Method

The **SetDisplay** method enables an authoring tool to obtain ICE status messages through a callback function.

## Syntax

```
C++HRESULT SetDisplay(  
    [in] LPDISPLAYVAL pDisplayFunction,  
    [in] LPVOID pContext  
);
```

## Parameters

*pDisplayFunction* [in]

Specifies a callback function that conforms to the [LPDISPLAYVAL](#) specification.

*pContext* [in]

A pointer to an application context that is passed to the callback function. This parameter can be used for error checking. The *pContext* parameter can be NULL.

## Return Value

The method can return one of the following values.

Return code	Description
S_OK	The method succeeded.
E_POINTER	The <i>pDisplayFunction</i> is invalid.

## Requirements

Version	Evalcom2.dll version 3.0.3790.371 or later
---------	--

<b>Header</b>	Evalcom2.h
<b>DLL</b>	Evalcom2.dll
<b>IID</b>	IID_IValidate is defined as E482E5C6-E31E-4143-A2E6-DBC3D8E4B8D3

## See Also

### [IValidate](#)

[Using Evalcom2](#)

[Validation Callback Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IValidate::SetStatus Method

The **SetStatus** method enables an authoring tool to receive information about the progress of validation through a registered callback function.

## Syntax

```
C++HRESULT SetStatus(  
    [in] LPEVALCOMCALLBACK pStatusFunction,  
        LPVOID pContext  
);
```

## Parameters

*pStatusFunction* [in]

Specifies a callback function that conforms to the **LPEVALCOMCALLBACK** specification. The *pStatusFunction* can be NULL.

*pContext*

A pointer to an application context that is passed to the callback function. This parameter can be used for error checking. The *pContext* can be NULL.

## Return Value

The method can return one of the following values.

Return code	Description
S_OK	The method succeeded.

## Requirements

<b>Version</b>	Evalcom2.dll version 3.0.3790.371 or later
<b>Header</b>	Evalcom2.h

<b>DLL</b>	Evalcom2.dll
<b>IID</b>	IID_IValidate is defined as E482E5C6-E31E-4143-A2E6-DBC3D8E4B8D3

## See Also

### [IValidate](#)

[Using Evalcom2](#)

[Validation Callback Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IValidate::Validate Method

The **Validate** method performs validation of the installation package or merge module using the specified internal consistency evaluator file.

## Syntax

```
C++HRESULT Validate(  
    [in, optional] LPCWSTR szICES  
);
```

## Parameters

*szICES* [in, optional]

Optional parameter that specifies which [Internal Consistency Evaluators \(ICE\)](#) should run. You can specify the ICEs in a delimited list or in a custom table.

When providing a delimited list of ICEs to be run, separate the ICEs in the list by colons (:), for example, "ICE01:ICE03:ICE08".

When providing the name of a custom sequence table, the ICEs to be run can be entered in the custom table.

If the value of *szICES* is NULL, all ICEs in the `_ICESequence` table are run. The `_ICESequence` table is the default table provided with `orca.msi` and `msival2.msi`.

## Return Value

The method can return one of the following values.

Return code	Description
S_OK	The method succeeded.
S_PENDING	The method failed.
E_FAIL	The method failed.

## Requirements

<b>Version</b>	Evalcom2.dll version 3.0.3790.371 or later
<b>Header</b>	Evalcom2.h
<b>DLL</b>	Evalcom2.dll
<b>IID</b>	IID_IValidate is defined as E482E5C6-E31E-4143-A2E6-DBC3D8E4B8D3

## See Also

### [IValidate](#)

[Using Evalcom2](#)

[Validation Callback Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Validation Callback Functions

This section describes the specifications for the callback functions used by Windows Installer package validation.

[LPDISPLAYVAL](#)

[LPEVALCOMCALLBACK](#)

## See Also

[Validation Automation](#)

[IValidate](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# LPDISPLAYVAL Callback Function

The **LPDISPLAYVAL** function specification defines a callback function prototype. The **IValidate::SetDisplay** method enables an authoring tool to receive ICE status messages through the registered callback function.

## Syntax

```
C++BOOL CALLBACK LPDISPLAYVAL(  
    LPVOID pContext,  
    __in RESULTTYPES uiType,  
    __in LPCWSTR szwVal,  
    __in LPCWSTR szwDescription,  
    __in LPCWSTR szwLocation  
);
```

## Parameters

### *pContext*

A pointer to an application context passed to the **SetDisplay** method.

This parameter can be used for error checking.

### *uiType* [in]

Specifies the type of message sent by the ICE.

This parameter can be one of the following values.

Value	Meaning
ieUnknown 0	Unknown ICE message.
ieError 1	ICE error message.
ieWarning 2	ICE warning message.
ieInfo	ICE information message.

3	
---	--

*szwVal* [in]

The name of the ICE reporting the message, or an error reported by evalcom2 during validation.

*szwDescription* [in]

The message text.

*szwLocation* [in]

The location of the error.

This parameter can be NULL if the error does not refer to an actual database table or row. Specify the location of the error using the following format: Table\Column\PrimaryKey1[PrimaryKey2\ . . .].

## Return Value

Return code/value	Description
TRUE 1	Validation procedure should continue.
FALSE 0	Validation was canceled. The callback function return FALSE to stop validation.

## Requirements

<b>Version</b>	Evalcom2.dll version 3.0.3790.371 or later
<b>Header</b>	Evalcom2.h

## See Also

### [IValidate](#)

[Using Evalcom2](#)

[Validation Callback Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# LPEVALCOMCALLBACK Callback Function

The **LPEVALCOMCALLBACK** specification defines a callback function prototype. The **IValidate::SetStatus** method enables an authoring tool to receive information about the progress of validation through the registered callback function.

## Syntax

```
C++BOOL CALLBACK LPEVALCOMCALLBACK(  
    __in STATUSTYPES iStatus,  
    __in LPCWSTR szwData,  
    LPVOID pContext  
);
```

## Parameters

*iStatus* [in]

Specifies the status message sent by evalcom2.

Value	Meaning
NULL	The value of this param
ieStatusICECount 1	Number of ICEs that are being run.
ieStatusMerge 2	Merging the package or merge module with the .cub file.
ieStatusSummaryInfo 3	Merging summary information streams.
ieStatusCreateEngine 4	Preparing to run the ICEs.
ieStatusRunICE	Running an individual ICE.

5		
ieStatusStarting 6		Starting validation.
ieStatusShutdown 7		Finish running the ICEs.
ieStatusSuccess 8		Validation completed successfully.
ieStatusFail 9		Validation failed.
ieStatusCancel 10		Validation was canceled.

*szwData* [in]

A string value containing information appropriate to the status. The value of *szwData* should be the number of ICEs that are being run if *iStatus* is *ieStatusICECount*. The value of *szwData* should be the name of the ICE being run if *iStatus* is *ieStatusRunICE*. Otherwise, the value of *szwData* should be NULL. The callback function should accept NULL as a possible value for this parameter.

*pContext*

Pointer to an application context passed to the **SetStatus** method. This parameter can be used for error checking.

## Return Value

Return code/value	Description
TRUE 1	Validation procedure should continue.
FALSE 0	Validation was canceled. The callback function return FALSE to stop validation.

## Remarks

The **SetStatus** method and **LPEVALCOMCALLBACK** can be used to provide progress information. For example, the `ieStatusICECount` message can provide the overall tick count for a progress bar. For each `ieStatusRunICE` message received, the caller can increment the progress bar one tick.

## Requirements

<b>Version</b>	Evalcom2.dll version 3.0.3790.371 or later
<b>Header</b>	Evalcom2.h

## See Also

[Using Evalcom2](#)

[Validation Callback Functions](#)

[IValidate](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge Modules

Merge modules provide a standard method by which developers deliver shared Windows Installer *components* and setup logic to their applications. Merge modules are used to deliver shared code, files, resources, registry entries, and setup logic to applications as a single compound file. Developers authoring new merge modules or using existing merge modules should follow the standard outlined in this section.

A merge module is similar in structure to a simplified Windows Installer *.msi file*. However, a merge module cannot be installed alone, it must be merged into an installation package using a merge tool. Developers wanting to use merge modules must obtain one of the freely distributed merge tools, such as Mergemod.dll, or purchase a merge tool from an independent software vendor. Developers can create new merge modules by using many of the same software tools used to create a Windows Installer installation package, such as the database table editor Orca provided with the Windows Installer SDK.

When a merge module is merged into the .msi file of an application, all the information and resources required to install the components delivered by the merge module are incorporated into the application's .msi file. The merge module is then no longer required to install these components and the merge module does not need to be accessible to a user. Because all the information needed to install the components is delivered as a single file, the use of merge modules can eliminate many instances of version conflicts, missing registry entries, and improperly installed files.

For more information about merge modules, see:

- [About Merge Modules](#)
- [Using Merge Modules](#)
- [Merge Module Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# About Merge Modules

Merge modules are essentially simplified .msi files, which is the file name extension for a Windows Installer installation package. A standard merge module has an .msm file name extension.

A merge module cannot be installed alone because it lacks some vital database tables that are present in an installation database. Merge modules also contain additional tables that are unique to themselves. To install the information delivered by a merge module with an application, the module must first be merged into the application's .msi file.

A merge module consists of the following parts:

- A [Merge Module Database](#) containing the installation properties and setup logic being delivered by the merge module.
- A [Merge Module Summary Information Stream Reference](#) describing the module.
- A [MergeModule.CABinet](#) cabinet file stored as a stream inside the merge module. This cabinet contains all the files required by the components delivered by the merge module.

[Multiple language merge modules](#) can deliver components to an installation package in multiple languages. In a multiple language merge module the database contains the installation properties and logic for more than one language and the MergeModule.CABinet cabinet includes all the files necessary to install components with all the languages supported by the module.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge Module Database

The database of a merge module contains all the installation properties and setup logic for the module. It is essentially a simplified [installer database](#) or .msi file. Standard merge module database files are indicated by an .msm extension. For a list of all database tables that can exist in merge modules, see [Merge Module Database Tables](#). The following tables are required in the database of every .msm file:

[ComponentDirectory](#)  
[FeatureComponents](#)  
[File](#)  
[ModuleSignature](#)  
[ModuleComponents](#)

Note that the Component, Directory, FeatureComponents, and File tables are also present in all .msi files. A merge module database does not contain a [Feature table](#) and so the .msm file cannot be installed alone. To install a merge module, it must first be merged by using a merge tool into an .msi file.

The [ModuleSignature Table](#) is only present in .msi files that has been merged with at least one .msm file. If this table is present in an .msi file, it contains one record for each merge module that has been previously merged into the installation database.

Merge modules may contain optional MergeModule Sequence tables. These tables occur only in .msm files. When the .msm files are merged into an .msi file, these tables modify the action [sequence tables](#) of the .msi file.

Merge modules may contain custom tables. These tables are used by [custom actions](#) defined in the merge module.

Merge modules rarely require user interface tables. These tables need to be present only in rare cases where the merge module requires input from the user during installation. For more information, see [Authoring User Interfaces in Merge Modules](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MergeModule.CABinet

Every file delivered by the merge module must be stored inside of a cabinet file that is embedded as a stream in the merge module's structured storage. In a standard merge module, the name of this cabinet is always: MergeModule.CABinet. For more information, see [Generating MergeModule.CABinet Cabinet Files](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Multiple Language Merge Modules

Multiple language modules can deliver components with several different languages as a single compound file. The design and functionality of multiple language merge modules is similar to single language modules. A multiple language merge module has more than one language listed in the **Template Summary** Property. The database of a multiple language merge module contains all the setup information for multiple languages. The MergeModule.CABinet cabinet inside a multiple language merge module contains all the files for all the supported languages.

When applying a multiple language .msm file to an .msi file, you must indicate the final language of the installation package after the merge. In the case of a single language merge module, the merge module's **File table** lists every file present in the MergeModule.CABinet cabinet. In the case of a multiple language merge module, MergeModule.CABinet contains all the files for every language supported by the module, but only the subset of files for the final language goes into the module's File table. The merge tool must ensure that the module provides the subset of information and files required for the requested final language.

Every merge module has a default language specified in the Language column of the **ModuleSignature table**. The default language of a merge module is also shown as the first, or only, language in the **Template Summary** Property. Depending on the requested final language and the module's default language, the merge tool may apply language transforms to a multiple language merge module so that it can be opened in the requested language, or an approximation of the requested language. The language transforms are embedded inside the merge module. Merge tools must apply language transforms in adherence to the following general rules:

- If the default and final languages are the same, the module can be merged without using language transforms.
- If the default language is 0 (a language neutral module), the module can be merged without using language transforms.
- If the final language is not the default language, the merge tool must apply one of the language transforms embedded in the module to

change the module to the final language, or to an approximation of the final language.

For example, no language transforms are required if the final language is 1033 (US English) and the default language of the module is 1033 (US English), 0 (language neutral), or 9 (generic English).

Language transforms are required if the final language is 1033 (US English) and the default language is 1031 (German). In this case, the merge tool may first search the multiple language module for an embedded language transform to 1033 (US English). If that fails it may then search for a transform to a language with a matching primary LANGID, even if the secondary LANGID does not match. For example, if the tool cannot find a transform to 1033 (US English), it searches for a transform to 9 (Generic English). If this fails the merge tool searches for a transform to 0 (language neutral). If all these searches for a suitable transform fail, the module fails to open.

For more information, see [Authoring Multiple Language Merge Modules](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Configurable Merge Modules

Merge modules (.msm files) may be authored to contain attributes that are configurable by the consumer of the merge module. This enables the merge module to be configured at the time the installation package and module are merged and installed by the end-user. Configurable merge modules require Mergemod.dll version 2.0 but can run on any version of the Windows Installer.

The implementation of configurable merge modules consists of two parts. First, when creating the merge module (.msm file), the merge module author adds information to the module database that specifies which items can be modified and how these items can be configured by the module user. The author adds entries to the [Merge Module Database Tables](#) that are reserved for configurable information ([ModuleConfiguration table](#) and [ModuleSubstitution table](#)), updates the [\\_Validation table](#), and adds entries for the configurable merge module tables to the [ModuleIgnoreTable table](#). The additions to the ModuleIgnore table are required to make the module compatible with Mergemod.dll versions earlier than 2.0.

Second, when merging the module into an installation package (.msi file), the end-user of the module uses a merge tool. The merge tool calls Mergemod.dll to expose the configuration information in the module to a client configuration tool. The configuration tool may interact with the end-user but is not required to expose all possible configuration options. If the user declines to provide a selection for a configurable item, the module may provide a default value. After the user gives the configuration tool his selections, the merge tool calls Mergemod.dll to perform the merge.

Configurable merge modules are fully compatible with tools earlier than Mergemod.dll version 2.0. In these cases, the tool uses the default values in the module.

For more information, see [Using Configurable Merge Modules](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Merge Modules

For more details about using Windows Installer merge modules, see the following:

- [Authoring Merge Modules](#)
- [Authoring Multiple Language Merge Modules](#)
- [Applying Merge Modules](#)
- [Using Configurable Merge Modules](#)
- [Using 64-bit Merge Modules](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring Merge Modules

The following procedure describes the general steps to authoring merge modules.

## ► To create a new merge module

1. Obtain a software tool you can use to edit the merge module database.
2. Obtain a blank merge module database.
3. Generate a [GUID](#) for the merge module. You need to use this GUID when authoring the primary keys of database tables in the merge module.
4. Add a record to the [Component table](#) for each component delivered by the merge. A Component table is required in every merge module. Note that merge modules operate with components and not with features. In certain cases, however, a database table entry may need to reference a feature. For details, see [Referencing Features in Merge Modules](#).
5. Add a [Directory table](#) to the merge module that specifies the layout of directories the merge module adds to the target database. A Directory table is required in every merge module.
6. Import a blank [FeatureComponents table](#) into the merge module database. This empty table provides a guideline for the merge tool in cases where the .msi file does not contain its own FeatureComponents table.
7. Collect all the files delivered by this merge module and create the [MergeModule.CABinet](#) cabinet file. Add the cabinet to the merge module as a stream inside the .msm file.
8. Add a record to the File table for every file stored in MergeModule.CABinet.
9. Add the information necessary to identify the merge module in the

- [ModuleSignature table](#). Every merge module requires a ModuleSignature table.
10. List the components in the merge module in the [ModuleComponents table](#). Every merge module requires a ModuleComponents table.
  11. Add merge module sequence tables to the .msm file only if the merge module needs to modify the [sequence tables](#) of the target installation database.
  12. Add a \_Validation table to the merge module. A merge module requires a \_Validation table to pass validation.
  13. Merge modules require a user interface in only rare cases. Including a UI with a merge module is not recommended. In cases where a user interface is required, the UI tables can be merged into the .msi file the same as other tables.
  14. Add registry information to the appropriate registry tables in the merge module database. Add registry information for type libraries, classes, extensions, and verbs into the [TypeLib](#), [Class](#), [AppId](#), [ProgId](#), [Extension](#), [Verb](#), or [MIME](#) tables. All other registry information can go into the [Registry table](#). The use of the SelfReg table is not recommended.
  15. Add the summary information to the [Merge Module Summary Information Stream](#).
  16. Run validation on all merge modules before attempting to install.

## See Also

[Obtaining Blank Merge Module Databases](#)  
[Obtaining Merge Module Authoring Tools](#)  
[Naming Primary Keys in Merge Module Databases](#)  
[Authoring Merge Module Component Tables](#)  
[Authoring Merge Module Directory Tables](#)  
[Authoring Merge Module FeatureComponents Tables](#)  
[Generating MergeModule.CABinet Cabinet Files](#)

[Authoring Merge Module File Tables](#)  
[Authoring ModuleSignature Tables](#)  
[Authoring ModuleComponents Tables](#)  
[Authoring Merge Module Sequence Tables](#)  
[Validating Merge Modules](#)  
[Authoring User Interfaces in Merge Modules](#)  
[Authoring Merge Module Registry Tables](#)  
[Authoring Merge Module Summary Information Streams](#)  
[Merge Module Summary Information Stream Reference](#)  
[Validating Merge Modules](#)  
[Using 64-bit Merge Modules](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Obtaining Merge Module Authoring Tools

To generate a merge module, you need to obtain a software tool with which to edit an .msm file. Because an .msm file is basically a simplified .msi file, you may be able to use the same tools used to create an installation database. For example, the application Orca.exe provided with the Windows Installer SDK.

An alternative is to purchase one of the installer packaging tools to be available from independent software vendors. There are several third-party tools under development that may be used for producing merge modules. If you choose to use a third-party tool you should verify that it generates merge modules that are consistent with the standard described in this document. In particular, you should determine that the editing tools have not done any of the following to the merge module.

- Added extraneous tables to the merge module that are not referenced in the [ModuleIgnoreTable table](#).  
Delete these tables or add them to the ModuleIgnoreTable table.
- Added an unnecessary [TextStyle table](#) to the merge module.  
If your merge module has no UI (and it generally should not) you can safely delete this table.
- Added unnecessary entries to the [Directory table](#).  
Remove unnecessary entries from the Directory table.
- Left information out of the merge module's `_Validation` table.  
This prevents merge module validation. Add a complete `_Validation` table.

## See Also

[Authoring User Interfaces in Merge Modules](#)

[Authoring Merge Module Directory Tables](#)

[Validating Merge Modules](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Obtaining Blank Merge Module Databases

Obtain a blank merge module database. You can use the file Schema.msm provided with the Windows Installer SDK as a starting database for your merge module. For more information, see [Windows SDK Components for Windows Installer Developers](#).

Developers should author merge modules using the simplest database schema that installs their components. The use of a simple schema ensures the greatest compatibility for the merge module. Merging a merge module into an installation package with a different database schema usually results in merge conflicts.

For a complete list of all the required and optional tables in merge modules, see [Merge Module Database Tables](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Naming Primary Keys in Merge Module Databases

The names of primary keys a merge module database must adhere to a standard naming convention. The purpose of this naming convention is to reduce the possibility of creating a name conflict between the table columns in the merge module and the target installation package. The naming convention cannot be applied to tables in which the primary key is installable data. Do not apply the naming convention to the following tables:

- [MIME Table](#)
- [Extension Table](#)
- [Icon Table](#)
- [Verb Table](#)
- [ProgId Table](#)

For example, do not use for the primary key of the MIME table because this is the MIME type and applying the naming procedure would change its meaning. In these cases, name conflicts depend on the meaning of the data being unique across modules.

The name of a primary key in a merge module must consist of a readable name appended with a string made from the merge module's GUID. Every merge module must have its own [GUID](#). The merge module's GUID should also be authored into the [Revision Number Summary](#) property of the merge module. Developers can create GUIDs using a utility such as GUIDGEN.

The following procedure describes how to generate a primary database key that adheres to the standard naming convention. Apply the following procedure only to tables where the primary key is not data being installed.

## ▶ To name a primary key of a table record in a merge module

1. Author the readable part of the name for the primary key. Pick a

readable name that identifies this record, for example, MyRowEntry.

2. Generate or obtain the GUID of the merge module. Note that all GUIDs must be authored in uppercase. For more information about GUIDs, see [GUID](#). The following is an example of a GUID: {880DE2F0-CDD8-11D1-A849-006097ABDE17}. In the following steps you modify this into a character string that must be appended to every primary key name in the merge module.
3. Remove the curly braces from the beginning and end of the GUID.
4. Change all the dashes to underscores.
5. Append the result to the end of the readable part of the primary key name. Separate the readable name from the modified GUID by a period. The primary key name for the example GUID given above becomes  
MyRowEntry.880DE2F0\_CDD8\_11D1\_A849\_006097ABDE17.
6. Repeat to name all the primary keys of all tables in the merge module.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring Merge Module Component Tables

A [Component table](#) is required in every merge module. This table contains a record for each component delivered by the merge module to the target .msi file. Note that each of these components must also be specified in the [ModuleComponents table](#) described in [Authoring ModuleComponents Tables](#).

Use the standard naming convention when entering names into the Component column to ensure that the identifier for each component is unique for all merge modules and installation databases. For more information, see [Naming Primary Keys in Merge Module Databases](#).

Complete the remaining fields in each record as described for an installation database in [Component table](#). The components added to a package by a merge module must adhere to the guidelines for valid Windows Installer Components described in the following sections:

- [Windows Installer Components](#)
- [Organizing Applications into Components](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Referencing Features in Merge Modules

Merge modules only operate with components and not with features. However, some tables in merge modules, such as the [PublishComponent table](#), contain fields that refer to features.

A null GUID: {00000000-0000-0000-0000-000000000000} must be authored into any field of a merge module database that references a feature. When the merge module is merged into an installation package, the merge tool replaces the null GUID with the feature specified in the installation package, except for tables that require special handling, such as the [ModuleSignature table](#) and the ModuleSequence tables.

Note that if a null GUID is used as a primary key, it is not guaranteed that the value substituted by the merge tool is unique. Authors of merge modules are responsible for ensuring that no existing primary key in the user's interface is duplicated when the merge tool replaces null GUIDs with features.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring Merge Module Directory Tables

A merge module can be applied to an .msi file to add directories to the installation but it cannot replace or remove any existing directories. The [Directory table](#) specifies the layout of the directories the merge module provides to the target installation. A Directory table is required in every merge module.

Use the following guidelines when authoring the Directory Table in a merge module. For more information, see [Directory Table](#) and [Using The Directory Table](#).

- The directory structure added by the merge module must have a single root directory. The root must be named TARGETDIR. The user may change the value of TARGETDIR during the merge to specify where to attach the module's directory structure onto the target's directory tree.
- Merge module tables other than the Directory table must not directly reference directory locations to TARGETDIR. The location of such a reference changes if the value of TARGETDIR is changed by the user.
- The tables in the merge module must reference the location of a child directory of TARGETDIR, or another directory in the merge module's tree. Do the following to specify TARGETDIR as the parent of a directory in the merge module. Enter the directory into the Directory column and enter TARGETDIR into the Directory\_Parent column. Use the "." notation in the DefaultDir column to indicate that this directory is located in TARGETDIR without a subdirectory. For more information, see [Using the Directory Table](#).
- The names of directories added by the merge module must use the naming conventions described in [Naming Primary Keys in Merge Module Databases](#). This includes directories predefined by

properties such as the **SystemFolder** property and **ProgramFilesFolder** property.

- Append a **GUID** to every entry in the Directory table (except TARGETDIR.) This includes Directory table entries that specify Windows Installer **SystemFolder** properties, for example, SystemFolder.00000000\_0000\_0000\_0000\_000000000000. The library Mergemod.dll adds custom actions to set the **SystemFolder** property.
- When a predefined directory is included in a merge module, the merge tool automatically adds a **Custom Action Type 51** to the target database. The merge module author must ensure that a **CustomAction table** is also included. The CustomAction table may be empty, but this table is required to exist in the target database and ensures that the modified predefined directories are written to the correct locations. For example, when a system directory is included in a merge module, the merge module author must ensure that a Custom Action table exists.

Note that the matching algorithm for the generation of these type 51 custom actions only checks that the directory name begins with one of the predefined **SystemFolder** properties. It does not verify that the directory name exactly equals the directory property. Any directory beginning with one of these standard folder names gets a type 51 custom action, even if the rest of the name is not a GUID. Authors need to take care that this does not generate false positive matches, and unintended custom action generation, on derivative primary keys that begin with one of the **SystemFolder** properties.

The following is an example of a Directory table in a merge module and the expected resolved directories.

Directory	Directory_Parent	DefaultDir
TARGETDIR		SourceDir

Dir00.BC82E350_ C7FC_11d1_ A848-006097ABDE17	TARGETDIR	.:MMM_Prog
SystemFolder.BC82E350_ C7FC_11d1_ A848-006097ABDE17	TARGETDIR	MMM_Sys
Dir02.BC82E350_ C7FC_11d1_ A848-006097ABDE17	Dir00.BC82E350_ C7FC_11d1_ A848_006097ABDE17	MFC_OCX

A merge module with the above Directory table is expected to result in the following directory structure.

<b>Directory</b>	<b>Target</b>	<b>Source</b>
Dir00.BC82E350_ C7FC_11d1_ A848-006097ABDE17	[Merge Module's Install Point]\	[Merge Module's Source Point]\MMM_Prog
SystemFolder.BC82E350_ C7FC_11d1_ A848-006097ABDE17	[SystemFolder]\	[Merge Module's Source Point]\MMM_Sys
Dir02.BC82E350_ C7FC_11d1_ A848-006097ABDE17	[Merge Module's Install Point]\MFC_OCX	[Merge Module's Source Point]\MMM_Prog\MFC_OCX

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring Merge Module FeatureComponents Tables

A blank [FeatureComponents table](#) is required in every merge module. This empty table provides a schema for the merge tool if the target .msi file does not have its own FeatureComponents table.

If, and only if, there is no FeatureComponents table in the target .msi file does the merge tool use the blank table provided in the module. In this case, the merge tool creates a duplicate table in the target .msi file and adds rows that associate the new components in the merge module to the features in the application's installation package.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Generating MergeModule.CABinet Cabinet Files

Every file that is delivered to the target installation package by the merge module must be stored inside of a cabinet file embedded as a stream inside the .msm file. The name of this cabinet is always MergeModule.CABinet.

The names of files in MergeModule.CABinet must match the primary keys used in the merge module's [File table](#) and must adhere to the convention described in [Naming Primary Keys in Merge Module Databases](#).

The installer skips extra files included in MergeModule.CABinet that are not listed in the merge module's [File table](#). The sequence numbers of files specified in the File table do not need to be consecutive, but they must follow the same sequence as the files stored inside MergeModule.CABinet. For more information, see [Authoring Merge Module File Tables](#).

This means that a single cabinet file can contain all the files needed for a merge module to support multiple languages. All the language files can be given unique sequence numbers in the cabinet and then a language transform can be used to add or remove files from the File table to obtain a merge module for a particular language. For details, see [Authoring Multiple Language Merge Modules](#).

MergeModule.CABinet can be added to the merge module by opening a temporary [\\_Streams Table](#). For example, the tool Msidb.exe provided with the Windows Installer SDK can be used to add the MergeModule.CABinet to the merge module. For more information, see [Including a Cabinet File in an Installation](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring Merge Module File Tables

A [File Table](#) is required in every merge module, and should have a record for each file that is being delivered to the target installation package by the merge module. When the merge module is merged into a .msi file, every file in the merge module File Table is stored inside a [cabinet file](#) in the .msm file. The name of the cabinet in a merge module is always the following: MergeModule.CABinet.

For more information, see [Generating MergeModule.CABinet Cabinet Files](#).

- Because the files of a merge module are always stored inside a cabinet file, it is not necessary to set the `msidbFileAttributesNoncompressed` or `msidbFileAttributesCompressed` bit flags in the Attributes column of the [File Table](#).
- The names of files in MergeModule.CABinet must match the primary key in the merge module's [File Table](#).  
The File column is the primary key of the [File Table](#) and the entries in this field must follow the convention that is described in [Naming Primary Keys in Merge Module Databases](#).
- File sequence numbers are specified in the Sequence column of the [File Table](#).  
Files must be listed in the merge module's [File Table](#) in the same sequence that they are stored in MergeModule.CABinet. The sequence numbers of files do not need to be consecutive, but they must follow the same sequence as the files that are stored inside the cabinet. For example, the first, second, and third files stored in the cabinet can have the sequence numbers 100, 200, and 300.
- The Installer skips extra files included in MergeModule.CABinet that are not listed in the [File Table](#).  
One cabinet file can contain all the files necessary for a merge

module that supports multiple languages using transforms. All the language files can be given a unique sequence number in the cabinet, and then a transform can add or remove files from the [File Table](#) when needed for a specific language. For more information, see [Authoring Multiple Language Merge Modules](#).

For more information, see [File Table](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring ModuleSignature Tables

The [ModuleSignature table](#) contains all the information needed to identify the merge module.

The ModuleID field is the primary key for this table and must follow the convention described in [Naming Primary Keys in Merge Module Databases](#). For example, if the readable name of the merge module is MyLibrary and the GUID for the merge module is {880DE2F0-CDD8-11D1-A849-006097ABDE17}, the entry in the ModuleID column of the ModuleSignature table becomes  
MyLibrary.880DE2F0\_CDD8\_11D1\_A849\_006097ABDE17.

Enter the decimal identifier for the default language of the merge module into the Language field of the ModuleSignature table.

Enter the version of the merge module into the Version field.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring ModuleComponents Tables

List the all the components delivered by the merge module in the [ModuleComponents table](#). Note that the merge module's components are also specified in the Component table. For more information, see [Authoring Merge Module Component Tables](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring Merge Module Sequence Tables

Include the MergeModuleSequence tables in the .msm file if the merge module must modify the action [sequence tables](#) of the target .msi file. Merging does not add these tables to the .msi file. These tables only occur in merge modules.

If any of the ModuleSequence tables are present in an .msm file, an empty copy of the corresponding installer sequence table must also be authored into the merge module. For example, if a merge module contains a ModuleAdminExecuteSequence table, the merge module must also include an empty AdminExecuteSequence table. During a merge, these empty tables provide the merge tool with necessary schema guidelines.

When using [standard actions](#) in merge module sequence tables, the value in the Sequence column should be the recommended action sequence number for the standard action. See the suggested action sequences given below for the recommended sequence numbers in each sequence table. If the sequence number in the merge module sequence table differs from the sequence number for the same action in the .msi file, the merge tool uses the sequence number in the .msi file during the merge.

<b>MergeModuleSequence table</b>	<b>Recommended action sequences</b>
<a href="#">ModuleAdminUISequence</a>	<a href="#">Suggested AdminUISequence</a>
<a href="#">ModuleAdminExecuteSequence</a>	<a href="#">Suggested AdminExecuteSequence</a>
<a href="#">ModuleAdvtUISequence</a>	<a href="#">Suggested AdvtUISequence</a>
<a href="#">ModuleAdvtExecuteSequence</a>	<a href="#">Suggested AdvtExecuteSequence</a>
<a href="#">ModuleInstallUISequence</a>	<a href="#">Suggested InstallUISequence</a>
<a href="#">ModuleInstallExecuteSequence table</a>	<a href="#">Suggested InstallExecuteSequence</a>

If a [standard action](#) is used in the Action column of a merge module sequence table, the BaseAction and After columns of that record must be

Null.

If a custom action or dialog is entered into the Action column, the Sequence column must be Null.

If an action returning a termination flag is entered into the Action column, the Sequence column should contain the negative value for that flag and the BaseAction and After columns of that record must be Null. The following negative values indicate that the action is called if the installer returns the termination flag.

<b>Termination flag</b>	<b>Value</b>	<b>Description</b>
msiDoActionStatusSuccess	-1	Successful completion.
msiDoActionStatusUserExit	-2	User terminates install.
msiDoActionStatusFailure	-3	Fatal exit terminates.
msiDoActionStatusSuspend	-4	Install is suspended.

The BaseAction column can contain a standard action, a custom action specified in the merge module's custom action table, or a dialog specified in the module's dialog table. The BaseAction column is a key into the Action column of this table. It cannot be a foreign key into another merge table or table in the .msi file. This means that every standard action, custom action, or dialog listed in the BaseAction column must also be listed in the Action column of another record in this table.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring User Interfaces in Merge Modules

Merge modules rarely require a user interface. Releasing a merge module that contains both installable components and a user interface ultimately restricts the flexibility of the module user. Combining both components and UI into one module can prevent developers from using their own UI or from providing silent installations. A better alternative is to release two merge modules, one that silently installs the components and a second optional module that contains the UI. The module with the UI should list the component module in its [ModuleDependency table](#). This method enables module authors to provide a UI without forcing it on developers.

When UI tables are used in merge modules they can be merged in the same way as any other tables.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring Merge Module Registry Tables

Use Merge Module Registry tables according to the type of registry information.

## TypeLib, Class, Appld, ProgId, Extension, Verb, or MIME Tables

For type libraries, classes, extensions, and verbs, author registry information into the merge module's [TypeLib](#), [Class](#), [Appld](#), [ProgId](#), [Extension](#), [Verb](#), or [MIME](#) tables. If you use the [Registry table](#) to add this information, Windows 2000 cannot provide system-wide advertisement for these components.

Merge module authors may decide not to register using the Class table for the following reasons:

- To be registered by the Class table, the file has to be the KeyPath of its component. This may require an unacceptable change in the organization of components.
- A COM call may trigger an installer attempt to reinstall an advertised class. Authors may decide not to register a class using the Class table in order to avoid triggering a reinstallation when the client computer does not support a user interface.

## Registry Table

Use the Registry table to add registry information that cannot be authored into the TypeLib, Class, Appld, ProgId, Extension, Verb, or MIME tables. Windows 2000 cannot provide system-wide advertisement for components that use the Registry table.

When authoring the Registry table, refer to file paths using the [#File] or [!File] format rather than as [Directory]Filename. The latter format does not support run-from-source installation. The former format also makes

missing files and faulty components easier to detect.

Care is needed when using formatted text in the Key column of the Registry table. Because Windows Installer does not reinstall components that are already installed, using formatted text in this field may cause registry keys to be left behind on application removal.

## **SelfReg Table**

Using the SelfReg table is not recommended. For a list of reasons for not using self-registration, see [SelfReg table](#). You should use the TypeLib, Class, AppId, ProgId, Extension, Verb, and MIME tables where possible instead and the Registry table in all other cases.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring Merge Module Summary Information Streams

When authoring a merge module, the summary information stream is required to include these required summary information properties.

Every merge module is required to have a Merge Module Summary Information Stream with the following summary information properties:

- [Template Summary](#)
- [Revision Number Summary](#)
- [Page Count Summary](#)
- [Word Count Summary](#)

For a complete list of summary information properties in merge modules, see [Merge Module Summary Information Stream Reference](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Validating Merge Modules

Authors should run validation on every new, or newly modified, merge module before attempting to merge the module into an installation database for the first time. Merge module validation works in the same way as [package validation](#) but uses a different set of [Internal Consistency Evaluators](#) (ICEs) specific to merge modules. For more information about these merge module ICEs, see [Merge Module ICE Reference](#).

Merge module ICEs are stored in the merge module .cub file, Mergemod.cub. The installation of the Orca tool or msival2 also provides the Logo.cub, Darice.cub, and Mergemod.cub files.

For more information, see [Merge Module ICE Reference](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Adding Readme Information Streams to Merge Modules

A merge module must not include a readme information stream.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using 64-bit Merge Modules

A 64-bit merge module has any of the characteristics identified in this this topic.

- The merge module contains at least one component that has been compiled for 64-bit operating systems.
- The merge module contains no 64-bit components, but is intended for use only with [64-bit Windows Installer](#) packages.

Merge modules can be used as follows:

- A 64-bit merge module can be merged into a 64-bit Windows Installer package. The **Template Summary** properties in the merge module and in the Windows Installer package must be set to the same type of 64-bit processor. A x64 merge module can be merged only into x64 packages and an Intel64 merge module can be merged only into Intel64 packages.
- A 32-bit merge module can be merged into 32-bit or 64-bit Windows Installer packages.
- A 64-bit merge module can be merged into a 64-bit package on a 32-bit operating system.

Adhere to the following when authoring a 64-bit merge module:

- Use the same general procedure as when authoring 32-bit merge modules. For information, see [About Merge Modules](#) and [Authoring Merge Modules](#).
- You must set the **Template Summary** property with the Intel64 value if running an Intel64 system. You must set the **Template Summary** property with the x64 value if running an x64 system. For information see [Merge Module Summary Information Stream Reference](#).
- When both 32-bit and 64-bit merge modules exist for the same

component, it is recommended that the modules have different signatures. This will enable a package to contain both versions of the component.

For more information, see [Windows Installer on 64-bit Operating Systems](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring Multiple Language Merge Modules

The following sections discuss authoring multiple language merge modules in detail.

[Choosing the Default Language of a Multiple Language Merge Module](#)

[Authoring a Language Transform for a Multiple Language Merge Module](#)

[Ordering the File Sequence in the CAB of a Multiple Language Merge Module](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Choosing the Default Language of a Multiple Language Merge Module

The default language for a merge module is the language that is listed in the [ModuleSignature Table](#) of the module before language transforms are applied. This is also the first language that is listed in the [Template Summary](#) Property.

The default language for the module should be one of the most specific languages that you support, because the default language is always checked first, and if it satisfies the requested language, it is used even if another language is a better match. For example, if a module supports 1033 and 0, 1033 should be the default language. If 0 were the default language, it would always satisfy any request, and the 1033 transform would never be used, even if 1033 is the exact language requested.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Authoring a Language Transform for a Multiple Language Merge Module

When a module is merged into an database that has a different default language, the merge tool may need to apply a language transform to the module to provide the final language. For more information, see [Multiple Language Merge Modules](#).

The language transforms are stored in the module's .msm file and must have the name and format: MergeModule.Lang####. The #### represents the up-to four digit [LANGID](#) of the final language. For example, MergeModule.Lang1033, MergeModule.Lang9, and MergeModule.Lang0 for transforms to US English, world English, and language neutral. These are the same as [Embedded Transforms](#) and you can add them to substorages in the .msm file.

The language transform should do the following:

- Change the default language in the Language column of the [ModuleSignature table](#) to the new language of the module.
- Change the default language in the Language column of the [ModuleComponents table](#) to the new language of the module. The transform may add or remove rows from this table.
- If necessary, change the language in the RequiredLanguage column, or add or delete rows, to the [ModuleDependency table](#).
- If necessary, change the language in the ExcludedLanguage column, or add or delete rows, to the [ModuleExclusion table](#).
- The transform may perform any valid transform operations on the module, including adding or removing components, files, registry entries, or actions.

Note that applying a language transform when opening the module does not change the default language or the languages supported by the module, it just changes the language that is being requested. Therefore the [Template Summary](#) property does not change, it should already list

all of the languages supported by the module with the default language listed first.

All files needed by all possible language transforms are commonly stored in a single cabinet file included with the module. Because it is not practical to have the language transform modify this cabinet file, it is best to use a global file sequence in the cabinet file, [File table](#), and language transform. For details, see [Ordering the File Sequence in the CAB of a Multiple Language Merge Module](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Ordering the File Sequence in the CAB of a Multiple Language Merge Module

Multiple-language merge modules, language transforms, and cabinet files must be authored such that the order of the files in the .cab matches the installation order of files specified in the [File Table](#), even after the application of the language transform. If the order in the module and in the .cab do not match, the merge module cannot be used.

Assign to each file in the module, a unique sequence number that is independent of its language, and then always use that sequence number for the file. Use the same sequence when building the cabinet file and authoring a language transform.

Because the Installer only installs the files listed in the [File Table](#), the use of a global file sequence in the cabinet, [File Table](#), and language transform enables the merge tool to skip any extra files stored in the cabinet that are not listed in the [File Table](#). Other files may exist in the cabinet, but they must not be listed in the [File Table](#). For example, a module installing Code.dll, English.dat, German.dat, and French.dat can use the following global file sequence order.

File	Sequence
Code.Dll	1
English.Dat	2
German.Dat	3
French.Dat	4

Language transforms can then be authored to modify the [File Table](#) of the module for English, German, or French.

[File Table](#) (partial for English)

File	Sequence

Code.Dll	1
English.Dat	2

#### File Table (partial for German)

File	Sequence
Code.Dll	1
German.Dat	3

#### File Table (partial for French)

File	Sequence
Code.Dll	1
French.Dat	4

For more information, see [Authoring a Language Transform for a Multiple Language Merge Module](#) and [Authoring Merge Module File Tables](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Applying Merge Modules

Merge modules must be merged into an installation package using a merge tool. The best alternative is to obtain a freely distributed merge tool or purchase one of the merging tools to be available from independent software vendors. For example you can use the functionality provided by Mergemod.dll.

Multiple-language merge modules can be opened in different languages and then merged into a target to produce a different language version of the package. For more information, see:

- [Using Automation to Merge a Merge Module into a Database](#)
- [Using the API to Merge a Merge Module into a Database](#)
- [Opening a Multiple-Language Merge Module in a Specific Language](#)
- [Merging a Multiple Language Module into the Same Package Multiple Times](#)
- [Connecting a Merge Module to Multiple Features](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Automation to Merge a Merge Module into a Database

[Merge Modules](#) provide a standard method for you to deliver shared Windows Installer *components*, and setup logic to applications.

Merge modules must be merged into an installation package by using a merge tool. The best practice is to obtain a freely distributed merge tool, or purchase one of the merge tools that are available from independent software vendors, for example, you can use [Mergemod.dll](#).

The following procedure shows you how to merge a merge module into a Windows Installer database by using Merge Module Automation.

## ► To merge a module into a database

1. Open a log file by using the [OpenLog](#) method.  
This step is required only if you need to create a log file, or append an existing log file for the merge process.
2. Open the [.msi](#) installation database by using the [OpenDatabase](#) method of the [Merge Object](#).  
This step is required.  
  
The database that you open is the one that you want to receive the merge module.
3. Open the [.msm](#) merge module by using the [OpenModule](#) method.  
This step is required.  
  
This is the merge module that is being merged into the database. A module must be opened before it can be merged with an installation database.
4. Merge the module into the installation database by calling the [Merge](#) method or [MergeEx](#) method.  
This step is required.

The **Merge** method or **MergeEx** method can only be called one time to merge a specific combination of .msi and .msm files.

**Note** The **MergeEx** method is only available in Mergemod.dll version 2.0 or later, and only when using the [IMsmMerge2](#) interface.

5. Retrieve the [Errors](#) property and examine the collection of [Error](#) objects it returns for merge conflicts or other errors.

You must resolve any errors.

The retrieval is nondestructive, and multiple instances of the error collection can be retrieved by repeatedly reading the **Errors** property.

6. Associate the components of the merge module with the features by using the [Connect](#) method.

This step is only required if you have existing features and you want to add features to merge into the installation database.

A feature must exist before you call this method. For more information, see [Connecting a Merge Module to Multiple Features](#).

7. If necessary, extract source files from the module by doing one or more of the following:

- Use [ExtractFiles](#) or [ExtractFilesEx](#) to extract files from an embedded .cab file and then copy into a specified directory.  
**Note** [ExtractFilesEx](#) requires Mergemod.dll version 2.0 or later.
- Use [ExtractCAB](#) to extract files from an embedded .cab file, and then save in a specified file.
- Use [CreateSourceImage](#) to extract files from a module, and then after the merge, copy to a source image on disk.  
**Note** [CreateSourceImage](#) is only available in Mergemod.dll version 2.0 or later.

8. Close the current open merge module by using the [CloseModule](#) method.

This step is required.

9. Close the open installation database by using the [CloseDatabase](#) method.

This step is required.

Closing a database clears all dependency information, but does not affect any errors that are not retrieved.

10. Close the current log file by using the [CloseLog](#) method.

This step is required if you have an open log file.

After the module has been merged into the database using Mergemod.dll, the [Media Table](#) must be updated to describe the desired source image layout. The merge process provided by Mergemod.dll does not update the Media Table because the consumer of the merge module can select various ways to layout the source image.

## See Also

[Released Versions, Tools, and Redistributables](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using the API to Merge a Merge Module into a Database

[Merge modules](#) provide a standard method for developers to deliver shared Windows Installer *components* and setup logic to their applications. Merge modules must be merged into an installation package using a merge tool. The best alternative is to obtain a freely distributed merge tool or purchase one of the merging tools available from independent software vendors. For example you can use the functionality provided by [Mergemod.dll](#).

Use the following steps in sequence to merge a merge module into a Windows Installer installation database by the API of Mergemod.dll.

## ► To merge a merge module into a Windows Installer installation database

1. Open a log file using [OpenLog](#). This step is required only if you need to create a log file, or append an existing log file, for the merge process.
2. Open the installation database, an [.msi file](#), that will receive the merge module using [OpenDatabase](#). This step is required.
3. Open the merge module, an [.msm file](#), that is being merged into the database using [OpenModule](#). A module must be opened before it can be merged with an installation database. This step is required.
4. Merge the module into the installation database by using [Merge](#) or [MergeEx](#). Note that **Merge** or **MergeEx** can only be called once to merge a particular combination of [.msi](#) and [.msm](#) files. **MergeEx** is only available when using Mergemod.dll version 2.0 or later and only when using the [IMsmMerge2](#) interface. This step is required.
5. Call [get\\_Errors](#) and examine the retrieved collection of errors for merge conflicts or other errors. The retrieval is non-destructive.

Multiple instances of the error collection may be retrieved by repeatedly reading calling **get\_Errors**. You will need to resolve any errors as appropriate for your case.

6. Associate the components of the merge module with any additional features that have been, or will be, merged into the installation database using **Connect**. The feature must already exist before calling this method. This step is only required if you have additional features, for more information see [Connecting a Merge Module to Multiple Features](#)
7. If necessary, extract source files from the module by doing one or more of the following.
  - To extract files from an embedded .cab file and then copy into a specified directory, use **ExtractFiles** or **ExtractFilesEx**. Note that **ExtractFilesEx** requires Mergemod.dll version 2.0 or later.
  - To extract files from an embedded .cab file and then save in a specified file, use **ExtractCAB**.
  - To extract files from a module and then copy to a source image on disk after the merge, use **CreateSourceImage**. Note that **CreateSourceImage** is only available with Mergemod.dll version 2.0 or later.
8. Close the current open merge module using **CloseModule**. This step is required.
9. Close the current open installation database using **CloseDatabase**. This step is required. Closing a database clears all dependency information but does not affect any errors that have not been retrieved.
10. Close the current log file using **CloseLog**. This step is required if you have opened a log file.

After the module has been merged into the database using Mergemod.dll, the [Media Table](#) must be updated to describe the desired

source image layout. The merge process provided by Mergemod.dll does not update the Media Table because the consumer of the merge module can select various ways to layout the source image.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Opening a Multiple-Language Merge Module in a Specific Language

When merging a module into an installation database, there are two important languages. The first is the language of the target installation package specified by **ProductLanguage** in the [Property Table](#). The second is the language of the merge module that appears in the Language column of the [ModuleSignature Table](#).

The language of the installation package can be passed to the module by the merge tool when the package is opened for a merge. However, sometimes it may be necessary to disregard the language of the target, and request that the module be opened in another language, for example, an English package installing both the English and German resources from the module.

When opening a module with a language request, the merge tool checks the requested language against the languages that are specified in the Language column of the [ModuleSignature Table](#).

The following process is used to determine which language to use.

## ► To determine which language to use

1. If the language in the [ModuleSignature Table](#) is equal or more general than the requested language, the module opens.
2. If the module supports the exact language requested, that language is used.
3. If the module supports the language group of the language requested that language group is used, for example, check 9 if 1033 was requested but not found in step 2.
4. Check if there is a language transform that changes the module to neutral.
5. If none of the previous steps succeed, the module does not support the requested language and the merge fails.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merging a Multiple Language Module into the Same Package Multiple Times

When a module supports multiple languages, you can merge it into the same Windows Installer database multiple times, but make sure that each merge uses a different language. Before each merge, request a different language from the module. The resulting .msi database then has a record in the [ModuleSignature Table](#) for each merge of the module. Components that are shared between languages exist only one time in the [Component Table](#), but are associated with each language in the [ModuleComponents Table](#).

When merging multiple languages of a module into the same package, each merge must satisfy the same restrictions on code pages as single-language modules. The modules cannot contain strings in different code pages.

When merging a module multiple times into a single .msi file, you may need to modify the order of the files in the [File Table](#) to use the existing .cab from the module directly in your installation. The order of files in the File Table must match the order of files in the .cab. When merging a module multiple times into an installation database the sequence may be modified, because files shared between the languages may already exist in the module from a prior merge, and they have a different relative sequence number.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Connecting a Merge Module to Multiple Features

The **Connect** method of the **Merge Object** can be used to connect a module to an additional feature that has been merged into the database or that will be merged into the database. The feature must exist before calling this method.

A merge module should never deliver a component containing system files to the main feature of an application, because this may cause the Installer to validate and repair the application each time the system file is used. A .msi file that is intended to accept components from a merge module should be authored so that any components that contain system files only belong to features that are separate from the main feature of the application.

If your package uses a merge module containing system files that link all the components from the merge module to the main feature of the application, an attempt to use the system file may trigger the Installer to try and repair the main feature of the application. If the main feature cannot be repaired, the installation may fail.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Configurable Merge Modules

The following sections describe how merge module authors, module consumers, and merge tool developers may use configurable merge modules:

[Creating a Merge Module that Can Be Configured by the End-User](#)

[Applying a Configurable Merge Module with Customizations](#)

[Adding Module Configuration Capability to a Merge Tool](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Creating a Merge Module that Can Be Configured by the End-User

To create merge modules, use the general guidelines that are described in the [Authoring Merge Modules](#) topic. In addition, you must do the following to create a merge module that may be configured by the end-user of the module:

- End-users need to have Mergemod.dll version 2.0 to configure your module. Users who have earlier versions of Mergemod.dll can apply the module, but they always get the default settings.
- Add a [ModuleConfiguration Table](#) to the merge module to identify the items that may be configured by an end-user. Add a record in this table for each configurable item. These items are substituted into the templates that are specified in the [ModuleSubstitution Table](#). Enter a name for each configurable item into the Name field. Enter the format, type, and semantic context for each item in the Format, Type, and ContextData columns. For more information, see [Semantic Types](#). Enter a default value for the item in the DefaultValue field using the [CMSM Special Format](#).
- Add a [ModuleSubstitution Table](#) to the merge module. Each record in this table corresponds to a substitution of one or more configurable items into one field of the merge module database. Enter the table, row, and column of the field that receives the substitution. Enter a formatting template for the substitution into the Value column using the [CMSM Special Format](#).
- Add records to the [Validation Table](#) for the ModuleSubstitution and ModuleConfiguration tables.
- Add records to the [ModuleIgnoreTable Table](#) for the ModuleSubstitution Table and the ModuleConfiguration Table. This ensures that the module is compatible for users who have versions

of Mergemod.dll that are earlier than version 2.0.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Applying a Configurable Merge Module with Customizations

Follow the general guidelines for applying merge modules described in [Applying Merge Modules](#). In addition, merge module consumers must do the following to configure a merge module at the time of installation:

- Use a merge module that is authored to have configurable settings. For more information, see [Creating a Merge Module that Can Be Configured by the End-User](#)
- Use a merge and configuration tool that calls Mergemod.dll version 2.0 or later. Earlier versions of Mergmod.dll cannot configure merge module settings. Authors should create merge modules that provide default values and are compatible with earlier versions of Mergmod.dll.
- Provide customization information when prompted by the configuration client tool. Authors should create merge modules that use default values when a user declines to provide information.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Adding Module Configuration Capability to a Merge Tool

To enable end users to use configurable merge modules, you can author merge and configuration tools to do the following:

- Merge tools should call the functions in Mergemod.dll version 2.0 to merge the module. Earlier versions of Mergemod.dll cannot be used to configure merge modules.
- Client configuration applications should interact with the merge module user to collect the user selections for configurable items.
- Merge tools should expose configuration information authored into the merge module to the client application so that the client can use this information to query the user.
- When a merge tool encounters a configurable item during a merge, it should call the client configuration tool for customization information obtained from the user. The merge tool should make the specified changes to the merge module.
- Configuration applications should enable the user to provide choices for configurable items, but all the possible choices do not need to be revealed to the user. The merge tool can use default values for configurable items that the user does not select.
- If a user does not provide customization information, merge tools should use the default configuration values that are specified in the merge module.
- After a user gives the configuration tool specific selections, the merge tool calls Mergemod.dll to perform the merge.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge Module Reference

This section contains detailed reference information about merge module database tables and summary information streams.

- [Merge Module Database Tables](#)
- [Merge Module Summary Information Stream Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge Module Automation

Mergemod.dll provides a COM object that implements merge operations and source image generation for merge modules. The main object implements interfaces for C/C++ programs and automation clients, including Visual Basic and VBScript.

The preferred method for installing Mergemod.dll is by using the Windows Installer. The Component ID for the component containing the Mergemod.dll COM interface is {FD153241-37EC-11D2-8892-00A0C981B015}. The same binary can be used on all Windows systems and the dll will self-register through regsvr32 on older systems.

Note that Mergemod.dll requires that the Msvcrt.dll be installed on the system.

Note that Mergemod.dll 2.0 is required to create [Configurable Merge Modules](#). Mergemod.dll version 2.0 provides extended functionality at build time through the [IMsmMerge2](#) Interface. This CLSID supports all the existing functionality of the [IMsmMerge](#) Interface provided by Mergemod.dll version 1.0. The default interface on the [Merge](#) object of Mergemod.dll 2.0 is the [IMsmMerge2](#) interface instead of the [IMsmMerge](#) interface.

[Object Model for Mergemod.dll Version 1.0](#)

[Object Model for Mergemod.dll Version 2.0](#)

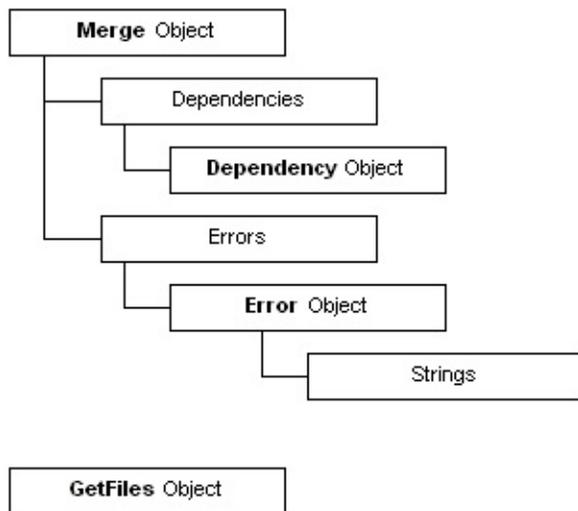
[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Object Model for Mergemod.dll Version 1.0

The object model for Mergemod.dll is organized as follows:



The **Merge object** is the primary object of the model. The **GetFiles object** is a secondary object. Dependencies are collections of **Dependency objects**. Errors are collections of **Error objects**.

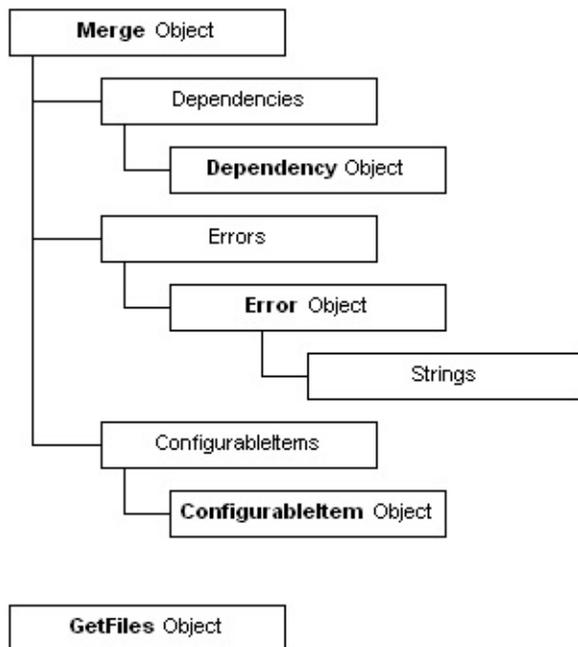
[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Object Model for Mergemod.dll Version 2.0

The object model for Mergemod.dll version 2.0 is organized as follows.



The **Merge object** is the primary object of the model. The **GetFiles object** is a secondary object. Dependencies are collections of **Dependency objects**. Errors are collections of **Error objects**. ConfigurableItems are collections of **ConfigurableItem object**.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Mergemod Objects

The following table lists the Mergemod objects and interfaces.

<b>Object</b>	<b>Interface</b>
<b>ConfigurableItem</b>	<b>IMsmConfigurableItem</b>
<b>ConfigureModule</b>	<b>IMsmConfigureModule</b>
<b>Dependency</b>	<b>IMsmDependency</b>
<b>Error</b>	<b>IMsmError</b>
<b>GetFiles</b>	<b>IMsmGetFiles</b>
<b>Merge</b>	<b>IMsmMerge2IMsmMerge</b>

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ConfigurableItem Object

The **ConfigurableItem object** represents a single row from the [ModuleConfiguration table](#). This is a single configurable "attribute" from the module. The interface consists read-only properties, one for each column in the ModuleConfiguration table. The Interface definition is as follows.

## Methods

The **ConfigurableItem** object does not define any methods.

## Properties

The **ConfigurableItem** object defines the following properties.

<b>Property</b>	<b>Access type</b>	<b>Description</b>
<b>Attributes</b>	Read-only	Returns the value in the Attributes field of this object's record in the ModuleConfiguration table.
<b>Context</b>	Read-only	Returns the value in the Context field of this object's record in the ModuleConfiguration table.
<b>DefaultValue</b>	Read-only	Returns the value in the DefaultValue field of this object's record in the ModuleConfiguration table.
<b>Description</b>	Read-only	Returns the value in the Description field of this object's record in the ModuleConfiguration table.
<b>DisplayName</b>	Read-only	Returns the value in the DisplayName field of this object's record in the ModuleConfiguration table.

<b>Format</b>	Read-only	Returns the value in the Format field of this object's record in the ModuleConfiguration table.
<b>HelpKeyword</b>	Read-only	Returns the value in the HelpKeyword field of this object's record in the ModuleConfiguration table.
<b>HelpLocation</b>	Read-only	Returns the value in the HelpLocation field of this object's record in the ModuleConfiguration table.
<b>Name</b>	Read-only	Returns the value in the Name field of this object's record in the ModuleConfiguration table.
<b>Type</b>	Read-only	Returns the value in the Type field of this object's record in the ModuleConfiguration table.

## C++

interface **IMsmConfigurableItem** : **IDispatch**

### Interface ID

<b>Constant</b>	<b>Value</b>
IID_IMsmConfigurableItem	{4D6E6284-D21D-401E-84F6-909E00B50F71}

### Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h

**DLL**

Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ConfigurableItem.Attributes Property

The **Attributes** property returns the value from the Attributes column of the [ModuleConfiguration table](#). This is a read-only property.

## Syntax

```
Script  
ConfigurableItem.Attributes
```

## C++

See [get\\_Attributes](#) function.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ConfigurableItem.Context Property

The **Context** property returns the value from the Context column of the [ModuleConfiguration table](#). This is a read-only property.

## Syntax

```
Script  
ConfigurableItem.Context
```

## C++

See [get\\_Context](#) function

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ConfigurableItem.DefaultValue Property

The **DefaultValue** property of the [ConfigurableItem](#) object returns the value from the DefaultValue column of the [ModuleConfiguration table](#). This is a read-only property.

## Syntax

```
Script  
ConfigurableItem.DefaultValue
```

## C++

See [get\\_DefaultValue](#) function

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ConfigurableItem.Description Property

The **Description** property returns the value from the Description column of the [ModuleConfiguration table](#). This is a read-only property.

## Syntax

```
Script  
ConfigurableItem.Description
```

## C++

See [get\\_Description](#) function

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ConfigurableItem.DisplayName Property

The **DisplayName** property of the [ConfigurableItem](#) object returns the value from the DisplayName column of the [ModuleConfiguration table](#). This is a read-only property.

## Syntax

```
Script  
ConfigurableItem.DisplayName
```

## C++

See [get\\_DisplayName](#) function.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ConfigurableItem.Format Property

The **Format** property of the [ConfigurableItem](#) object returns the value from the Format column of the [ModuleConfiguration table](#). This is a read-only property.

## Syntax

```
Script  
ConfigurableItem.Format
```

## Remarks

This property can only have the following values.

Constant	Value
msmConfigurableItemText	0
msmConfigurableItemKey	1
msmConfigurableItemInteger	2
msmConfigurableItemBitfield	3

## C++

See [get\\_Format](#) function.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ConfigurableItem.HelpLocation Property

The **HelpLocation** property returns the value from the HelpLocation column of the [ModuleConfiguration table](#). This is a read-only property.

## Syntax

```
Script  
ConfigurableItem.HelpLocation
```

## C++

See [get\\_HelpLocation](#) function.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ConfigurableItem.HelpKeyword Property

The **HelpKeyword** property returns the value from the HelpKeyword column of the [ModuleConfiguration table](#). This is a read-only property.

## Syntax

```
Script  
ConfigurableItem.HelpKeyword
```

## C++

See [get\\_HelpKeyword](#) function.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ConfigurableItem.Name Property

The **Name** property of the [ConfigurableItem](#) object returns the value from the Name column of the ModuleConfiguration table. This is a read-only property.

## Syntax

```
Script  
ConfigurableItem.Name
```

## C++

See [get\\_Name](#) function.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ConfigurableItem.Type Property

The **Type** property returns the value from the Type column of the ModuleConfiguration table . This is a read-only property.

## Syntax

```
Script  
ConfigurableItem.Type
```

## C++

See [get\\_Type Function \(ConfigurableItem Object\)](#).

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ConfigureModule Object

The **ConfigureModule object** is an interface implemented by the client. Mergemod.dll calls methods on the **IMsmConfigureModule** interface to request that the client tool provide configuration information. The module is configured based on the client's responses to calls on this interface.

## Methods

The **ConfigureModule** object defines the following methods.

Method	Description
<b>ProvideTextData</b>	Called by Mergemod.dll to obtain text used to configure the module.
<b>ProvideIntegerData</b>	Called by Mergemod.dll to obtain integers used to configure the module.

## Remarks

### C++

```
interface IMsmConfigureModule : IDispatch
```

### Interface ID

Constant	Value
<b>IID_IMsmConfigureModule</b>	{AC013209-18A7-4851-8A21-2353443D70A0}

## Requirements

--

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ConfigureModule.ProvideTextData Method

The **ProvideTextData** method is called by Mergemod.dll to retrieve text data from the client tool. Mergemod.dll provides the *Name* from the corresponding entry in the ModuleConfiguration table.

The tool should return S\_OK and provide the appropriate customization text in ConfigData. The client tool is responsible for allocating the data, but Mergemod.dll is responsible for releasing the memory. This argument MUST be a **BSTR** object. **LPCWSTR** is NOT accepted.

If the tool does not provide any configuration data for this *Name* value, the function should return S\_FALSE. In this case Mergemod.dll ignores the value of the ConfigData argument and uses the Default value from the ModuleConfiguration table.

Any return code other than S\_OK or S\_FALSE will cause an error to be logged (if a log is open) and will result in the merge failing.

Because this function follows standard **BSTR** convention, null is equivalent to the empty string.

## Syntax

```
Script
ProvideTextData(
    Name,
    ConfigData
)
```

## Parameters

### *Name*

Name of item for which data is being retrieved.

### *ConfigData*

Pointer to customization text.

## Return Value

This method does not return a value.

## Remarks

The client may be called no more than once for each record in the [ModuleConfiguration table](#). Note that Mergemod.dll never makes multiple calls to the client for the same "Name" value. If no record in the ModuleSubstitution table uses the property, an entry in the ModuleConfiguration table causes no calls to the client.

## C++

See [ProvideTextData function](#).

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ConfigureModule.ProvideIntegerData Method

The **ProvideIntegerData** method of the **ConfigureModule object** is called by Mergemod.dll to retrieve integer data from the client tool.

Mergemod.dll provides the *Name* from the corresponding entry in the [ModuleConfiguration table](#).

The tool should return S\_OK and provide the appropriate customization integer in *ConfigData*.

If the tool does not provide any configuration data for this *Name* value, the function should return S\_FALSE. In this case Mergemod.dll ignores the value of the *ConfigData* argument and uses the Default value from the ModuleConfiguration table.

## Syntax

```
Script
ProvideIntegerData(
    Name,
    ConfigData
)
```

## Parameters

*Name*

Name of item for which data is being retrieved.

*ConfigData*

Pointer to customization text.

## Return Value

This method does not return a value.

## Remarks

The client may be called no more than once for each record in the [ModuleConfiguration table](#). Note that Mergemod.dll never makes multiple calls to the client for the same "Name" value. If no record in the ModuleSubstitution table uses the property, an entry in the ModuleConfiguration table causes no calls to the client.

**C++**

See [ProvideIntegerData function](#).

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Dependency Object

The **Dependency** object returns a module that the current module is dependent upon and which has not yet been merged into the current installation database.

## Methods

The **Dependency** object does not define any methods.

## Properties

The **Dependency** object defines the following properties.

<b>Property</b>	<b>Access type</b>	<b>Description</b>
<b>Language</b>	Read-only	Return the language of the module.
<b>Module</b>	Read-only	Return the module ID of the required module.
<b>Version</b>	Read-only	Return the version of the required module.

## C++

```
interface IMsmDependency : IDispatch
```

## Interface ID

<b>Constant</b>	<b>Value</b>
IID_IMsmDependency	{0ADDA82D-2C26-11D2-AD65-00A0C9AF11A6}

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Dependency.Language Property

The read-only **Language** property returns the **LANGID** of the required module.

## Syntax

```
Script  
Dependency.Language
```

## C++

See [get\\_Language Function \(Dependency Object\)](#).

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Dependency.Module Property

The read-only **Module** property of the [Dependency Object](#) returns the ModuleID of the module that is required by the current string in the form of a **BSTR**. The ModuleID is the same form that is used in the [ModuleSignature Table](#).

## Syntax

*Script*

```
Dependency.Module
```

## C++

See the [IMsmDependency\\_get\\_Module](#) method.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Dependency.Version Property

The read-only **Version** property returns the version of the module required by the current merge.

## Syntax

```
Script  
Dependency.Version
```

## C++

See [get\\_Version](#) function.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Error Object

The **Error** object returns the information of a single merge error.

## Methods

The **Error** object does not define any methods.

## Properties

The **Error** object defines the following properties.

<b>Property</b>	<b>Access type</b>	<b>Description</b>
<b>DatabaseKeys</b>	Read-only	Returns the primary keys of the row in the database table that caused the error.
<b>DatabaseTable</b>	Read-only	Returns the table name of the table in the database causing the error.
<b>Language</b>	Read-only	Return the language of the error.
<b>ModuleKeys</b>	Read-only	Returns the primary keys of the row in the module table that caused the error.
<b>ModuleTable</b>	Read-only	Returns the table name of the table in the module causing the error.
<b>Path</b>	Read-only	Return the path to the file or directory causing the error. Currently unused.
<b>Type</b>	Read-only	Return the type of error.

interface **IMsmError** : **IDispatch**

## Interface ID

Constant	Value
IID_IMsmError	{0ADDA828-2C26-11D2-AD65-00A0C9AF11A6}

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Error.DatabaseKeys Property

The read-only **DatabaseKeys** property of the **Error** object returns a string collection that contains the primary keys of the database row causing the error. There is one key per entry in the collection.

## Syntax

```
Script  
Error.DatabaseKeys
```

## Remarks

The client is responsible for releasing the string collection when it is no longer needed.

The collection is empty if the values do not apply to the type of the error. You can determine the type of error by calling the **Type** property of the **Error** object.

## C++

See [get\\_DatabaseKeys](#) function.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Error.DatabaseTable Property

The read-only **DatabaseTable** property of the **Error** object returns the name of the table in the database that caused the error.

## Syntax

```
Script  
Error.DatabaseTable
```

## Remarks

The collection is empty if the values do not apply to the type of the error. You can determine the type of error by calling **Type** property of the **Error** object.

## C++

See [get\\_DatabaseTable](#) function.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Error.Language Property

The read-only **Language** property of the **Error** object returns the **LANGID** of the current error.

## Syntax

```
Script  
Error.Language
```

## Remarks

The value of the **Language** property is -1 unless the error is of type `msmErrorLanguageUnsupported` or `msmErrorLanguageFailed`. You can determine the type of error by calling the **Type** property of the **Error** object.

## C++

See [get\\_Language Function \(Error Object\)](#).

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Error.ModuleKeys Property

The read-only **ModuleKeys** property of the **Error** object returns a pointer to a string collection containing the primary keys of the row in the module causing the error, one key per entry in the collection.

## Syntax

```
Script  
Error.ModuleKeys
```

## Remarks

The client is responsible for releasing the string collection when it is no longer needed.

The collection is empty if the values do not apply to the type of the error. You can determine the type of error by calling the **Type** property of the **Error** object.

## C++

See [get\\_ModuleKeys](#) function.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Error.ModuleTable Property

The read-only **ModuleTable** Property returns the name of the table in the module that caused the error.

## Syntax

```
Script  
Error.ModuleTable
```

## Remarks

The collection is empty if the values do not apply to the type of the error. You can determine the type of error by calling the **Type** property of the **Error** object.

## C++

See [get\\_ModuleTable](#) function.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Error.Path Property

In Windows Installer versions 1.0 and 1.1 the **Path** property is always the empty string. Future versions may use this value to return the path to the file or directory that could not be created.

## Syntax

```
Script  
Error.Path
```

## Remarks

This value is only valid for errors of type `msmErrorFileCreate` or `msmErrorDirCreate`. You can determine the type of error by calling **Type** property of the **Error** object.

## C++

See [get\\_Path](#) function.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Error.Type Property

The read-only **Type** property returns one of the values of `msmErrorType`, indicating the type of error represented by this object.

## Syntax

```
Script  
Error.Type
```

## C++

See [get\\_Type Function](#) function (Error Object).

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# GetFiles Object

The **GetFiles** object retrieves the files needed in a particular language of the module. Requires certain actions be performed through the [Merge object](#) before all parts of this interface returns valid results.

## Methods

The **GetFiles** object does not define any methods.

## Properties

The **GetFiles** object defines the following property.

Property	Access type	Description
<b>ModuleFiles</b>	Read-only	<b>ModuleFiles</b> property returns all the primary keys of the <a href="#">File table</a> for the currently open module.

## C++

```
interface IMsmGetFiles : IDispatch
```

## Interface ID

Constant	Value
IID_IMsmGetFiles	{7041AE26-2D78-11D2-888A-00A0C981B015}

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
----------------	---------------------------

<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# GetFiles.ModuleFiles Property

**ModuleFiles** property of the **GetFiles** object returns all the primary keys of the **File table** for the currently open module. The primary keys are returned as a collection of strings. The module must be opened by a call to the **OpenModule** method of the **Merge object** before calling **ModuleFiles**.

## Syntax

```
Script  
GetFiles.ModuleFiles
```

## Remarks

Note that order of the files listed in the collection may not be in the same sequence as listed in the File table.

If the module has no File table, or contains no files in the particular language, ModuleFiles returns an empty collection of strings.

## C++

See [get\\_ModuleFiles](#) function.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge Object

The **Merge** object provides access to other top-level objects. A **Merge** object must be created before loading the automation support required by COM to access the functions in Mergemod.dll.

Mergemod.dll provides access to the extended functionality at build time through a second version of the existing CLSID. This CLSID supports existing functionality available through the **IMsmMerge** interface, but the default interface on the object (and the associated dual interface) is the **IMsmMerge2** interface instead of the **IMsmMerge** interface.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge.CloseDatabase Method

The [CloseDatabase](#) method of the [Merge](#) object closes the currently open Windows Installer database.

## Syntax

```
Script  
CloseDatabase(  
    bCommit  
)
```

## Parameters

*bCommit*

**TRUE** if changes should be saved, **FALSE** otherwise.

## Return Value

This method does not return a value.

## Remarks

Closing a database clears all dependency information but does not affect any errors that have not been retrieved.

## C++

See [CloseDatabase](#) function.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge.CloseLog Method

The **CloseLog** method of the **Merge** object closes the current log file.

## Syntax

```
Script  
CloseLog( )
```

## Parameters

This method has no parameters.

## Return Value

This method does not return a value.

## C++

See **CloseLog** function.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge.CloseModule Method

The **CloseModule** method of the **Merge** object closes the currently open Windows Installer merge module.

## Syntax

```
Script  
CloseModule()
```

## Parameters

This method has no parameters.

## Return Value

This method does not return a value.

## Remarks

Closing a merge module will not affect any errors that have not been retrieved.

## C++

See **CloseModule** function.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge.ConfigurableItems Property

The read-only **ConfigurableItems** property of the **Merge** object returns a collection **ConfigurableItem** objects, each of which represents a single row from the **ModuleConfiguration table**. Semantically, each interface in the enumerator represents an item that can be configured by the module consumer. The collection is a read-only collection and implements the standard read-only collection interfaces of **Item()**, **Count()** and **\_NewEnum()**. The **IEnumMsmConfigItems** enumerator implements **Next()**, **Skip()**, **Reset()**, and **Clone()** with the standard semantics.

## Syntax

```
Script  
Merge.ConfigurableItems
```

## C++

See [get\\_ConfigurableItems](#) function.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge.Connect Method

The **Connect** method of the **Merge** object connects a module to an additional feature. The module must have already been merged into the database or will be merged into the database. The feature must exist before calling this function.

Changes made to the database are not saved to disk unless the **CloseDatabase** method is called with *bCommit* set to TRUE.

## Syntax

```
Script  
Connect(  
    Feature  
)
```

## Parameters

*Feature*

The name of a feature already existing in the database.

## Return Value

This method does not return a value.

## Remarks

Errors may be retrieved through the **Errors** property. Errors and informational messages are posted to the current log file.

## C++

See **Connect** function.

## Requirements



<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge.CreateSourceImage Method

The **CreateSourceImage** method of the **Merge** object allows the client to extract the files from a module to a source image on disk after a merge, taking into account changes to the module that might have been made during module configuration. The list of files to be extracted is taken from the file table of the module during the merge process. The list of files consists of every file successfully copied from the file table of the module to the target database. File table entries that were not copied due to primary key conflicts with existing rows in the database are not a part of this list. At image creation time, the directory for each of these files comes from the open (post-merge) database. The path specified in the *Path* parameter is the root of the source image for the install. *fLongFileNames* determines whether or not long file names are used for both path segments and final file names. The function fails if no database is open, no module is open, or no merge has been performed.

## Syntax

```
Script
CreateSourceImage(
    Path,
    fLongFileNames,
    pFilePaths
)
```

## Parameters

### *Path*

The path of the root of the source image for the install.

### *fLongFileNames*

*fLongFileNames* determines whether or not long file names are used for both path segments and final file names.

### *pFilePaths*

This is a list of fully-qualified paths for the files that were successfully extracted.

## Return Value

This method does not return a value.

## Remarks

Any files in the destination directory with the same name are overwritten. The path is created if it does not already exist.

## C++

See [CreateSourceImage](#) function.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge.Dependencies Property

The read-only **Dependencies** property of the **Merge** object returns a collection of **Dependency** objects that enumerates a set of unsatisfied dependencies for the current database.

## Syntax

```
Script  
Merge.Dependencies
```

## Remarks

A module does not need to be open to retrieve dependency information.

## C++

See [get\\_Dependencies](#) Function.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge.Errors Property

The read-only **Errors** property of the **Merge** object returns a collection of **Error** objects that is the current set of errors.

## Syntax

```
Script  
Merge.Errors
```

## Remarks

The retrieval is non-destructive. Multiple instances of the error collection may be retrieved by repeatedly calling this method.

## C++

See [get\\_Errors](#) Function.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge.ExtractCAB Method

The **ExtractCAB** method of the **Merge** object extracts the embedded .cab file from a module and saves it as the specified file. The installer creates this file if it does not already exist and overwritten if it does exist.

## Syntax

```
Script  
ExtractCAB(  
    FileName  
)
```

## Parameters

*FileName*

The fully qualified destination file.

## Return Value

This method does not return a value.

## C++

See **ExtractCAB** function.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge.ExtractFiles Method

The **ExtractFiles** method of the [Merge](#) object extracts the embedded .cab file from a module and then writes those files to the destination directory.

## Syntax

```
Script  
ExtractFiles(  
    Path  
)
```

## Parameters

*Path*

The fully qualified destination directory.

## Return Value

This method does not return a value.

## Remarks

Any files in the destination directory with the same name are overwritten. The path is created if it does not already exist.

**ExtractFiles** always extracts files using short file names for the path. To use long file names for the path, use the [ExtractFilesEx method](#).

## C++

See [ExtractFiles](#) function.

## Requirements

Version	
	Mergemod.dll 1.0 or later

<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge.ExtractFilesEx Method

The **ExtractFilesEx** method of the **Merge** object extracts the embedded .cab file from a module and then writes those files to the destination directory.

## Syntax

```
Script  
ExtractFilesEx(  
    Path,  
    fLongFileNames,  
    pFilePaths  
)
```

## Parameters

### *Path*

The fully qualified destination directory.

### *fLongFileNames*

Set to specify using long file names for path segments and final file names.

### *pFilePaths*

This is a list of fully-qualified paths for the files that were successfully extracted.

## Return Value

This method does not return a value.

## Remarks

Any files in the destination directory with the same name are overwritten. The path is created if it does not already exist.

## C++

See [ExtractFilesEx](#) function.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge.Log Method

The **Log** method of the **Merge** object writes a text string to the currently open log file.

## Syntax

```
Script  
Log(  
    Message  
)
```

## Parameters

*Message*

The text string to write to the log file.

## Return Value

This method does not return a value.

## C++

See **Log** function.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge.Merge Method

The **Merge** method of the [Merge](#) object executes a merge of the current database and current module. The merge attaches the components in the module to the feature identified by *Feature*. The root of the module's directory tree is redirected to the location given by *RedirectDir*.

The **Merge** method can only be called once to merge a particular combination of .msi and .msm files.

## Syntax

```
Script
Merge(
    Feature,
    RedirectDir
)
```

## Parameters

### *Feature*

The name of a feature in the database.

### *RedirectDir*

The key of an entry in the [Directory table](#) of the database. This parameter may be null or an empty string.

## Return Value

This method does not return a value.

## Remarks

Once the merge is complete, components in the module are attached to the feature identified by *Feature*. This feature is not created and must be an existing feature. Note that the **Merge** method gets all the feature references in the module and substitutes the feature reference for all occurrences of the null GUID in the module database. For more information, see [Referencing Features in Merge Modules](#).

The module may be attached to additional features using the **Connect** method. Note that calling the **Connect** method only creates feature-component associations. It does not modify the rows that have already been merged in to the database.

Changes made to the database are saved if and only if the **CloseDatabase** method is called with *bCommit* set to TRUE.

If any merge conflicts occur, including exclusions, they are placed in the error enumerator for later retrieval, but does not cause the merge to fail. Errors may be retrieved through the **Errors** property. Errors and informational messages are posted to the current log file.

## C++

See **Merge** function.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge.MergeEx Method

The **MergeEx** method of the **Merge** object is equivalent to the **Merge** function, except that it takes an extra argument. The *pConfiguration* argument is an interface implemented by the client. The argument may be null. The presence of this argument indicates that the client is capable of supporting the configuration functionality, but does not obligate the client to provide configuration data for any specific configurable item.

The **Merge** method executes a merge of the current database and current module. The merge attaches the components in the module to the feature identified by *Feature*. The root of the module's directory tree is redirected to the location given by *RedirectDir*.

## Syntax

```
Script
MergeEx(
    Feature,
    RedirectDir,
    pConfiguration
)
```

## Parameters

### *Feature*

The name of a feature in the database.

### *RedirectDir*

The key of an entry in the **Directory table** of the database. This parameter may be null or an empty string.

### *pConfiguration*

The *pConfiguration* argument is an interface implemented by the client. The argument may be null. The presence of this argument indicates that the client is capable of supporting the configuration functionality, but does not obligate the client to provide configuration data for any specific configurable item.

## Return Value

This method does not return a value.

## Remarks

Once the merge is complete, components in the module are attached to the feature identified by *Feature*. This feature is not created and must be an existing feature. The module may be attached to additional features using the [Connect](#) method.

Changes made to the database are saved if and only if the [CloseDatabase](#) method is called with *bCommit* set to TRUE.

If any merge conflicts occur, including exclusions, they are placed in the error enumerator for later retrieval, but does not cause the merge to fail. Errors may be retrieved through the [Errors](#) property. Errors and informational messages are posted to the current log file.

When the merge fails because of an incorrect module configuration the [MergeEx](#) function returns E\_FAIL. This includes these `msmErrorType` errors: **`msmErrorBadNullSubstitution`**, **`msmErrorBadSubstitutionType`**, **`msmErrorBadNullResponse`**, **`msmErrorMissingConfigItem`**, and **`msmErrorDataRequestFailed`**. These errors cause the merge to stop immediately when the error is encountered. The error object is still added to the enumerator when [MergeEx](#) returns E\_FAIL. For more information about `msmErrorType` errors, see [get\\_Type Function \(Error Object\)](#). All other errors cause [MergeEx](#) to return S\_FALSE and cause the merge to continue.

## C++

See [MergeEx](#) function.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h

**DLL**

Mergemod.dll

---

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge.OpenDatabase Method

The **OpenDatabase** method of the **Merge** object opens a Windows Installer installation database, located at a specified path, that is to be merged with a module.

## Syntax

```
Script  
OpenDatabase(  
    Path  
)
```

## Parameters

*Path*

Path to the database being opened.

## Return Value

This method does not return a value.

## C++

See [OpenDatabase](#) function.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge.OpenLog Method

The **OpenLog** method of the **Merge** object opens a log file that receives progress and error messages. If the log file already exists, the installer appends new messages. If the log file does not exist, the installer creates a log file.

## Syntax

```
Script  
OpenLog(  
    Filename  
)
```

## Parameters

*Filename*

Fully qualified file name pointing to a file to open or create.

## Return Value

This method does not return a value.

## Remarks

Clients may send their own messages to this log file through the **Log** method.

## C++

See **OpenLog** function.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h

**DLL**

Mergemod.dll

---

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge.OpenModule Method

The **OpenModule** method of the **Merge** object opens a Windows Installer merge module in read-only mode. A module must be opened before it can be merged with an installation database.

## Syntax

```
Script  
OpenModule(  
    FileName,  
    Language  
)
```

## Parameters

*FileName*

Fully qualified file name pointing to a merge module.

*Language*

A valid language identifier (LANGID).

## Return Value

This method does not return a value.

## Remarks

This function opens the merge module in read-only mode and excludes other programs from writing to the merge module until the **CloseModule** method is called.

The installer attempts to open the module in the language specified by *Language*, or a more general language. For example, if *Language* is specified as 1033, a module with a default language of 1033, 9, or 0 can be opened in its default language. A *Language* value of 9 opens modules with a default language of 9 or 0. If the default language of the module does not meet the specified requirements, an attempt is made to transform the module to the requested language. If that fails, the module

is transformed to increasingly general languages, all the way to language neutral. If none of the transforms succeed, the module fails to open. In this case, an error is added to the error list of type `msmErrorLanguageUnsupported`. If there is an error transforming the module to the desired language, an error is added to the error list of type `msmErrorLanguageFailed`. For details, see the [Type](#) property of the [Error](#) object. Opening a merge module clears any errors that have not already been retrieved.

## C++

See [OpenModule](#) function.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Mergemod Interfaces

The following are the Mergemod interfaces.

- [IMsmConfigurableItem](#)
- [IMsmConfigureModule](#)
- [IMsmDependency](#)
- [IMsmError](#)
- [IMsmGetFiles](#)
- [IMsmMerge](#)
- [IMsmMerge2](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmConfigurableItem Interface

The **IMsmConfigurableItem** interface manages a single row from the [ModuleConfiguration](#) table. This is a single configurable "attribute" from the module. The interface consists of read-only properties, one for each column in the ModuleConfiguration table.

## Methods

The **IMsmConfigurableItem Interface** interface inherits the methods of the **IDispatch** interface.

In addition, **IMsmConfigurableItem Interface** defines the following methods.

Method	Description
<a href="#">get_Name</a>	Retrieves the <a href="#">Name</a> property.
<a href="#">get_Format</a>	Retrieves the <a href="#">Format</a> property.
<a href="#">get_Type</a>	Retrieves the <a href="#">Type</a> property.
<a href="#">get_Context</a>	Retrieves the <a href="#">Context</a> property.
<a href="#">get_DefaultValue</a>	Retrieves the <a href="#">DefaultValue</a> property.
<a href="#">get_Attributes</a>	Retrieves the <a href="#">Attributes</a> property.
<a href="#">get_DisplayName</a>	Retrieves the <a href="#">DisplayName</a> property.
<a href="#">get_Description</a>	Retrieves the <a href="#">Description</a> property.
<a href="#">get_HelpLocation</a>	Retrieves the <a href="#">HelpLocation</a> property.
<a href="#">get_HelpKeyword</a>	Retrieves the <a href="#">HelpKeyword</a> property.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmConfigurableItem is defined as 4D6E6284-D21D-401E-84F6-909E00B50F71

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmConfigurableItem::get\_Attribute Method

The [get\\_Attributes](#) method retrieves the [Attributes](#) property of the [ConfigurableItem](#) object.

## Syntax

```
C++HRESULT get_Attributes(  
    [out] long *Attributes  
);
```

## Parameters

*Attributes* [out]

A pointer to a location in memory with the format of a configurable item listed in the Attributes column of the [ModuleConfiguration table](#).

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	Invalid argument.
S_OK	The function succeeded.
E_FAIL	No module is open.
E_OUTOFMEMORY	Out of memory.
ERROR_FUNCTION_FAILED as HRESULT	The function failed.
ERROR_INVALID_HANDLE as HRESULT	The function failed.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmConfigurableItem is defined as 4D6E6284-D21D-401E-84F6-909E00B50F71

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmConfigurableItem::get\_Context Method

The [get\\_Context](#) method retrieves the [Context](#) property of the [ConfigurableItem](#) object.

## Syntax

```
C++HRESULT get_Context(  
    [out] BSTR *Context  
);
```

## Parameters

*Context* [out]

A pointer to a location in memory with the context of a configurable item listed in the context column of the [ModuleConfiguration table](#). The client must free the **BSTR** when it is no longer needed.

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	Invalid argument.
S_OK	The function succeeded.
E_FAIL	No module is open.
E_OUTOFMEMORY	Out of memory.
ERROR_FUNCTION_FAILED as HRESULT	The function failed.
ERROR_INVALID_HANDLE as HRESULT	The function failed.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmConfigurableItem is defined as 4D6E6284-D21D-401E-84F6-909E00B50F71

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmConfigurableItem::get\_DefaultValue Method

The `get_DefaultValue` method retrieves the [DefaultValue](#) property of the [ConfigurableItem](#) object.

## Syntax

```
C++HRESULT get_DefaultValue(  
    [out] BSTR *DefaultValue  
);
```

## Parameters

*DefaultValue* [out]

A pointer to a location in memory with the default value of a configurable item listed in the [DefaultValue](#) column of the [ModuleConfiguration table](#). The client must free the **BSTR** when it is no longer required.

## Return Value

The method can return one of the following values.

Value	Meaning
E_FAIL	No module is open.
E_INVALIDARG	Invalid argument.
E_OUTOFMEMORY	Out of memory.
ERROR_FUNCTION_FAILED as HRESULT	The function failed.
ERROR_INVALID_HANDLE as HRESULT	The function failed.

S\_OK

The function succeeded.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmConfigurableItem is defined as 4D6E6284-D21D-401E-84F6-909E00B50F71

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmConfigurableItem::get\_Description Method

The [get\\_Description](#) method retrieves the [Description](#) property of the [ConfigurableItem](#) object.

## Syntax

```
C++HRESULT get_Description(  
    [out] BSTR *Description  
);
```

## Parameters

*Description* [out]

A pointer to a location in memory with the description of a configurable item listed in the Description column of the [ModuleConfiguration table](#). The client must free the **BSTR** when it is no longer needed.

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	Invalid argument.
S_OK	The function succeeded.
E_FAIL	No module is open.
E_OUTOFMEMORY	Out of memory.
ERROR_FUNCTION_FAILED as HRESULT	The function failed.
ERROR_INVALID_HANDLE	The function failed.

as HRESULT

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmConfigurableItem is defined as 4D6E6284-D21D-401E-84F6-909E00B50F71

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmConfigurableItem::get\_DisplayName Method

The [get\\_DisplayName](#) method retrieves the [DisplayName](#) property of the [ConfigurableItem](#) object.

## Syntax

```
C++HRESULT get_DisplayName(  
    [out] BSTR *DisplayName  
);
```

## Parameters

*DisplayName* [out]

A pointer to a location in memory with the format of a configurable item listed in the [DisplayName](#) column of the [ModuleConfiguration table](#). The client must free the **BSTR** when it is no longer needed.

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	Invalid argument.
S_OK	The function succeeded.
E_FAIL	No module is open.
E_OUTOFMEMORY	Out of memory.
ERROR_FUNCTION_FAILED as HRESULT	The function failed.
ERROR_INVALID_HANDLE as HRESULT	The function failed.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmConfigurableItem is defined as 4D6E6284-D21D-401E-84F6-909E00B50F71

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmConfigurableItem::get\_Format Method

The [get\\_Format](#) method retrieves the [Format](#) property of the [ConfigurableItem](#) object.

## Syntax

```
C++HRESULT get_Format(  
    [out] msmConfigurableItemFormat *Format  
);
```

## Parameters

*Format* [out]

A pointer to a location in memory with the format of a configurable item listed in the Format column of the [ModuleConfiguration table](#).

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	Invalid argument.
S_OK	The function succeeded.
E_FAIL	No module is open.
E_OUTOFMEMORY	Out of memory.
ERROR_FUNCTION_FAILED as HRESULT	The function failed.
ERROR_INVALID_HANDLE as HRESULT	The function failed.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmConfigurableItem is defined as 4D6E6284-D21D-401E-84F6-909E00B50F71

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmConfigurableItem::get\_HelpLocation Method

The [get\\_HelpLocation](#) method retrieves the [HelpLocation](#) property of the [ConfigurableItem](#) object.

## Syntax

```
C++HRESULT get_HelpLocation(  
    [out] BSTR *HelpLocation  
);
```

## Parameters

*HelpLocation* [out]

A pointer to a location in memory with the help location of a configurable item listed in the HelpLocation column of the [ModuleConfiguration table](#). The client must free the **BSTR** when it is no longer needed.

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	Invalid argument.
S_OK	The function succeeded.
E_FAIL	No module is open.
E_OUTOFMEMORY	Out of memory.
ERROR_FUNCTION_FAILED as HRESULT	The function failed.
ERROR_INVALID_HANDLE	The function failed.

as HRESULT

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmConfigurableItem is defined as 4D6E6284-D21D-401E-84F6-909E00B50F71

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmConfigurableItem::get\_HelpKey Method

The [get\\_HelpKeyword](#) method retrieves the [HelpKeyword](#) property of the [ConfigurableItem](#) object.

## Syntax

```
C++HRESULT get_HelpKeyword(  
    [out] BSTR *HelpKeyword  
);
```

## Parameters

*HelpKeyword* [out]

A pointer to a location in memory with the help key word of a configurable item listed in the [HelpKeyword](#) column of the [ModuleConfiguration](#) table. The client must free the **BSTR** when it is no longer needed.

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	Invalid argument.
S_OK	The function succeeded.
E_FAIL	No module is open.
E_OUTOFMEMORY	Out of memory.
ERROR_FUNCTION_FAILED as HRESULT	The function failed.
ERROR_INVALID_HANDLE	The function failed.

as HRESULT

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmConfigurableItem is defined as 4D6E6284-D21D-401E-84F6-909E00B50F71

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmConfigurableItem::get\_Name Method

The [get\\_Name](#) method retrieves the [Name](#) property of the [ConfigurableItem](#) object.

## Syntax

```
C++HRESULT get_Name(  
    [out] BSTR *Name  
);
```

## Parameters

*Name* [out]

A pointer to a location in memory with the name of a configurable item as listed in the Name column of the [ModuleConfiguration table](#). The client must free the **BSTR** when it is no longer needed.

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	Invalid argument.
S_OK	The function succeeded.
E_FAIL	No module is open.
E_OUTOFMEMORY	Out of memory.
ERROR_FUNCTION_FAILED as HRESULT	The function failed.
ERROR_INVALID_HANDLE as HRESULT	The function failed.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmConfigurableItem is defined as 4D6E6284-D21D-401E-84F6-909E00B50F71

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmConfigurableItem::get\_Type Method

The [get\\_Type](#) method retrieves the [Type](#) property of the [ConfigurableItem](#) object.

## Syntax

```
C++HRESULT get_Type(  
    [out] BSTR *Type  
);
```

## Parameters

*Type* [out]

A pointer to a location in memory with the format of a configurable item listed in the Type column of the [ModuleConfiguration table](#). The client must free the **BSTR** when it is no longer needed.

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	Invalid argument.
S_OK	The function succeeded.
E_FAIL	No module is open.
E_OUTOFMEMORY	Out of memory.
ERROR_FUNCTION_FAILED as HRESULT	The function failed.
ERROR_INVALID_HANDLE as HRESULT	The function failed.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmConfigurableItem is defined as 4D6E6284-D21D-401E-84F6-909E00B50F71

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmConfigureModule Interface

## Interface

The **IMsmConfigureModule** interface is a callback interface; it enables the client to provide merge configuration information during the merge process.

### Methods

The **IMsmConfigureModule Interface** inherits the methods of the **IDispatch** interface.

In addition, **IMsmConfigureModule Interface** defines the following methods.

Method	Description
<a href="#">ProvideTextData</a>	Retrieves text data from the client tool.
<a href="#">ProvideIntegerData</a>	Retrieves integer data from the client tool.

### Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmConfigureModule is defined as AC013209-18A7-4851-8A21-2353443D70A0

### See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmConfigureModule::ProvideIntegerData Method

The [ProvideIntegerData](#) method retrieves integer data from the client tool. For more information, see the [ProvideIntegerData](#) method of the [ConfigureModule](#) object.

## Syntax

```
C++HRESULT ProvideIntegerData(  
    [in]    BSTR Name,  
    [out]   long *ConfigData  
);
```

## Parameters

*Name* [in]

If the tool does not need to provide any configuration data for this *Name* value, the function should return `S_FALSE`. In this case `Mergemod.dll` ignores the value of the *ConfigData* argument and will use the Default value from the `ModuleConfiguration` table.

*ConfigData* [out]

The tool should return `S_OK` and provide the appropriate customization text in *ConfigData*. The client tool is responsible for allocating the data, but `Mergemod.dll` is responsible for releasing the memory.

## Return Value

Any return code other than `S_OK` or `S_FALSE` causes an error to be logged (if a log is open) and results in the merge failing.

Value	Meaning
<code>S_FALSE</code>	The tool does not need to provide configuration data.

S_OK	Function succeeded.
------	---------------------

## Remarks

The client may be called no more than once for each record in the [ModuleConfiguration table](#). Note that Mergemod.dll never makes multiple calls to the client for the same "Name" value. If no record in the ModuleSubstitution table uses the property, an entry in the ModuleConfiguration table causes no calls to the client.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmConfigureModule is defined as AC013209-18A7-4851-8A21-2353443D70A0

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmConfigureModule::ProvideTextData Method

The [ProvideTextData](#) method retrieves text data from the client tool. For more information, see the [ProvideTextData](#) method of the [ConfigureModule](#) object.

## Syntax

```
C++HRESULT ProvideTextData(  
    [in]    BSTR Name,  
    [out]   BSTR *ConfigData  
);
```

## Parameters

*Name* [in]

If the tool does not provide any configuration data for this value, the function should return S\_FALSE. In this case, Mergemod.dll ignores the value of the *ConfigData* argument and uses the Default value from the ModuleConfiguration table.

*ConfigData* [out]

The tool should return S\_OK and provide the appropriate customization text in *ConfigData*. The client tool is responsible for allocating the data, but Mergemod.dll is responsible for releasing the memory. This argument must be a **BSTR** object. **LPCWSTR** is not accepted.

## Return Value

Any return code other than S\_OK or S\_FALSE causes an error to be logged (if a log is open) and results in the merge failing.

S\_FALSE  
S\_OK

## Remarks

The client may be called no more than once for each record in the [ModuleConfiguration table](#). Note that Mergemod.dll never makes multiple calls to the client for the same "Name" value. If no record in the ModuleSubstitution table uses the property, an entry in the ModuleConfiguration table causes no calls to the client.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmConfigureModule is defined as AC013209-18A7-4851-8A21-2353443D70A0

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmDependency Interface Interface

The **IMsmDependency** interface retrieves details for a single module dependency.

## Methods

The **IMsmDependency Interface** interface inherits the methods of the **IDispatch** interface.

In addition, **IMsmDependency Interface** defines the following methods.

Method	Description
<a href="#">get_Module</a>	Retrieves the <b>Module</b> property of the <b>Dependency</b> object.
<a href="#">get_Language</a>	Retrieves the <b>Language</b> property of the <b>Dependency</b> object.
<a href="#">get_Version</a>	Retrieves the <b>Version</b> property of the <b>Dependency</b> object.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmDependency is defined as 0ADDA82D-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmDependency::get\_Language Method

The **get\_Language** method retrieves the **Language** property of the **Dependency** object. This method returns the **LANGID** of the required module.

## Syntax

```
C++HRESULT get_Language(  
    [out] short *Language  
);
```

## Parameters

*Language* [out]

A pointer to a location in memory that receives the language.

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	Language is null.
S_OK	The function succeeded.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
	IID_IMsmDependency is defined as 0ADDA82D-

**IID**

2C26-11D2-AD65-00A0C9AF11A6

## **See Also**

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmDependency::get\_Module Method

The [get\\_Module](#) method retrieves the [Module](#) property of the [Dependency](#) object. This method returns the ModuleID of the module required by the current string in the form of a **BSTR**. The ModuleID is of the same form as used in the [ModuleSignature table](#).

## Syntax

```
C++HRESULT get_Module(  
    [out] BSTR *Module  
);
```

## Parameters

*Module* [out]

A pointer to a location in memory that is filled in with a **BSTR** value.

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	Module is null
S_OK	The function succeeded

## Remarks

The client is responsible for freeing the resulting string using [SysFreeString](#).

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmDependency is defined as 0ADDA82D-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmDependency::get\_Version Method

The [get\\_Version](#) method retrieves the [Version](#) property of the [Dependency](#) object. This method returns the version of the required module in the form of a **BSTR**.

## Syntax

```
C++HRESULT get_Version(  
    [out] BSTR *Version  
);
```

## Parameters

*Version* [out]

A pointer to a location in memory that is filled in with a **BSTR** value.

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	Version is null.
S_OK	The function succeeded.

## Remarks

The client is responsible for freeing the resulting string using [SysFreeString](#).

## Requirements

---

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmDependency is defined as 0ADDA82D-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmError Interface Interface

The **IMsmError** interface retrieves details about a single merge error.

## Methods

The **IMsmError Interface** interface inherits the methods of the **IDispatch** interface.

In addition, **IMsmError Interface** defines the following methods.

Method	Description
<b>get_Type</b>	Retrieves the <b>Type</b> property of the <b>Error</b> object.
<b>get_Path</b>	Retrieves the <b>Path</b> property of the <b>Error</b> object.
<b>get_Language</b>	Retrieves the <b>Language</b> property of the <b>Error</b> object.
<b>get_DatabaseTable</b>	Retrieves the <b>DatabaseTable</b> property of the <b>Error</b> object.
<b>get_DatabaseKeys</b>	Retrieves the <b>DatabaseKeys</b> property of the <b>Error</b> object.
<b>get_ModuleTable</b>	Retrieves the <b>ModuleTable</b> property of the <b>Error</b> object.
<b>get_ModuleKeys</b>	Retrieves the <b>ModuleKeys</b> property of the <b>Error</b> object.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
----------------	---------------------------

<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmError is defined as 0ADDA828-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmError::get\_DatabaseKeys Method

The **get\_DatabaseKeys** method retrieves the **DatabaseKeys** property of the **Error** object. This method returns a pointer to a string collection containing the primary keys of the row in the database causing the error, one key per entry in the collection.

## Syntax

```
C++HRESULT get_DatabaseKeys(  
    [out] IMsmStrings **ErrorKeys  
);
```

## Parameters

*ErrorKeys* [out]

A pointer to a location in memory that receives a pointer to a string collection.

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	ErrorKeys is null.
S_OK	The function succeeded.

## Remarks

The client is responsible for releasing the string collection when it is no longer required.

The collection is empty if the values do not apply to the type of the error.

You can determine the type of error using [IMsmError::get\\_Type](#).

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmError is defined as 0ADDA828-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmError::get\_DatabaseTable Method

The [get\\_DatabaseTable](#) method retrieves the [DatabaseTable](#) property of the [Error](#) object. The method returns the name of the table in the database that caused the error.

## Syntax

```
C++HRESULT get_DatabaseTable(  
    [out] BSTR *Table  
);
```

## Parameters

*Table* [out]

A pointer to a location in memory that is filled in with a **BSTR** value.

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	Table is null.
E_OUTOFMEMORY	The system was unable to allocate memory for the string.
S_OK	The function succeeded.

## Remarks

The client is responsible for freeing the resulting string using [SysFreeString](#).

The collection is empty if the values do not apply to the type of the error. You can determine the type of error by calling [IMsmError::get\\_Type](#).

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmError is defined as 0ADDA828-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmError::get\_Language Method

The [get\\_Language](#) method retrieves the [Language](#) property of the [Error](#) object. This function returns the **LANGID** of the error.

## Syntax

```
C++HRESULT get_Language(  
    [out] short *Language  
);
```

## Parameters

*Language* [out]

A pointer to a location in memory that receives the language value causing this error.

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	Language is null.
S_OK	The function succeeded.

## Remarks

The function returns -1 unless the error is of type `msmErrorLanguageUnsupported` or `msmErrorLanguageFailed`. You can determine the type of error by calling [IMsmError::get\\_Type](#).

## Requirements

Version
Mergemod.dll 1.0 or later

<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmError is defined as 0ADDA828-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmError::get\_ModuleKeys Method

The **get\_ModuleKeys** method retrieves the **ModuleKeys** property of the **Error** object. This method returns a pointer to a string collection that contains the primary keys of the row in the module causing the error, one key per entry in the collection.

## Syntax

```
C++HRESULT get_ModuleKeys(  
    [out] IMsmStrings **ErrorKeys  
);
```

## Parameters

*ErrorKeys* [out]

A pointer to a location in memory that receives a pointer to a string collection.

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	ErrorKeys is null.
S_OK	The function succeeded.

## Remarks

The client is responsible for releasing the string collection when it is no longer required.

The collection is empty if the values do not apply to the type of the error. You can determine the type of error by calling **IMsmError::get\_Type**.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmError is defined as 0ADDA828-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmError::get\_ModuleTable Method

The [get\\_ModuleTable](#) method retrieves the [ModuleTable](#) property of the [Error](#) object. This method returns the name of the table in the module that caused the error.

## Syntax

```
C++HRESULT get_ModuleTable(  
    [out] BSTR *Table  
);
```

## Parameters

*Table* [out]

A pointer to a location in memory that is filled in with a **BSTR** value.

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	Table is null.
E_OUTOFMEMORY	The system was unable to allocate memory for the string.
S_OK	The function succeeded.

## Remarks

The client is responsible for freeing the resulting string using [SysFreeString](#).

The collection is empty if the values do not apply to the type of the error. You can determine the type of error by calling [IMsmError::get\\_Type](#).

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmError is defined as 0ADDA828-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmError::get\_Path Method

The [get\\_Path](#) method retrieves the [Path](#) property of the [Error](#) object.

## Syntax

```
C++HRESULT get_Path(  
    [out] BSTR *Path  
);
```

## Parameters

*Path* [out]

A pointer to a location in memory that is filled in with a **BSTR** value.

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	Path is null.
E_OUTOFMEMORY	The system was unable to allocate memory for the string.
S_OK	The function succeeded.

## Remarks

The client is responsible for freeing the resulting string using [SysFreeString](#).

In Windows Installer versions 1.0 and 1.1 [get\\_Path](#) always returns the empty string. Future versions of the class may use this function to return the path to the file or directory that could not be created. This value is only valid for errors of type `msmErrorFileCreate` or `msmErrorDirCreate`.

You can determine the type of error by calling [IMsmError::get\\_Type](#).

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmError is defined as 0ADDA828-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmError::get\_Type Method

The **get\_Type** method retrieves the **Type** property of the **Error** object. This method returns a **msmErrorType** value indicating the type of error represented by this object.

## Syntax

```
C++HRESULT get_Type(  
    [out] msmErrorType *ErrorType  
);
```

## Parameters

*ErrorType* [out]

A pointer to a location in memory that receives the type of error.

<b>msmErrorType</b>	<b>Value</b>	<b>Description</b>
msmErrorLanguageUnsupported	1	A request was made to open a module in a language not supported by the module. The general language is supported by the <b>msmErrorLanguageUnsupported</b> property and the requested language is not supported. All <b>Error</b> properties are empty. The <b>OpenModule</b> function returns <b>ERROR_INSTALL_LANGUAGES</b> (as HRESULT).
msmErrorLanguageFailed	2	A request was made to open a module in a language not supported by the module. The general language is supported by the <b>msmErrorLanguageFailed</b> property and the requested language is not supported. All <b>Error</b> properties are empty. The <b>OpenModule</b> function returns <b>ERROR_INSTALL_LANGUAGES</b> (as HRESULT).

		ERROR_INSTALL_LANGUAG (as HRESULT).
msmErrorExclusion	3	The module cannot be merged be is excluded by, another module in msmErrorExclusion to the <b>Type Error</b> object. The <b>ModuleKeys DatabaseKeys property</b> contain of the excluded module's row in <b>ModuleExclusion table</b> . If an exi excludes the module being merge module's ModuleSignature inform ModuleKeys. If the module being an existing module, DatabaseKey excluded module's ModuleSigna other properties are empty (or -1)
msmErrorTableMerge	4	Merge conflict during merge. The property is set to msmErrorTable <b>DatabaseTable property</b> and <b>D property</b> contain the table name the conflicting row in the databas <b>ModuleTable property</b> and <b>Mo contain the table name and prima conflicting row in the module. Th ModuleKeys entries may be null exist in the database. For exampl a generated FeatureComponents merging a <b>configurable merge m</b> may cause these properties to ref not exist in the module.</b>
msmErrorResequeneMerge	5	There was a problem resequenci to contain the necessary merged property is set to msmErrorResec DatabaseTable and DatabaseKey the sequence table name and prin name) of the conflicting row. The ModuleKeys properties contain t name and primary key (action na

		conflicting row. When merging a module, configuration may cause refer to rows that do not exist in 1
msmErrorFileCreate	6	Not used.
msmErrorDirCreate	7	There was a problem creating a c file to disk. The <b>Path</b> property c that could not be created. All oth empty or -1.
msmErrorFeatureRequired	8	A feature name is required to cor no feature name was provided. T set to msmErrorFeatureRequired and DatabaseKeys contain the tal primary keys of the conflicting r ModuleTable and ModuleKeys p table name and primary keys of t merged. When merging a configu module, configuration may cause refer to rows that do not exist in 1 failure is in a generated <b>FeatureC DatabaseTable</b> and <b>DatabaseKey</b> empty and the <b>ModuleTable</b> and properties refer to the row in the causing the failure.
msmErrorBadNullSubstitution	9	Substitution of a Null value into column. This enters msmErrorBa the <b>Type</b> property and enters "M and the keys from the <b>ModuleSu</b> this row into the <b>ModuleTable</b> p <b>ModuleKeys</b> property. All other <b>Error</b> object are set to an empty This error causes the immediate 1 and the <b>MergeEx function</b> to ret
msmErrorBadSubstitutionType	10	Substitution of <b>Text Format Type</b> <b>Type</b> into a <b>Binary Type</b> data col error returns msmErrorBadSubst <b>Type</b> property and enters "Modu

		<p>the keys from the <a href="#">ModuleSubstit</a> row into the <b>ModuleTable</b> properties of the <b>Error</b> object are string or -1.</p> <p>This error causes the immediate f and the <b>MergeEx function</b> to ret</p>
msmErrorMissingConfigItem	11	<p>A row in the <a href="#">ModuleSubstitution</a> configuration item not defined in <a href="#">ModuleConfiguration</a> table. This msmErrorMissingConfigItem in and enters "ModuleSubstitution" the <a href="#">ModuleSubstitution</a> table for <b>ModuleTable</b> property. All other <b>Error</b> object are set to an empty This error causes the immediate f and the <b>MergeEx function</b> to ret</p>
msmErrorBadNullResponse	12	<p>The authoring tool has returned a item marked with the msmConfig attribute. An error of this type ret msmErrorBadNullResponse in th enters "ModuleSubstitution" and <a href="#">ModuleSubstitution</a> table for for <b>ModuleTable</b> property. All other <b>Error</b> object are set to an empty This error causes the immediate f and the <b>MergeEx function</b> to ret</p>
msmErrorDataRequestFailed	13	<p>The authoring tool returned a fail or S_FALSE) when asked for dat type will return msmErrorDataRe <b>Type</b> property and enters "Modu the keys from the <a href="#">ModuleSubstit</a> item into the <b>ModuleTable</b> prop properties of the <b>Error</b> object are string or -1.</p> <p>This error causes the immediate f and the <b>MergeEx function</b> to ret</p>

msmErrorPlatformMismatch	14	Indicates that an attempt was made to load a module into a package that was not designed for that platform. An error of this type returns <code>msmErrorPlatformMismatch</code> in the <code>ErrorCode</code> property. All other properties of the error are empty string or -1. This error causes the failure of the merge and causes the <code>MergeEx</code> function to return <code>E_PLATFORM_MISMATCH</code> .
--------------------------	----	--

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	ErrorType is Null.
S_OK	The function succeeded.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmError is defined as 0ADDA828-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmGetFiles Interface Interface

The **IMsmGetFiles** interface enables the client to retrieve the files needed in a particular language of the module.

## Methods

The **IMsmGetFiles Interface** interface inherits the methods of the **IDispatch** interface.

In addition, **IMsmGetFiles Interface** defines the following method.

Method	Description
<a href="#">get_ModuleFiles</a>	Retrieves the <b>ModuleFiles</b> property of the <b>GetFiles</b> object.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmGetFiles is defined as 7041AE26-2D78-11D2-888A-00A0C981B015

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmGetFiles::get\_ModuleFiles Method

The [get\\_ModuleFiles](#) method retrieves the [ModuleFiles](#) property of the [GetFiles](#) object. This method returns the primary keys in the [File table](#) of the currently open module. The primary keys are returned as a collection of strings. The module must be opened by a call to the [OpenModule](#) function before calling [get\\_ModuleFiles](#).

## Syntax

```
C++HRESULT get_ModuleFiles(  
    [out] IMsmStrings *Files  
);
```

## Parameters

*Files* [out]

Collection of IMsmStrings that are the primary keys of the [File table](#) for the currently open module.

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	Invalid argument.
S_OK	The function succeeded.
E_FAIL	No module is open.
E_OUTOFMEMORY	Out of memory.
ERROR_FUNCTION_FAILED as HRESULT	The function failed.

ERROR_INVALID_HANDLE as HRESULT	The function failed.
------------------------------------	----------------------

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmGetFiles is defined as 7041AE26-2D78-11D2-888A-00A0C981B015

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmMerge Interface

The **IMsmMerge** interface and the **IMsmMerge2** interface provide interfaces to the **Merge** object. The Merge object provides access to other top-level objects. A **Merge** object must be created before loading the automation support required by COM to access the functions in Mergemod.dll.

## Methods

The **IMsmMerge** interface inherits the methods of the **IDispatch** interface.

In addition, **IMsmMerge** defines the following methods.

Method	Description
<b>OpenDatabase</b>	Opens a database to use as the merge target.
<b>OpenModule</b>	Opens a merge module for use as the merge source.
<b>CloseDatabase method</b>	Closes the current database.
<b>CloseModule</b>	Closes the current module
<b>OpenLog</b>	Opens a log file.
<b>CloseLog</b>	Closes the current log file.
<b>Log</b>	Logs a string to the current log file.
<b>Merge</b>	Merges the current module into the current database.
<b>Connect</b>	Connects the components in a module to the specified feature.
<b>ExtractCAB</b>	Extracts the embedded CAB of a Merge Module to a disk file.

<b>ExtractFiles</b>	Creates a source image of the Merge Module beneath the specified path.
---------------------	--

## Properties

The **IMsmMerge** interface defines the following properties.

<b>Property</b>	<b>Access type</b>	<b>Description</b>
<b>Dependencies</b>	Read-only	Returns a collection of all unsatisfied dependencies in the database.
<b>Errors</b>	Read-only	Returns a collection of all errors from the most recent merge operation.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmMerge is defined as 0ADDA82E-2C26-11D2-AD65-00A0C9AF11A6

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmMerge2 Interface

The **IMsmMerge** interface and the **IMsmMerge2** interface provide interfaces to the **Merge object**. The **IMsmMerge2** interface provides a way for the client merge tool to utilize the new configurable-module functionality. Mergemod.dll provides access to the extended functionality at build time through a second version of the existing CLSID. This CLSID supports existing functionality available through the **IMsmMerge** interface, but the default interface on the object (and the associated dual interface) is the **IMsmMerge2** interface instead of the **IMsmMerge** interface.

Requesting this interface does not commit the tool to using the new functionality. The interface supports both the standard and the "Ex" versions of the appropriate interface calls.

The Merge object provides access to other top-level objects. A Merge object must be created before loading the automation support required by COM to access the functions in Mergemod.dll.

## Methods

The **IMsmMerge2** interface inherits the methods of the **IDispatch** interface.

In addition, **IMsmMerge2** defines the following methods.

Method	Description
<b>OpenDatabase</b>	Opens a database to use as the merge target.
<b>OpenModule</b>	Opens a merge module for use as the merge source.
<b>CloseDatabase</b>	Closes the current database.
<b>CloseModule</b>	Closes the current module
<b>OpenLog</b>	Opens a log file.
<b>CloseLog</b>	Closes the current log file.

<b>Log</b>	Logs a string to the current log file.
<b>get_Errors</b>	Returns a collection of all errors from the most recent merge operation.
<b>get_Dependencies</b>	Returns a collection of all unsatisfied dependencies in the database.
<b>Merge</b>	Merges the current module into the current database.
<b>Connect</b>	Connects the components in a module to the specified feature.
<b>ExtractCAB</b>	Extracts the embedded CAB of a Merge Module to a disk file.
<b>ExtractFiles</b>	Creates a source image of the Merge Module beneath the specified path.
<b>MergeEx</b>	Merges the current module into the current database.
<b>ExtractFilesEx</b>	Creates a source image of the Merge Module beneath the specified path.
<b>get_ConfigurableItems</b>	Returns a collection of configurable items in the database.
<b>CreateSourceImage</b>	Extracts files from a module to a source image on disk after a merge, with configuration changes.

## Properties

The **IMsmMerge2** interface defines the following properties.



<b>Property</b>	<b>Access type</b>	<b>Description</b>
<b>ConfigurableItems</b>	Read-only	Returns a collection of configurable items in the database.
<b>Dependencies</b>	Read-only	Returns a collection of all unsatisfied dependencies in the database.
<b>Errors</b>	Read-only	Returns a collection of all errors from the most recent merge operation.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmMerge2 is defined as 351A72AB-21CB-47ab-B7AA-C4D7B02EA305

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmMerge::CloseDatabase Method

The **CloseDatabase** method closes the currently open Windows Installer database. For more information, see the [CloseDatabase](#) method of the [Merge object](#).

**IMsmMerge2::CloseDatabase** Mergemod.dll version 2.0 or later.  
**IMsmMerge::CloseDatabase** All Mergemod.dll versions.

## Syntax

```
C++HRESULT CloseDatabase(  
    VARIANT_BOOL bCommit  
);
```

## Parameters

*bCommit*

**TRUE** if changes should be saved, **FALSE** otherwise.

## Return Value

The **CloseDatabase** function returns the following values.

Value	Meaning
E_FAIL	There was an error closing the database. The state of the <a href="#">IMsmMerge</a> or <a href="#">IMsmMerge2</a> interface is now in an undefined state.
S_FALSE	No database was open.
S_OK	The function succeeded.
STG_E_CANTSAVE as HRESULT	Unable to save database. This error is not generated if <i>bCommit</i> is <b>FALSE</b> .

## Remarks

This function closes the currently open database. Closing a database clears all dependency information but does not affect any errors that have not been retrieved.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmMerge is defined as 0ADDA82E-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmMerge::CloseLog Method

The **CloseLog** function method closes the current log. For more information, see the **CloseLog** method of the **Merge** object.

**IMsmMerge2::CloseLog** Mergemod.dll version 2.0 or later.  
**IMsmMerge::CloseLog** All Mergemod.dll versions.

## Syntax

```
C++HRESULT CloseLog();
```

## Parameters

This method has no parameters.

## Return Value

The method can return one of the following values.

Value	Meaning
E_FAIL	There was an error closing the log file.
S_FALSE	No log file was open.
S_OK	The function succeeded.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmMerge is defined as 0ADDA82E-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmMerge::CloseModule Method

The **CloseModule** method closes the currently open Windows Installer merge module. For more information, see the [CloseModule](#) method of the [Merge object](#).

**IMsmMerge2::CloseDatabase**      Mergemod.dll version 2.0 or later.  
**IMsmMerge::CloseDatabase**      All Mergemod.dll versions.

## Syntax

```
C++HRESULT CloseModule();
```

## Parameters

This method has no parameters.

## Return Value

The **CloseModule** function returns the following values.

Value	Meaning
E_FAIL	There was an error closing the module. The state of the <a href="#">IMsmMerge</a> or <a href="#">IMsmMerge2</a> interface is now undefined.
S_FALSE	No module was open.
S_OK	The function succeeded.

## Remarks

Closing a merge module does not affect any errors that have not been retrieved.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmMerge is defined as 0ADDA82E-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmMerge::Connect Method

The **Connect** method connects a module that has been, or will be, merged into the database to an additional feature. For more information, see the **Connect** method of the **Merge** object.

**IMsmMerge2::Connect** Mergemod.dll version 2.0 or later.  
**IMsmMerge::Connect** All Mergemod.dll versions.

## Syntax

```
C++HRESULT Connect(  
    [in] BSTR Feature  
);
```

## Parameters

*Feature* [in]

The name of a feature in the database. A **LPCWSTR** may be used in place of a **BSTR**.

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	One of the arguments is invalid.
E_FAIL	The connect failed.
S_OK	The function succeeded.

## Remarks

The feature must exist before this function is called. Errors may be retrieved using **get\_Errors**. Errors and informational messages are

posted to the current log file.

Changes made to the database are not be saved to disk unless [CloseDatabase](#) function is called with *bCommit* set to TRUE.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmMerge is defined as 0ADDA82E-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmMerge2::CreateSourceImage Method

The [CreateSourceImage](#) method enables the client to extract the files from a module to a source image on disk after a merge, taking into account changes to the module that might have been made during module configuration. For more information, see the [CreateSourceImage](#) method of the [Merge](#) object.

## Syntax

```
C++HRESULT CreateSourceImage(  
    [in]    BSTR Path,  
    [in]    VARIANT_BOOL fLongFileNames,  
    [out]   IMsmStrings **pFilePaths  
);
```

## Parameters

*Path* [in]

The path of the root of the source image for the install.

*fLongFileNames* [in]

*fLongFileNames* determines whether or not long file names are used for both path segments and final file names.

*pFilePaths* [out]

A pointer to a memory location. This memory location receives a second pointer to a string enumerator containing a list of fully-qualified paths for the files that were extracted. The list is empty if no files can be extracted. This argument may be null. No list is provided if *pFilePaths* is Null.

## Return Value

If the method succeeds, it returns S\_OK. Otherwise, it returns an HRESULT error code.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmMerge2 is defined as 351A72AB-21CB-47ab-B7AA-C4D7B02EA305

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmMerge::ExtractCAB Method

The **ExtractCAB** method extracts the embedded .cab file from a module and saves it as the specified file. The installer creates this file if it does not already exist and overwrites the file if it does exist. For more information, see the **ExtractCAB** method of the **Merge** object.

**IMsmMerge2::ExtractCAB**    Mergemod.dll version 2.0 or later.  
**IMsmMerge::ExtractCAB**    All Mergemod.dll versions.

## Syntax

```
C++HRESULT ExtractCAB(  
    [in] BSTR FileName  
);
```

## Parameters

*FileName* [in]

The fully qualified destination file. A **LPCWSTR** may be used in place of a **BSTR**.

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	One of the arguments is invalid.
ERROR_OPEN_FAILED as HRESULT	Could not create the output file.
ERROR_WRITE_FAULT as HRESULT	Could not write data to the output file.
E_FAIL	Unable to access embedded .cab file.
S_FALSE	No embedded .cab file was found.

S_OK	The function succeeded.
------	-------------------------

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmMerge is defined as 0ADDA82E-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmMerge::ExtractFiles Method

The [ExtractFiles](#) method extracts the embedded .cab file from a module and then writes those files to the destination directory. For more information, see the [ExtractFiles](#) method of the [Merge](#) object.

[IMsmMerge2::ExtractFiles](#)    Mergemod.dll version 2.0 or later.  
[IMsmMerge::ExtractFiles](#)    All Mergemod.dll versions.

## Syntax

```
C++HRESULT ExtractFiles(  
    [in] BSTR Path  
);
```

## Parameters

*Path* [in]

The fully qualified destination directory. A **LPCWSTR** may be used in place of a **BSTR**.

## Return Value

The method can return one of the following values.

Value	Meaning
ERROR_CANNOT_MAKE as HRESULT	Could not create the output path.
ERROR_OPEN_FAILED as HRESULT	Could not create the output file.
ERROR_WRITE_FAULT as HRESULT	Could not write data to the output file.
E_FAIL	Unable to access embedded .cab file, or create temporary file.

S_FALSE	No embedded .cab file was found.
S_OK	The function succeeded.

## Remarks

Any files in the destination directory with the same name are overwritten. The path is created if it does not already exist.

**ExtractFiles** always extracts files using short file names for the path. To use long file names for the path, use the **ExtractFilesEx** function.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmMerge is defined as 0ADDA82E-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmMerge2::ExtractFilesEx Method

The [ExtractFilesEx](#) method extracts the embedded .cab file from a module and then writes those files to the destination directory. For more information, see the [ExtractFilesEx](#) method of the [Merge](#) object.

## Syntax

```
C++HRESULT ExtractFilesEx(  
    [in]    BSTR Path,  
    [in]    VARIANT_BOOL fLongFileNames,  
    [out]   IMsmStrings **pFilePaths  
);
```

## Parameters

*Path* [in]

The fully qualified destination directory. A **LPCWSTR** may be used in place of a **BSTR**.

*fLongFileNames* [in]

Set to specify using long file names for path segments and final file names.

*pFilePaths* [out]

A pointer to a memory location. This memory location receives a second pointer to a string enumerator containing a list of fully-qualified paths for the files that were extracted. The list is empty if no files can be extracted. This argument may be null. No list is provided if *pFilePaths* is Null.

## Return Value

The method can return one of the following values.

Value	Meaning
ERROR_CANNOT_MAKE as HRESULT	Could not create the output path.

ERROR_OPEN_FAILED as HRESULT	Could not create the output file.
ERROR_WRITE_FAULT as HRESULT	Could not write data to the output file.
E_FAIL	Unable to access embedded .cab file, or create temporary file.
S_FALSE	No embedded .cab file was found.
S_OK	The function succeeded.

## Remarks

Any files in the destination directory with the same name are overwritten. The path is created if it does not already exist.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmMerge2 is defined as 351A72AB-21CB-47ab-B7AA-C4D7B02EA305

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmMerge2::get\_ConfigurableItems Method

The [get\\_ConfigurableItems](#) method retrieves the [ConfigurableItems](#) property of the [Merge](#) object.

## Syntax

```
C++HRESULT get_ConfigurableItems(  
    [out] IMsmConfigurableItems **ConfigurableItems  
);
```

## Parameters

*ConfigurableItems* [out]

Pointer to a memory location containing another pointer to an [IMsmConfigurableItems](#) interface.

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	The <i>ConfigurableItems</i> pointer is NULL
E_OUTOFMEMORY	The system ran out of memory.
S_OK	The function succeeded.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h

**DLL**

Mergemod.dll

**IID**

IID\_IMsmMerge2 is defined as 351A72AB-21CB-47ab-B7AA-C4D7B02EA305

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmMerge::get\_Dependencies Method

The `get_Dependencies` method retrieves the **Dependencies** property of the **Merge** object.

**IMsmMerge2::get\_Dependencies**    Mergemod.dll version 2.0 or later.  
**IMsmMerge::get\_Dependencies**    All Mergemod.dll versions.

## Syntax

```
C++HRESULT get_Dependencies(  
    IMsmDependencies **Dependencies  
);
```

## Parameters

### *Dependencies*

Pointer to a memory location to be filled with a pointer to a collection of unsatisfied dependencies for the current database. If there is an error, the memory location pointed to by *Dependencies* is set to null.

## Return Value

The `get_Dependencies` function returns the following values.

Value	Meaning
E_FAIL	There was no database open.
E_INVALIDARG	The <i>Dependencies</i> pointer is null.
E_OUTOFMEMORY	The system ran out of memory.
E_UNEXPECTED	Unable to verify dependencies due to internal error.
S_OK	The function succeeded.

## Remarks

A module does not need to be open to retrieve dependency information. The client is responsible for releasing the interface returned by this function.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmMerge is defined as 0ADDA82E-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmMerge::get\_Errors Method

The [get\\_Errors](#) method retrieves the [Errors](#) property of the [Merge](#) object. This retrieves the current collection of errors.

[IMsmMerge2::get\\_Errors](#) Mergemod.dll version 2.0 or later.  
[IMsmMerge::get\\_Errors](#) All Mergemod.dll versions.

## Syntax

```
C++HRESULT get_Errors(  
    [out] IMsmErrors **Errors  
);
```

## Parameters

*Errors* [out]

Pointer to a memory location containing another pointer to an [IMsmErrors](#) interface.

## Return Value

The method can return one of the following values.

Value	Meaning
E_INVALIDARG	The <i>Errors</i> pointer is NULL.
E_OUTOFMEMORY	The system is out of memory.
S_OK	The function succeeded.

## Remarks

The retrieval is non-destructive, meaning that several instances of the error collection may be retrieved by repeatedly calling this method.

If there is an error, the memory location pointed to by *Errors* is set to null. The client is responsible for releasing the interface returned by this function.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmMerge is defined as 0ADDA82E-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmMerge::Log Method

The **Log** method writes a text string to the currently open log file. For more information, see the **Log** method of the **Merge** object.

**IMsmMerge2::Log** Mergemod.dll version 2.0 or later.  
**IMsmMerge::Log** All Mergemod.dll versions.

## Syntax

```
C++HRESULT Log(  
    BSTR Message  
);
```

## Parameters

*Message*

The text string to display. A **LPCWSTR** may be used instead of a **BSTR**.

## Return Value

The method can return one of the following values.

Value	Meaning
E_FAIL	There was an error writing to the log file.
E_INVALIDARG	The argument is invalid.
S_FALSE	No log file is open.
S_OK	The function succeeded.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
----------------	---------------------------

<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmMerge is defined as 0ADDA82E-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmMerge::Merge Method

The **Merge** method executes a merge of the current database and current module. The merge attaches the components in the module to the feature identified by *Feature*. The root of the module's directory tree is redirected to the location given by *RedirectDir*. For more information, see the **Merge** method of the [Merge](#) object.

**IMsmMerge2::Merge** Mergemod.dll version 2.0 or later.

**IMsmMerge::Merge** All Mergemod.dll versions.

## Syntax

```
C++HRESULT Merge(  
    [in] const BSTR Feature,  
    [in] const BSTR RedirectDir  
);
```

## Parameters

*Feature* [in]

The name of a feature in the database. A **LPCWSTR** can be used in place of a **BSTR**.

*RedirectDir* [in]

The key of an entry in the Directory table of the database. A **LPCWSTR** can be used in place of a **BSTR**. This parameter can be null or an empty string.

## Return Value

The **Merge** function returns the following values.

Value	Meaning
E_FAIL	The merge failed catastrophically. This indicates an operational error, and is not the normal error return for a failed merge.
S_FALSE	The function succeeded, but there were errors

	and the merge itself may not be valid.
E_INVALIDARG	One of the arguments is invalid.
E_OUTOFMEMORY	The system ran out of memory and could not complete the operation.
S_OK	The function succeeded.

## Remarks

This function executes a merge of the current database and current module. The root of the module's directory tree is redirected to the location given by *RedirectDir*. If any merge conflicts occur, including exclusions, they are placed in the error enumerator for later retrieval, but does not cause the merge to fail. Errors can be retrieved using the [get\\_Errors](#) function. Errors and informational messages are posted to the current log file.

Note that the **Merge** function gets all the feature references in the module and substitutes the feature reference for all occurrences of the null GUID in the module database. For more information, see [Referencing Features in Merge Modules](#).

Once the merge is complete, components in the module are attached to the feature identified by *Feature*. This feature must already exist and is not created.

The module can be attached to additional features using the [Connect](#) function. Note that calling the **Connect** function only creates feature-component associations. It does not modify the rows that have already been merged in to the database.

Changes made to the database are not saved to disk unless the [CloseDatabase](#) function is called with *bCommit* set to TRUE.

## Requirements

### Version

Mergemod.dll 1.0 or later

<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmMerge is defined as 0ADDA82E-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmMerge2::MergeEx Method

The [MergeEx](#) method executes a merge of the current database and current module. The merge attaches the components in the module to the feature identified by *Feature*. The root of the module's directory tree is redirected to the location given by *RedirectDir*. For more information, see the [MergeEx](#) method of the [Merge](#) object.

## Syntax

```
C++HRESULT MergeEx(  
    [in] const BSTR Feature,  
    [in] const BSTR RedirectDir,  
    [in] IMsmConfigureModule *pConfiguration  
);
```

## Parameters

*Feature* [in]

The name of a feature in the database. A **LPCWSTR** may be used in place of a **BSTR**.

*RedirectDir* [in]

The key of an entry in the Directory table of the database. A **LPCWSTR** may be used in place of a **BSTR**. This parameter may be null or an empty string.

*pConfiguration* [in]

The *pConfiguration* argument is an interface implemented by the client. The argument may be null. The presence of this argument indicates that the client tool is capable of modifying configurable merge modules. The presence of this argument does not require the client to provide configuration data for any specific configurable item.

## Return Value

The method can return one of the following values.

Value	Meaning
-------	---------

E_OUTOFMEMORY	There system ran out of memory and could not complete the operation.
E_INVALIDARG	One of the arguments is invalid.
E_FAIL	The merge was stopped due to an error. Some tables may not have been merged. See the Remarks section for more information.
S_FALSE	The function succeeded, but there were errors and the merge itself may not be valid.
S_OK	The function succeeded.

## Remarks

This function executes a merge of the current database and current module. The root of the module's directory tree is redirected to the location given by *RedirectDir*. If any merge conflicts occur, including exclusions, they are placed in the error enumerator for later retrieval, but does not cause the merge to fail. Errors may be retrieved using [get\\_Errors](#) function. Errors and informational messages will be posted to the current log file.

Once the merge is complete, components in the module are attached to the feature identified by *Feature*. This feature must already exist and is not created. The module may be attached to additional features using [Connect](#) function.

Changes made to the database will not be saved to disk unless [CloseDatabase](#) function is called with *bCommit* set to TRUE.

When the merge fails because of an incorrect module configuration the function returns E\_FAIL. This includes these *msmErrorType* errors: *msmErrorBadNullSubstitution*, *msmErrorBadSubstitutionType*, *msmErrorBadNullResponse*, *msmErrorMissingConfigItem*, and *msmErrorDataRequestFailed*. These errors cause the merge to stop immediately when the error is encountered. The error object is still added to the enumerator when [MergeEx](#) returns E\_FAIL. For more information

about `msmErrorType` errors, see [get\\_Type Function \(Error Object\)](#). All other errors cause `MergeEx` to return `S_FALSE` and cause the merge to continue.

## Requirements

<b>Version</b>	Mergemod.dll 2.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmMerge2 is defined as 351A72AB-21CB-47ab-B7AA-C4D7B02EA305

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmMerge::OpenDatabase Method

The [OpenDatabase](#) method opens a Windows Installer installation database, located at a specified path, that is to be merged with a module. For more information, see the [OpenDatabase](#) method of the [Merge](#) object.

[IMsmMerge2::OpenDatabase](#) Mergemod.dll version 2.0 and later.  
[IMsmMerge::OpenDatabase](#) All Mergemod.dll versions.

## Syntax

```
C++HRESULT OpenDatabase(  
    const BSTR Path  
);
```

## Parameters

*Path*

Path to the database being opened.

## Return Value

The method can return one of the following values.

Value	Meaning
E_FAIL	There was an error opening the database.
S_OK	The function succeeded.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll

**IID**

IID\_IMsmMerge is defined as 0ADDA82E-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmMerge::OpenLog Method

The **OpenLog** method opens a log file that receives progress and error messages. If the log file already exists, the installer appends new messages. If the log file does not exist, the installer creates a log file. For more information, see the **OpenLog** method of the **Merge** object.

**IMsmMerge2::OpenLog** Mergemod.dll version 2.0 or later.

**IMsmMerge::OpenLog** All Mergemod.dll versions.

## Syntax

```
C++HRESULT OpenLog(  
    [in] BSTR FileName  
);
```

## Parameters

*FileName* [in]

Fully qualified file name pointing to a file to open or create. A **LPCWSTR** may be used in place of a **BSTR**.

## Return Value

The method can return one of the following values.

Value	Meaning
ERROR_TOO_MANY_OPEN_FILES as HRESULT	There is already a log file open.
ERROR_OPEN_FAILED as HRESULT	The file could not be opened or created.
S_OK	The function succeeded.

## Remarks

This function opens a log file to receive progress and error messages. If the log file already exists, new messages get appended to the log. If the log file does not exist it is created.

Clients may send their own messages to this log file using [Log](#).

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmMerge is defined as 0ADDA82E-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IMsmMerge::OpenModule Method

The **OpenModule** method opens a Windows Installer merge module in read-only mode. A module must be opened before it can be merged with an installation database. For more information, see the [OpenModule](#) method of the [Merge](#) object.

**IMsmMerge2::OpenModule** Mergemod.dll version 2.0 and later.  
**IMsmMerge::OpenModule** All Mergemod.dll versions.

## Syntax

```
C++HRESULT OpenModule(  
    [in] BSTR FileName,  
    [in] short Language  
);
```

## Parameters

*FileName* [in]

Fully-qualified file name that points to a merge module. A **LPCWSTR** can be used in place of a **BSTR**.

*Language* [in]

A language identifier (**LANGID**).

## Return Value

The **OpenModule** function returns the following values.

Value	Meaning
E_ABORT	The file specified is an Windows Installer database, but is not a merge module (missing <a href="#">ModuleSignature table</a> ).

ERROR_INSTALL_LANGUAGE_UNSUPPORTED as HRESULT	The language is not supported by the module.
ERROR_INSTALL_TRANSFORM_FAILURE as HRESULT	The language is supported by the module, but there was an error applying the transform.
ERROR_OPEN_FAILED as HRESULT	The file could not be opened as an Windows Installer database.
ERROR_TOO_MANY_OPEN_FILES as HRESULT	There is already a module open. Closes the current module first.
S_OK	The function succeeded.

## Remarks

This function opens the merge module in read-only mode (MSIDBOPEN\_READONLY), and excludes other programs from writing to the merge module until the [CloseModule](#) function is called. A merge module must be opened before it can be merged.

The installer attempts to open the module in the language specified by *Language* or in any more general language. For example, if 1033 is specified by the *Language* value, a module with a default language of 1033, 9, or 0 is opened in its default language. A *Language* value of 9 opens modules with a default language of 9 or 0. If the default language of the module does not meet the specified requirements, an attempt is made to transform the module into the requested language. If that fails, the installer attempts to transform the module into increasingly general

languages, all the way to language neutral. If none of the transforms succeed, the module fails to open. In this case, an error is added to the error list of type `msmErrorLanguageUnsupported` and the function returns `ERROR_INSTALL_LANGUAGE_UNSUPPORTED` as HRESULT.

If there is an error transforming the module to the desired language, an error is created of type `msmErrorLanguageFailed` and the function returns `ERROR_INSTALL_TRANSFORM_FAILURE` as HRESULT.

For more information, see the [Type](#) property of the [Error](#) object.

Opening a merge module clears any errors that have not already been retrieved.

## Requirements

<b>Version</b>	Mergemod.dll 1.0 or later
<b>Header</b>	Mergemod.h
<b>DLL</b>	Mergemod.dll
<b>IID</b>	IID_IMsmMerge is defined as 0ADDA82E-2C26-11D2-AD65-00A0C9AF11A6

## See Also

[Merge Module Automation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge Module Database Tables

The following tables are required in a standard merge module.

Table name	Comment
<a href="#">Component</a>	(REQUIRED)
<a href="#">Directory</a>	(REQUIRED)
<a href="#">FeatureComponents</a>	(REQUIRED)
<a href="#">File</a>	(REQUIRED)
<a href="#">ModuleSignature</a>	(REQUIRED) Merged into the installer database. Lists the information identifying a merge module.
<a href="#">ModuleComponents</a>	(REQUIRED) Merged into the installer database. Lists all the components in the merge module.

The following tables only occur in merge modules or other installer databases that have already been combined with a merge module.

Table name	Comment
<a href="#">ModuleDependency</a>	Merged into the installer database. Lists other merge modules required by this merge module.
<a href="#">ModuleExclusion</a>	Merged into the installer database. Lists other merge modules that are incompatible with this merge module.

The following ModuleSequence tables only occur in merge modules.

Table name	Comment
<a href="#">ModuleAdminUISequence</a>	Merges actions into the <a href="#">AdminUISequence table</a> .
<a href="#">ModuleAdminExecuteSequence</a>	Merges actions into the <a href="#">AdminExecuteSequence table</a> .

<a href="#">ModuleAdvtUISequence</a>	Do not use this table. For details, see <a href="#">AdvtUISequence table</a> .
<a href="#">ModuleAdvtExecuteSequence</a>	Merges actions into the <a href="#">AdvtExecuteSequence table</a> .
<a href="#">ModuleIgnoreTable</a>	Lists tables in the module that are not merged into the .msi file.
<a href="#">ModuleInstallUISequence</a>	Merges actions into the <a href="#">InstallUISequence table</a> .
<a href="#">ModuleInstallExecuteSequence</a>	Merges actions into the <a href="#">InstallExecuteSequence table</a> .

The following tables are required in every configurable merge module. Mergemod.dll 2.0 or later is required to create configurable merge module. For details, see [Configurable Merge Modules](#).

<b>Table name</b>	<b>Comment</b>
<a href="#">ModuleSubstitution Table</a>	(REQUIRED) This table is not merged into the target installation database. Specifies the configurable fields in the target database and provides a template for the configuration of each field.
<a href="#">ModuleConfiguration Table</a>	(REQUIRED) This table is not merged into the target installation database. Identifies the configurable attributes of the module.

The following installer tables cannot occur in a standard merge module.

[BBControl](#)  
[Billboard](#)  
[CCPSearch](#)  
[Error](#)  
[Feature](#)  
[LaunchCondition table](#)

Media  
Patch  
Upgrade

The following installer tables are optional in merge modules.

ActionText  
AdminExecuteSequence  
AdminUISequence  
AdvtExecuteSequence  
AdvtUISequence  
AppId  
AppSearch  
BindImage  
CheckBox  
Class  
ComboBox  
CompLocator  
Control  
ControlCondition  
CreateFolder  
CustomAction  
Dialog  
DrLocator  
DuplicateFile  
Environment  
EventMapping  
Extension  
Font  
Icon  
IniFile  
IniLocator  
InstallExecuteSequence  
InstallUISequence  
ListBox  
ListView  
MIME  
MoveFile  
ODBCAttribute  
ODBCDataSource

ODBCDriver  
ODBCSourceAttribute  
ODBCTranslator  
ProgID Table  
Property  
PublishComponent  
RadioButton  
Registry Table  
RegLocator  
RemoveFile  
RemoveIniFile  
RemoveRegistry  
ReserveCost  
SelfReg  
ServiceControl  
ServiceInstall  
Shortcut  
Signature  
TextStyle  
TypeLib  
UIText  
Verb

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ModuleSignature Table

The ModuleSignature Table is a required table. It contains all the information necessary to identify a merge module. The merge tool adds this table to the .msi file if one does not already exist. The ModuleSignature table in a merge module has only one row containing the ModuleID, Language, and Version. However, the ModuleSignature table in an .msi file has a row containing this information for each .msm file that has been merged into it.

Merge and verification tools check the ModuleSignature table in .msi files to determine if it has all of the dependent merge modules required by the current merge module (see [ModuleDependency Table](#)) and whether the installation package was previously merged with any conflicting merge modules (see [ModuleExclusion Table](#)).

The ModuleSignature table has the following columns.

Column	Type	Key	Nullable
ModuleID	Identifier	Y	N
Language	Integer	Y	N
Version	Version		N

## Columns

### ModuleID

An identifier that uniquely identifies the merge module. Two merge modules cannot have the same ModuleID unless the merge module is entirely backward compatible with its predecessor. You can create a GUID for this field using a utility such as GUIDGEN. The ModuleID column is a primary key for the table and therefore it must follow the naming convention in [Naming Primary Keys in Merge Module Databases](#). For example, if the readable name of the merge module is MyLibrary and the GUID is {880DE2F0-CDD8-11D1-A849-006097ABDE17}, the entry in the ModuleID column becomes MyLibrary.880DE2F0\_CDD8\_11D1\_A849\_006097ABDE17.

## Language

The Language identifier specifies the default language for the merge module. The language identifier is in decimal format, for example, U.S. English is 1033. The language used by the merge module can be changed by applying a transform to the merge module before merging.

## Version

The Version field contains a string that describes the major and minor versions of the merge module.

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE25](#)

## See Also

[Multiple Language Merge Modules](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ModuleComponents Table

The ModuleComponents table contains a list of the components found in the merge module.

The ModuleComponents table has the following columns.

Column	Type	Key	Nullable
Component	Identifier	Y	N
ModuleID	Identifier	Y	N
Language	Integer	Y	N

## Columns

### Component

An identifier referring to a component in the merge module. The Component column is a foreign key to the [Component table](#). The entry in the Component column must follow the naming convention in [Naming Primary Keys in Merge Module Databases](#).

### ModuleID

The identifier for the merge module. This is a foreign key to the [ModuleSignature table](#).

### Language

The Language identifier describes the default language for the merge module. The language identifier is in decimal format, for example, U.S. English is 1033. Foreign key to the [ModuleSignature table](#).

## Remarks

Language transforms must be able to update this table if the merge module supports multiple languages.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ModuleDependency Table

The ModuleDependency table keeps a list of other merge modules that are required for this merge module to operate properly. This table enables a merge or verification tool to ensure that the necessary merge modules are in fact included in the user's installer database. The tool checks by cross referencing this table with the ModuleSignature table in the installer database.

The ModuleDependency table has the following columns.

Column	Type	Key	Nullable
ModuleID	Identifier	Y	N
ModuleLanguage	Integer	Y	N
RequiredID	Identifier	Y	N
RequiredLanguage	Integer	Y	N
RequiredVersion	Version		Y

## Columns

### ModuleID

Identifier of the merge module. This is a foreign key into the [ModuleSignature table](#).

### ModuleLanguage

Decimal language ID of the merge module in ModuleID. This is a foreign key into the [ModuleSignature table](#).

### RequiredID

Identifier of the merge module required by the merge module in ModuleID.

### RequiredLanguage

Numeric language ID of the merge module in RequiredID. The RequiredLanguage column can specify the language ID for a single language, such as 1033 for U.S. English, or specify the language ID

for a language group, such as 9 for any English. If the field contains a group language ID, any merge module with having a language code in that group satisfies the dependency. If the RequiredLanguage is set to 0, any merge module filling the other requirements satisfies the dependency.

#### RequiredVersion

Version of the merge module in RequiredID. If this field is Null, any version fills the dependency.

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE25](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ModuleExclusion Table

The ModuleExclusion table keeps a list of other merge modules that are incompatible in the same installer database. This table enables a merge or verification tool to check that conflicting merge modules are not merged in the user's installer database. The tool checks by cross-referencing this table with the ModuleSignature table in the installer database.

The ModuleExclusion table has the following columns.

Column	Type	Key	Nullable
ModuleID	Identifier	Y	N
ModuleLanguage	Integer	Y	N
ExcludedID	Identifier	Y	N
ExcludedLanguage	Integer	Y	N
ExcludedMinVersion	Version		Y
ExcludedMaxVersion	Version		Y

## Columns

### ModuleID

Identifier of the merge module. This is a foreign key into the [ModuleSignature table](#).

### ModuleLanguage

Decimal language ID of the merge module in ModuleID. This is a foreign key into the [ModuleSignature table](#).

### ExcludedID

Identifier of an excluded merge module.

### ExcludedLanguage

Numeric language ID of the merge module in ExcludedID. The ExcludedLanguage column can specify the language ID for a single language, such as 1033 for U.S. English, or specify the language ID

for a language group, such as 9 for any English. The ExcludedLanguage column can accept negative language IDs. The meaning of the value in the ExcludedLanguage column is as follows.

<b>ExcludedLanguage</b>	<b>Meaning</b>
> 0	Exclude the language IDs specified by ExcludedLanguage.
= 0	Exclude no language IDs.
< 0	Exclude all language IDs except those specified by ExcludedLanguage.

#### ExcludedMinVersion

Minimum version excluded from a range. If the ExcludedMinVersion field is Null, all versions before ExcludedMaxVersion are excluded. If both ExcludedMinVersion and ExcludedMaxVersion are Null there is no exclusion based on version.

#### ExcludedMaxVersion

Maximum version excluded from a range. If the ExcludedMaxVersion field is Null, all versions after ExcludedMinVersion are excluded. If both ExcludedMinVersion and ExcludedMaxVersion are Null there is no exclusion based on version.

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE25](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ModuleAdminUISequence Table

A merge tool evaluates the ModuleAdminUISequence table and then inserts the calculated actions into the [AdminUISequence table](#) with a correct sequence number.

The ModuleAdminUISequence table has the following columns.

Column	Type	Key	Nullable
Action	Identifier	Y	N
Sequence	Integer		Y
BaseAction	Identifier		Y
After	Integer		Y
Condition	Condition		Y

## Columns

### Action

Action to insert into sequence. Refers to one of the installer [standard actions](#), or an entry in the merge module's [CustomAction table](#) or [Dialog table](#).

If a [standard action](#) is used in the Action column of a merge module sequence table, the BaseAction and After columns of that record must be Null.

### Sequence

The sequence number of a standard action. If a custom action or dialog is entered into the Action column of this row, this field must be set to Null.

When using [standard actions](#) in merge module sequence tables, the value in the Sequence column should be the recommended action sequence number. If the sequence number in the merge module differs from that for the same action in the .msi file sequence table, the merge tool uses the sequence number from the .msi file. See the

suggested sequences in [Using a Sequence Table](#) for the recommended sequence numbers of standard actions.

### BaseAction

The BaseAction column can contain a standard action, a custom action specified in the merge module's custom action table, or a dialog specified in the module's dialog table. The BaseAction column is a key into the Action column of this table. It cannot be a foreign key into another merge table or table in the .msi file. This means that every standard action, custom action, or dialog listed in the BaseAction column must also be listed in the Action column of another record in this table.

### After

Boolean for whether Action comes before or after BaseAction.

Value	Meaning
0	Action to come before BaseAction
1	Action to come after BaseAction

### Condition

A conditional statement that indicates if the action is to be executed. Null evaluates to true.

### Remarks

If this table is present the [AdminUISequence Table](#) must also be present in the merge module.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ModuleAdminExecuteSequence Table

A merge tool evaluates the ModuleAdminExecuteSequence table and then inserts the calculated actions into the [AdminExecuteSequence table](#) with a correct sequence number.

The ModuleAdminExecuteSequence table has the following columns.

Column	Type	Key	Nullable
Action	Identifier	Y	N
Sequence	Integer		Y
BaseAction	Identifier		Y
After	Integer		Y
Condition	Condition		Y

## Columns

### Action

Action to insert into sequence. Refers to one of the installer [standard actions](#), or an entry in the merge module's [CustomAction table](#) or [Dialog table](#).

If a [standard action](#) is used in the Action column of a merge module sequence table, the BaseAction and After columns of that record must be Null.

### Sequence

The sequence number of a standard action. If a custom action or dialog is entered into the Action column of this row, this field must be set to Null.

When using [standard actions](#) in merge module sequence tables, the value in the Sequence column should be the recommended action sequence number. If the sequence number in the merge module differs from that for the same action in the .msi file sequence table,

the merge tool uses the sequence number from the .msi file. See the suggested sequences in [Using a Sequence Table](#) for the recommended sequence numbers of standard actions.

### BaseAction

The BaseAction column can contain a standard action, a custom action specified in the merge module's custom action table, or a dialog specified in the module's dialog table. The BaseAction column is a key into the Action column of this table. It cannot be a foreign key into another merge table or table in the .msi file. This means that every standard action, custom action, or dialog listed in the BaseAction column must also be listed in the Action column of another record in this table.

### After

Boolean for whether Action comes before or after BaseAction.

Value	Meaning
0	Action to come before BaseAction
1	Action to come after BaseAction

### Condition

A conditional statement that indicates if the action is to be executed. Null evaluates to true.

## Remarks

If this table is present the [AdminExecuteSequence table](#) must also be present in the merge module.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ModuleAdvtUISequence Table

Do not author this table. The Windows Installer does not use the [AdvtUISequence table](#). The AdvtUISequence table should not exist or should be left blank in the installation database. Also do not author the ModuleAdvtUISequence table.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ModuleAdvtExecuteSequence Table

A merge tool evaluates the ModuleAdvtExecuteSequence table and then inserts the calculated actions into the [AdvtExecuteSequence table](#) with a correct sequence number.

The ModuleAdvtExecuteSequence table has the following columns.

Column	Type	Key	Nullable
Action	<a href="#">Identifier</a>	Y	N
Sequence	<a href="#">Integer</a>		Y
BaseAction	<a href="#">Identifier</a>		Y
After	<a href="#">Integer</a>		Y
Condition	<a href="#">Condition</a>		Y

## Columns

### Action

Action to insert into sequence. Refers to one of the installer [standard actions](#), or an entry in the merge module's [CustomAction table](#) or [Dialog table](#).

If a [standard action](#) is used in the Action column of a merge module sequence table, the BaseAction and After columns of that record must be Null.

### Sequence

The sequence number of a standard action. If a custom action or dialog is entered into the Action column of this row, this field must be set to Null.

When using [standard actions](#) in merge module sequence tables, the value in the Sequence column should be the recommended action sequence number. If the sequence number in the merge module differs from that for the same action in the .msi file sequence table, the merge tool uses the sequence number from the .msi file. See the

suggested sequences in [Using a Sequence Table](#) for the recommended sequence numbers of standard actions.

### BaseAction

The BaseAction column can contain a standard action, a custom action specified in the merge module's custom action table, or a dialog specified in the module's dialog table. The BaseAction column is a key into the Action column of this table. It cannot be a foreign key into another merge table or table in the .msi file. This means that every standard action, custom action, or dialog listed in the BaseAction column must also be listed in the Action column of another record in this table.

### After

Boolean for whether Action comes before or after BaseAction.

Value	Meaning
0	Action to come before BaseAction
1	Action to come after BaseAction

### Condition

A conditional statement that indicates if the action is to be executed. Null evaluates to true.

## Remarks

If this table is present the [AdvtExecuteSequence table](#) must also be present in the merge module.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ModuleIgnoreTable Table

If a table in the merge module is listed in the ModuleIgnoreTable table, it is not merged into the .msi file. If the table already exists in the .msi file, it is not modified by the merge. The tables in the ModuleIgnoreTable can therefore contain data that is unneeded after the merge.

The ModuleIgnoreTable table has the following columns.

Column	Type	Key	Nullable
Table	Identifier	Y	No

## Columns

### Table

Name of the table in the merge module that is not to be merged into the .msi file.

## Remarks

To minimize the size of the .msm file, it is recommended that developers remove unused tables from modules intended for redistribution rather than listing these tables in the ModuleIgnoreTable table.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ModuleInstallUISequence Table

A merge tool evaluates the ModuleInstallUISequence table and then inserts the calculated actions into the [InstallUISequence table](#) with a correct sequence number.

The ModuleInstallUISequence table has the following columns.

Column	Type	Key	Nullable
Action	<a href="#">Identifier</a>	Y	N
Sequence	<a href="#">Integer</a>		Y
BaseAction	<a href="#">Identifier</a>		Y
After	<a href="#">Integer</a>		Y
Condition	<a href="#">Condition</a>		Y

## Columns

### Action

Action to insert into sequence. Refers to one of the installer [standard actions](#), or an entry in the merge module's [CustomAction table](#) or [Dialog table](#).

If a [standard action](#) is used in the Action column of a merge module sequence table, the BaseAction and After columns of that record must be Null.

### Sequence

The sequence number of a standard action. If a custom action or dialog is entered into the Action column of this row, this field must be set to Null.

When using [standard actions](#) in merge module sequence tables, the value in the Sequence column should be the recommended action sequence number. If the sequence number in the merge module differs from that for the same action in the .msi file sequence table, the merge tool uses the sequence number from the .msi file. See the

suggested sequences in [Using a Sequence Table](#) for the recommended sequence numbers of standard actions.

### BaseAction

The BaseAction column can contain a standard action, a custom action specified in the merge module's custom action table, or a dialog specified in the module's dialog table. The BaseAction column is a key into the Action column of this table. It cannot be a foreign key into another merge table or table in the .msi file. This means that every standard action, custom action, or dialog listed in the BaseAction column must also be listed in the Action column of another record in this table.

### After

Boolean for whether Action comes before or after BaseAction.

Value	Meaning
0	Action to come before BaseAction
1	Action to come after BaseAction

### Condition

A conditional statement that indicates if the action is to be executed. Null evaluates to true.

## Remarks

If this table is present the [InstallUISequence table](#) must also be present in the merge module.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ModuleInstallExecuteSequence Table

A merge tool evaluates the ModuleInstallExecuteSequence table and then inserts the calculated actions into the [InstallExecuteSequence table](#) with a correct sequence number.

The ModuleInstallExecuteSequence table contains the following columns.

Column	Type	Key	Nullable
Action	Identifier	Y	N
Sequence	Integer		Y
BaseAction	Identifier		Y
After	Integer		Y
Condition	Condition		Y

## Columns

### Action

Action to insert into sequence. Refers to one of the installer [standard actions](#), or an entry in the merge module's [CustomAction table](#), or [Dialog table](#).

If a [standard action](#) is used in the Action column of a merge module sequence table, the BaseAction and After columns of that record must be null.

### Sequence

The sequence number of a standard action. If a custom action or dialog is entered into the Action column of this row, this field must be set to null.

When using [standard actions](#) in merge module sequence tables, the value in the Sequence column should be the recommended action sequence number. If the sequence number in the merge module

differs from that for the same action in the .msi file sequence table, the merge tool uses the sequence number from the .msi file. See the suggested sequences in [Using a Sequence Table](#) for the recommended sequence numbers of standard actions.

### BaseAction

The BaseAction column may contain a standard action, a custom action specified in the merge module's custom action table, or a dialog specified in the module's dialog table. The BaseAction column is a key into the Action column of this table. It cannot be a foreign key into another merge table or table in the Windows Installer file. This means that every standard action, custom action, or dialog listed in the BaseAction column must also be listed in the Action column of another record in this table.

### After

Boolean for whether Action comes before or after BaseAction.

Value	Meaning
0	Action to come before BaseAction
1	Action to come after BaseAction

### Condition

A conditional statement that indicates if the action is to be executed. A null value evaluates to true.

### Remarks

If the ModuleInstallExecuteSequence table is present, the [InstallExecuteSequence table](#) must also be present in the merge module.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ModuleSubstitution Table

The ModuleSubstitution table specifies the configurable fields of a module database and provides a template for the configuration of each field. The user or merge tool may query this table to determine what configuration operations are to take place. This table is not merged into the target database.

The following tables cannot contain configurable fields and must not be listed in this table:

ModuleSubstitution table

[ModuleConfiguration table](#)

[ModuleExclusion table](#)

[ModuleSignature table](#)

The ModuleSubstitution table has the following columns.

Column	Type	Key	Nullable
Table	Identifier	Y	N
Row	Text	Y	N
Column	Identifier	Y	N
Value	Text	N	Y

## Columns

### Table

This column specifies the name of the table being modified in the module database.

### Row

This field specifies the primary keys of the target row in the table named in the Table column. Multiple primary keys are separated by semicolons. Target rows are selected for modification before any changes are made to the target table. If one record in the ModuleSubstitution table changes the primary key field of a target

row, other records in the ModuleSubstitution table are applied based on the original primary key data, not the resulting of primary key substitutions. The order of row substitution is undefined.

Values in this column are always in [CMSM special format](#). A literal semicolon (;) or equal sign (=) can be added by prefixing the character with a backslash. \. A null value for a key is signified by a null, a leading semicolon, two consecutive semicolons, or a trailing semicolon, depending on whether the null value is a sole, first, middle, or final key column value.

### Column

This field specifies the target column in the row named in the Row column. If multiple rows in the ModuleSubstitution table change different columns of the same target row, all the column substitutions are performed before the modified row is inserted into the database. The order of column substitution is undefined.

### Value

This column contains a string that provides a formatting template for the data being substituted into the target field specified by Table, Row, and Column. When a substitution string of the form [=ItemA] is encountered, the string, including the bracket characters, is replaced by the value for the configurable "ItemA." The configurable item "ItemA" is specified in the Name column of the [ModuleConfiguration table](#) and its value is provided by the merge tool. If the merge tool declines to provide a value for any item in a replacement string, the default value specified in the DefaultValue column of the ModuleConfiguration Table is substituted. If a string references an item not in the ModuleConfiguration table, the merge fails.

- This column uses [CMSM special format](#). A literal semicolon (;) or equals sign (=) can be added to the table by prefixing the character with a backslash. \.
- The Value field may contain multiple substitution strings. For example, the configuration of items "Food1" and "Food2" in the string: "[=Food1] is good, but [=Food2] is better because [=Food2] is more nutritious."
- Replacement strings must not be nested. The template "

[=AB[=CDE]]" is invalid.

- If the Value field evaluates to null, and the target field is not nullable, the merge fails and an error object of type `msmErrorBadNullSubstitution` is created and added to the error list. For details, see the error types described in [get\\_Type Function](#).
- If the Value field evaluates to the null GUID: {00000000-0000-0000-0000-000000000000}, the null GUID is replaced by the name of the feature before the row is merged into the module. For details, see [Referencing Features in Merge Modules](#).
- The template in the Value field is evaluated before being inserted into the target field. Substitution into a row is done before replacing any features.
- If the Value column evaluates to a string of only integer characters (with an optional + or -), the string is converted into an integer before being substituted into an target field of the [Integer Format Type](#). If the template evaluates to a string that does not consist only of integer characters (and an optional + or -) the result cannot be substituted into an integer target field. Attempting to insert a non-integer into an integer field causes the merge to fail and adds a `msmErrorBadSubstitutionType` error object to the error list.
- If the target column specified in the Table and Column fields is a [Text Format Type](#), and evaluation of the Value field results in an [Integer Format Type](#), a decimal representation of the number is inserted into the target text field.
- If the target field is an [Integer Format Type](#), and the Value field consists of a non-delimited list of items in [Bitfield Format](#), the value in the target field is combined using the bitwise **AND** operator with the inverse of the bitwise **OR** of all of the mask values from the items, then combined using the bitwise **OR**

operator with each of the integer or bitfield items when masked by their corresponding mask values. Essentially, this explicitly sets the bits from the properties to the provided values but leaves all other bits in the cell alone.

- If the Value field evaluates to a [Key Format Type](#), and is a key into a table that uses multiple primary keys, the item name may be followed by a semicolon and an integer value that indicates the 1-based index into the set of values that together make a primary key. If no integer is specified, the value 1 is used. For example, the [Control table](#) has two primary key columns, Dialog\_ and Control. The value of an item "Item1" that is a key into the Control table will be of the form "DialogName;ControlName", where DialogName is the value in the Dialog\_ table and ControlName is the value in the Control column. To substitute just ControlName, the substitution string [=Item1;2] should be used.

## Remarks

The ModuleSubstition table is used by [Configurable Merge Modules](#). Mergemod.dll version 2.0 or later is required to create a configurable merge module.

To ensure compatibility with versions of Mergemod.dll earlier than version 2.0, the [ModuleConfiguration table](#) and ModuleSubstitution tables should be included in the [ModuleIgnoreTable table](#) of every module.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ModuleConfiguration Table

The ModuleConfiguration table identifies the configurable attributes of the module. This table is not merged into the database.

The ModuleConfiguration table has the following columns.

Column	Type	Key	Nullable
Name	Identifier	Y	N
Format	Integer	N	N
Type	Text	N	Y
ContextData	Text	N	Y
DefaultValue	Text	N	Y
Attributes	Integer	N	Y
DisplayName	Text	N	Y
Description	Text	N	Y
HelpLocation	Text	N	Y
HelpKeyword	Text	N	Y

## Columns

### Name

This field defines the name of the configurable item. This name is referenced in the formatting template in the Value column of the [ModuleSubstitution table](#).

### Format

This column specifies the format of the data being changed.

Format	Value
Text	0
Key	1

Integer	2
Bitfield Format	3

## Type

This column specifies the type for the data being changed. This type is used to provide a context for any user-interface and is not used in the merge process. The valid values for this column depend on the value in the Format column.

## ContextData

This column specifies a semantic context for the requested data. The type is used to provide a context for any user-interface and is not used in the merge process. The valid values for this column depend on the values in the Format and Type columns.

## DefaultValue

This column specifies a default value for the item in this record if the merge tool declines to provide a value. This value must have the format, type, and context of the item. If this is a "Key" format item, the foreign key must be a valid key into the tables of the module. Null may be a valid value for this column depending on the item. For "Key" format items, this value is in [CMSM special format](#). For all other types, the value is treated literally.

Module authors must ensure that the module is valid in its default state. This ensures that versions of Mergemod.dll earlier than version 2.0 can still use the module in its default state.

## Attributes

This column is a bit field containing attributes for this configurable item. Null is equivalent to 0. All other bits in this column are reserved for future use and must be 0.

Name	Decimal	Hexadecimal	Description
mismConfigurableOptionKeyNoOrphan	1	0x00000001	This attribute that lists items in their I tool ignore

			<p>formats  <a href="#">Types</a>. It  <a href="#">ModuleS</a>  from the  The mer  row refe  column i  followin  completi</p> <p>Every ro  table wit  the  msmCor  set.</p> <p>No rows  the auth  a value.</p> <p>The mer  of the fo  satisfied</p> <p>The mer  not have  msmCor</p> <p>If the me  Default\  tool decl</p>
msmConfigurableOptionNonNullable	2	0x00000002	<p>When th  valid res  attribute  <a href="#">Format</a> <a href="#">Types</a>.</p>

DisplayName

This column provides a short description of this item that the authoring tool may use in the user interface. This column may not be localized. Set this column to null to have the module request that the authoring tool not expose this property in the UI. The tool may disregard the value in this field.

#### Description

This column provides a description of this item that the authoring tool may use in UI elements. This string may be localized by the module's language transform. This column may be null.

#### HelpLocation

This column provides either the name of a help file (without the .chm extension) or a semicolon delimited list of help namespaces. This column can be null if no help is available. This column can be null only if the HelpKeyword column is null.

#### HelpKeyword

This column provides a keyword into the help file or namespace from the HelpLocation column. The interpretation of this keyword depends on the HelpLocation column. This column may be null.

## Remarks

The ModuleConfiguration table is used by [Configurable Merge Modules](#). Mergemod.dll 2.0 or later is required to create a configurable merge module.

To ensure compatibility with older versions of Mergemod.dll, the ModuleConfiguration table and [ModuleSubstitution table](#) should be added to the [ModuleIgnoreTable table](#) of every module.

## Validation

[ICE03](#)

[ICE06](#)

[ICE25](#)

[ICE45](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge Module Summary Information Stream Reference

The following table identifies the properties for the summary information stream of the merge module. For more information, see [Summary Information Stream](#).

Property	ID	PID	Type	Description
<b>Codepage</b>	PID_CODEPAGE	1	VT_I2	Identifies the code page used to display the summary information.
<b>Title</b>	PID_TITLE	2	VT_LPSTR	"merge module".
<b>Subject</b>	PID_SUBJECT	3	VT_LPSTR	<b>ProductName</b> property.
<b>Author</b>	PID_AUTHOR	4	VT_LPSTR	<b>Manufacturer</b> property.
<b>Keywords</b>	PID_KEYWORDS	5	VT_LPSTR	MergeModule, MSI, database.
<b>Comments</b>	PID_COMMENTS	6	VT_LPSTR	Describes the merge module and its components.
<b>Template</b>	PID_TEMPLATE	7	VT_LPSTR	Platform and language versions supported by database. Required in every merge module. For more

				<p>information, see <a href="#">Template</a> for the syntax. A module that contains 64-bit components must have Intel64 or x64 set. For information, see <a href="#">Using 64-bit Merge Modules</a>.</p> <p>Lists the numeric language identifiers for all languages supported by the module. The first language in the list is the default language of the module. Specifying more than one language results in a multilanguage merge.</p>
<b>Last Saved By</b>	PID_LASTAUTHOR	8	VT_LPSTR	Specifies the platform and language of the patched database using

				the same syntax as the <a href="#">Template Summary</a> property.
<b>Revision Number</b>	PID_REVNUMBER	9	VT_LPSTR	The unique GUID for this merge module. Required in every merge module.
<b>Last Printed</b>	PID_LASTPRINTED	11	VT_FILETIME	Null.
<b>Create Time/Date</b>	PID_CREATE_DTM	12	VT_FILETIME	The time and date when the installer database was created.
<b>Last Saved Time/Date</b>	PID_LASTSAVE_DTM	13	VT_FILETIME	Initially null. Each time a user changes an installation database the value is updated to the current system time/date at the time the merge database was saved.
<b>Page Count</b>	PID_PAGECOUNT	14	VT_I4	Minimum required installer version. Stored as an

				integer in the form: Major * 100 + minor. Required in every merge module.
<b>Word Count</b>	PID_WORDCOUNT	15	VT_I4	Enter 0 (zero) for this property. Note that in a merge module, files are always inside an embedded cabinet file regardless of the value of this property. Required in every merge module.
<b>Character Count</b>	PID_CHARCOUNT	16	VT_I4	Null.
<b>Creating Application</b>	PID_APPNAME	18	VT_LPSTR	Application used to create the installer database. Typically, the value is the name of the software used to author this merge module.
<b>Security</b>	PID_SECURITY	19	VT_I4	"2".

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge Module ICE Reference

The following list provides links to each Merge Module ICE.

ICEM	Brief description of Merge Module ICE
<a href="#">ICEM01</a>	Validates that the ICE mechanism is working.
<a href="#">ICEM02</a>	Validates that all module exclusions and dependencies relate to the current module.
<a href="#">ICEM03</a>	Validates that all actions in the module are either base actions or derive from a base action.
<a href="#">ICEM04</a>	Verifies that the merge module's required empty tables are indeed empty.
<a href="#">ICEM05</a>	Checks for invalid associations with components.
<a href="#">ICEM06</a>	Checks for invalid references to features by the module.
<a href="#">ICEM07</a>	Validates that the order of files in the sequence tables and in <a href="#">MergeModule.CABinet</a> file match.
<a href="#">ICEM08</a>	Ensures that a module does not exclude something it depends on.
<a href="#">ICEM09</a>	Verifies that the merge module safely handles predefined directories.
<a href="#">ICEM10</a>	Verifies that a merge module does not contain disallowed properties in the <a href="#">Property Table</a> .
<a href="#">ICEM11</a>	Verifies that a Configurable Merge Module lists the <a href="#">ModuleConfiguration table</a> and <a href="#">ModuleSubstitution table</a> in the <a href="#">ModuleIgnoreTable table</a> of the module.
<a href="#">ICEM12</a>	Verifies that in a ModuleSequence table, standard actions have Sequence numbers and custom actions have BaseAction and After values.
<a href="#">ICEM13</a>	Verifies that publisher policy and configuration assemblies are not included in the merge module.
<a href="#">ICEM14</a>	Validates the Value Column of the <a href="#">ModuleSubstitution table</a> .

## See Also

[Validating Merge Modules](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICEM01

ICEM01 validates that the ICE mechanism is working. This ICE uses the [Time](#) property to get the time and returns either the system time or None.

Merge module ICEs are stored in a merge module .cub file called Mergemod.cub and not in the .cub file containing the ICEs used for package validation.

## Result

ICEM01 posts a message giving the time the installer called ICEM01. This ICE should never return an error.

## See Also

[Merge Module ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICEM02

ICEM02 verifies that all the module dependencies and exclusions are related to the current module.

Merge module ICEs are stored in a merge module .cub file called Mergemod.cub and not in the .cub file containing the ICEs used for package validation.

## Result

ICEM02 posts error messages if the module database attempts to specify dependencies or exclusions that do not relate to the current module.

ICEM02 posts an error message if the module database attempts to specify the current module as dependent or as excluded by itself.

## Example

ICEM02 would post the following error messages for a module containing the database entries shown below.

```
The dependency OtherModule.GUID2.1033.OtherModule.GUID3.0 in
ModuleDependency table creates a dependency for an unrelated
module can only define dependencies for itself
```

```
This module is listed as depending on itself!
```

```
The exclusion OtherModule.GUID2.1033.OtherModule.GUID3.0 in
ModuleExclusion table creates an excluded module for an unre
module. A module can only define exclusions for itself.
```

```
This module excludes itself from the target database!
```

## ModuleSignature Table

ModuleID	Language	Version
MyModule.GUID1	1033	1.0

## ModuleDependency Table

ModuleID	ModuleLanguage	RequiredID	RequiredLan
OtherModule.GUID2	1033	OtherModule.GUID3	0
MyModule.GUID1	1033	MyModule.GUID1	1033

## ModuleExclusion Table (partial)

ModuleID	ModuleLanguage	ExcludedID	ExcludedLan
OtherModule.GUID2	1033	OtherModule.GUID3	0
MyModule.GUID1	1033	MyModule.GUID1	1033

The merge module ICE posts the first error because the of the first row in the ModuleDependency table, which does not specify a required dependency for the current module specified in the ModuleSignature table. The dependencies of a module can only be specified in its own ModuleDependency table. If OtherModule.GUID3 is required by the current module, replace the first two columns of the row with the data from the ModuleSignature table. If OtherModule.GUID3 is not required by this module, delete this row.

The merge module ICE posts the second error because a module cannot specify a dependency upon itself.

The merge module ICE posts the third error because of the first row in the ModuleExclusion table, which does not specify a required exclusion for the current module specified in the ModuleSignature table. The exclusions of a module can only be specified in its own ModuleExclusion table. If the current module excludes OtherModule.GUID3, replace the first two columns of the row with the data from the ModuleSignature table. If the current module does not exclude OtherModule.GUID3, delete this row.

The merge module ICE posts the fourth error because a module cannot

specify that it exclude itself.

## **See Also**

[Merge Module ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICEM03

ICEM03 verifies that all actions in the module are either base actions or derive from a valid base action.

Merge module ICEs are stored in a merge module .cub file called Mergemod.cub and not in the .cub file containing the ICEs used for package validation.

## Result

ICEM03 posts the error messages for a module containing actions in a sequence table that is not a base action or derived from a valid base action.

## Example

ICEM03 posts the following error messages for a module containing the database entries shown below.

```
The action 'Action1' in the 'ModuleInstallExecuteSequence' table is orphaned. It is not a valid base action and does not derive from a valid base action.
The action 'Action2' in the 'ModuleInstallExecuteSequence' table is orphaned. It is not a valid base action and does not derive from a valid base action.
```

## ModuleInstallExecuteSequence Table

Action	Sequence	BaseAction	After	Condition
Action1		Action2	0	
Action2		Action1	0	

ICEM03 posts errors for these two actions because they refer to each other as base actions in the ModuleInstallExecuteSequence table. All actions in the [ModuleAdminUISequence](#), [ModuleAdminExecuteSequence](#), [ModuleAdvtUISequence](#),

[ModuleAdvtExecuteSequence](#), [ModuleInstallUISequence](#), and [ModuleInstallExecuteSequence](#) tables must either be base actions or derive their position from the combination of another action and a before and after flag.

To fix this error, determine the base actions for the two actions. Add a record for the base actions with a default sequence number. For Action1 and Action2 enter the base action names in the BaseAction column and 0 or 1 in the After column.

## **See Also**

[Merge Module ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICEM04

ICEM04 verifies that the merge module's required empty tables are empty. Failure to fix an error that ICEM04 reports can result in incorrect merging of the merge module.

## Result

ICEM04 posts an error when the merge module's required empty tables are not empty.

## Example

ICEM04 posts the following error messages for a module that contains the database entries shown.

```
An empty FeatureComponents table is required in a Merge Module.  
  
The Merge Module contains the 'ModuleInstallExecuteSequence' table.  
must therefore have an empty 'InstallExecuteSequence' table.  
  
Action 'CostInitialize' found in the AdvtExecuteSequence table.  
table must be empty in a Merge Module
```

The following table shows you a partial [AdvtExecuteSequence Table](#).

Action	Sequence
CostInitialize	1

The following list shows you the partial contents of MergeModule:

- ModuleInstallExecuteSequence
- ModuleAdvtExecuteSequence
- InstallUISequence

The following example shows you another possible error.

Feature-Component '[1].[2]' found in the FeatureComponents table. FeatureComponents table must be empty in a Merge Module.

If a merge module contains a module sequence table, it must contain the corresponding empty sequence table, whether or not the module sequence table is empty. For example, if the merge module contains the [ModuleAdminExecuteSequence Table](#), it must also contain an empty AdminExecuteSequence table.

The [FeatureComponents Table](#) is required in all merge modules and must be empty.

The following procedure shows you how to fix errors.

#### ► To fix errors

1. Add an empty FeatureComponents Table to the merge module.
2. Add an empty [InstallExecuteSequence Table](#) to the merge module.
3. Remove the 'CostInitialize' action from the AdvtExecuteSequence Table.

**Note** This table must be empty in a merge module. Actions should only appear in the ModuleAdvtExecuteSequence table.

## Tables Used During Execution

The following list identifies the tables that are used during execution:

- FeatureComponents Table
- Module\*Sequence tables and corresponding \*Sequence tables.

## See Also

[About Merge Modules](#)  
[Merge Module ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICEM05

ICEM05 verifies that the merge module is correctly associated with components in the module. Incorrectly associating a component with a module causes the component to be incorrectly associated with the target database.

Merge module ICEs are stored in a merge module .cub file called Mergemod.cub and not in the .cub file containing the ICEs used for package validation.

## Result

ICEM05 posts an error if the module database incorrectly associates components and the module.

## Example

ICEM05 posts the following error messages for a module containing the database entries shown below.

```
The component Component2.OtherModule.GUID2.1033 in the
ModuleComponents table does not belong to this Merge Module.
The component Component1.MyModule.GUID1.1033 in the ModuleCo
table is not listed in the Component table.
The component 'Component3' in the Component table is not lis
ModuleComponents table.
```

## ModuleSignature Table

ModuleID	Language	Version
MyModule.GUID1	1033	1.0

## ModuleComponents Table

Component	ModuleID	Language
Component1	MyModule.GUID1	1033

Component2	OtherModule. <i>GUID2</i>	1033
------------	---------------------------	------

### Component Table (partial)

Component	ComponentID
Component3	<i>GUID4</i>
Component2	<i>GUID5</i>

The merge module ICE reports the first error because the ModuleComponents table attempts to associate a component with another module that is not the current module specified in the ModuleSignature table. To fix this, change the ModuleID and Language columns of the ModuleComponents record for Component2 to that for the current module, MyModule.*GUID1*.

The merge module ICE reports the second error because the first record in the ModuleComponents table attempts to associate Component1 with the module. This component does not exist in the Component Table of the merge module. A module can only be associated with a component that exists in the module. To fix this, remove the record for the non-existent component.

The merge module ICE reports the third error because the module attempts to add Component3 to the target database. This component has not been associated with the module in the ModuleComponents table. To fix this error, add a record for Component3 to the ModuleComponents table.

### See Also

[Merge Module ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICEM06

ICEM06 checks for invalid direct references to features by the module.

Merge module ICEs are stored in a merge module .cub file called Mergemod.cub and not in the .cub file containing the ICEs used for package validation.

## Result

ICEM06 posts an error when the module database contains direct references to a feature. Feature information must be provided by the user of the module.

## Example

ICEM06 posts the following error messages for a module containing the database entries shown below.

```
The target of shortcut Shortcut1.GUID1 is not a property and
Modules may not directly reference features.
The row GUID2.LocalServer32.Component2 in the Class table has
that is not a null GUID. Modules may not directly reference
```

### Shortcut Table (partial)

Shortcut	Target
Shortcut1.GUID1	cmd.exe
Shortcut2.GUID1	[MyProp]
Shortcut3.GUID1	{00000000-0000-0000-0000-000000000000}

### Class Table (partial)

CLSID	Context	Component_	Feature_
GUID1	LocalServer32	Component1	{00000000-0000-0000-0000-000000000000}

<i>GUID2</i>	LocalServer32	Component2	MyFeature
--------------	---------------	------------	-----------

ICEM06 reports the first error because the first record in the Shortcut table has an entry in the Target field that is not a property or a null GUID. A module cannot reference a feature directly. Feature information must be provided by the user of the module. To fix this error, references to a feature should be replaced by a null GUID.

ICEM06 reports the second error because the second record in the Class table has an entry in the Feature field that is not a null GUID. A module cannot reference a feature directly. Feature information must be provided by the user of the module. To fix this error, references to a feature should be replaced by a null GUID.

## See Also

[Merge Module ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICEM07

ICEM07 verifies that the order of files in the sequence table matches the order of files in [MergeModule.CABinet](#).

Merge module ICEs are stored in a merge module .cub file called Mergemod.cub and not in the .cub file containing the ICEs used for package validation.

## Result

ICEM07 posts an error if the order of files in the File table does not match the order in the cabinet file.

## Example

ICEM07 would post the following error message for the example shown.

```
The file 'FileB.GUID1' appears to be out of sequence. It has  
in the CAB, but not when the file table is ordered by sequer
```

## File Table

File	Sequence
FileA.GUID1	1
FileB.GUID1	8
FileC.GUID1	52

## Embedded [MergeModule.CABinet](#)

File
FileA.GUID1
FileC.GUID1
FileD.GUID1
FileB.GUID1

Although the file sequence numbers in the file table do not have to be consecutive, and extra files can exist in the cabinet file, the relative sequence of all files in the File table must match the order in [MergeModule.CABinet](#). To fix this error, change the sequence number of FileB to come after FileC to match the file order in the CAB, or rebuild the CAB with the files in the correct order.

## **See Also**

[Merge Module ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICEM08

ICEM08 ensures that a module does not exclude another module that it depends on.

## Result

ICEM08 posts an error when a module excludes another module that it depends on.

## Example

ICEM08 posts the following error message for a module containing the database entries shown in the example.

```
Error: This module requires module ModuleB.<GUID> (1033v1.0) lists it as an exclusion.
```

## ModuleDependency Table

ModuleID	ModuleLanguage	RequiredID	RequiredLanguage	RequiredVersion
ModuleA.<GUID>	1033	ModuleB.<GUID>	1033	1.0

## ModuleExclusion Table

ModuleID	ModuleLanguage	ExcludedID	ExcludedLanguage	ExcludedVersion
ModuleA.<GUID>	1033	ModuleB.<GUID>	1033	

To fix the error, remove either the dependency or the exclusion. If ModuleB is a dependency (RequiredID) of ModuleA, you cannot exclude it (as shown in the ExcludedID column of the ModuleExclusion table). If you must exclude ModuleB, then you must remove ModuleA's

dependency on it.

## **See Also**

[Merge Module ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICEM09

ICEM09 verifies the merge module safely handles predefined directories. It does this by verifying that no component in the module installs a directory to a predefined system directory such as "ProgramFilesFolder" or "StartMenuFolder". Instead, modules should use directories with unique names (created with the merge module naming convention) and use custom actions to target the appropriate target directory. This approach prevents modules from conflicting with an existing directory structure in the final database. ICEM09 checks that the custom actions needed for this technique to work either do not exist (so that the merge tool can generate them) or exist in the correct form (so that they work as expected).

Failure to fix a warning or error reported by ICEM09 could cause problems for the clients of your merge module. Directory table rows with primary keys such as ProgramFilesFolder often exist in a database; therefore, if components in your module install directly to predefined directories such as ProgramFilesFolder, the directory entries in the module may collide with already existing rows. This condition would require the user of your module to split the source files from your module in order to match the existing source directory.

## Result

ICEM09 reports an error or warning when a module component installs a directory to a pre-defined system directory, causing a possible name conflict with the existing directory structure.

## Example

ICEM09 posts the following warnings for a module containing the database entries shown.

```
Warning: The component 'Component1.<GUID>' installs directly
directory 'ProgramFilesFolder'. It is recommended that merge
all such directories to unique names.
```

Rename the merge module directory so it does not match a Windows

Installer property and therefore is unique. Then set a property of the same name to the value of the Windows Installer directory. When directory resolution takes place, the directory has a property of the same name, so the install location of the directory is the value of the property. Files move from the distinct source location to the same target location. This process should completely remove the merge conflicts.

Warning: The 'ModuleInstallExecuteSequence' table contains a (StartMenuFolder.<GUID>) for a pre-defined directory, but the does not have sequence number '1'

If the action does not have sequence number 1, it may not merge in to the target database early enough in the sequence to work effectively.

To fix this warning, set the sequence number to 1. Note that most current merge tools (but not some older versions) will generate these custom actions at merge time, so it is not always necessary to explicitly author the actions into the merge module.

Warning: The 'CustomAction' table contains a type 51 action for a pre-defined directory, but the name is not the same as Many merge tools will generate duplicate actions."

Because the CustomAction column is the primary key of CustomAction table, some merge tools may generate duplicate actions because the pre-authored action name is different.

To fix this warning, name the action the same as the target directory. Note that most current merge tools (but not some older versions) generate these custom actions at merge time, so it is not always necessary to explicitly author the actions into the merge module.

## Directory Table

Directory	Directory_Parent	DefaultDir
ProgramFilesFolder	Directory1	A
StartMenuFolder	Directory2	B:C
AppDataFolder	Directory3	D
MyPicturesFolder	Directory4	E

## Component Table

Component	Directory
Component1.<GUID>	ProgramFilesFolder
Component2.<GUID>	StartMenuFolder
Component3.<GUID>	AppDataFolder
Component4.<GUID>	MyPicturesFolder

## CustomAction Table

CustomAction	Type	Source	Target
StartMenuFolder.<GUID>	51	StartMenuFolder.<GUID>	[StartMenuFolder]
MyAppDataFolderAction	51	AppDataFolder.<GUID>	[AppDataFolder]

## ModuleInstallExecuteSequence Table

Action	Sequence	BaseAction	After	Condition
StartMenuFolder.<GUID>	100			

## See Also

[Merge Module ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICEM10

ICEM10 verifies that a merge module contains only properties that are allowed in the [Property Table](#). The following product specific properties are not allowed in the Property Table:

- [ProductLanguage Property](#)
- [ProductCode Property](#)
- [ProductVersion Property](#)
- [ProductName Property](#)
- [Manufacturer Property](#)

## Result

ICEM10 posts an error when a merge module contains a property that is not allowed in the [Property Table](#).

## Example

ICEM10 posts the following error messages for a module that contains the database entries shown.

```
The property 'ProductLanguage' is not allowed in a merge module  
The property 'Manufacturer' is not allowed in a merge module
```

The following table shows you a partial [Property Table](#).

Property	Values
Color	Red
<a href="#">Manufacturer</a>	Microsoft
<a href="#">ProductLanguage</a>	1033

The following procedure shows you how to fix errors.

### ► To fix the errors

1. Remove the '**Manufacturer**' property from the Property Table.
2. Remove the '**ProductLanguage**' property from the Property Table.

## Table Used During Execution

The [Property Table](#) is used during execution.

## See Also

[Merge Module ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICEM11

ICEM11 verifies that a Configurable Merge Module lists the [ModuleConfiguration table](#) and [ModuleSubstitution table](#) in the [ModuleIgnoreTable table](#) of the module. This ensures that merge tools that do not recognize configurable merge modules (less than version 2.0) do not copy these tables into the target database.

This ICEM is available in the Mergemod.cub file provided in the Windows Installer 2.0 SDK and later. For details, see [Windows SDK Components for Windows Installer Developers](#).

## Result

ICEM11 posts an error if the module contains a ModuleConfiguration or ModuleSubstitution table not listed in the ModuleIgnoreTable table.

## Example

ICEM11 posts the following error messages for a module containing the database entries shown below.

```
Error The module contains a ModuleConfiguration or ModuleSubstitution table. These tables must be listed in the ModuleIgnoreTable
```

### [ModuleConfiguration](#) (partial)

Name	Format	Type	ContextData	DefaultValue
IconKey1	1	Binary	Icon	DefaultIcon

### [ModuleSubstitution](#)

Table	Row	Column	Value
Control	Dialog1;Control1	Text	[IconKey1]

## ModuleIgnoreTable

Table
ModuleConfiguration

To fix this error include both the ModuleSubstitution and ModuleConfiguration tables in the ModuleIgnoreTable table.

### Table Used During Execution

[ModuleSubstitution](#)

[ModuleConfiguration](#)

[ModuleIgnoreTable](#)

### See Also

[Merge Module ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICEM12

ICEM12 verifies that in a ModuleSequence table, standard actions have sequence numbers and custom actions have BaseAction and After values.

This ICEM is available in the Mergemod.cub file provided in the Windows Installer 2.0 SDK and later. For details, see [Windows SDK Components for Windows Installer Developers](#).

## Result

ICEM12 posts an error in the following cases:

- It finds the module contains a [standard action](#) without a sequence number.
- It finds that a standard action has values entered in the BaseAction or After fields of the [ModuleAdminUISequence table](#), [ModuleAdminExecuteSequence table](#), [ModuleAdvtExecuteSequence table](#), [ModuleInstallUISequence table](#), or [ModuleInstallExecuteSequence table](#).
- It finds the module contains a [custom action](#) without any values entered into the Sequence, BaseAction or After fields of the [ModuleAdminUISequence table](#), [ModuleAdminExecuteSequence table](#), [ModuleAdvtExecuteSequence table](#), [ModuleInstallUISequence table](#), or [ModuleInstallExecuteSequence table](#).

ICEM12 posts a warning if it finds a custom action that has a Sequence number specified, but no value in the BaseAction or After fields.

Note that all actions found in the [CustomAction table](#) are considered custom actions. All other action are considered standard actions.

## Example

ICEM12 posts the following error and warning messages for a module

that contains the database entries shown below:

Error. Custom actions should use the BaseAction and After fields in the ModuleAdminExecuteSequence table. The custom action 'Action1' does not use the BaseAction and After fields in the ModuleAdminExecuteSequence table.

Error. Custom actions should not leave the Sequence, BaseAction, and After fields empty in the ModuleAdminExecuteSequence table. The custom action 'Action1' has BaseAction, and After fields empty in the ModuleAdminExecuteSequence table.

Error. Standard actions should not use the BaseAction and After fields in the ModuleAdminExecuteSequence table. The standard action 'Action2' has a value in the BaseAction or After fields of the ModuleAdminExecuteSequence table.

Error. Standard actions must have a entry in the Sequence field in the ModuleAdminExecuteSequence table. The standard action 'Action2' does not have a Sequence field entry in the ModuleAdminExecuteSequence table.

### CustomAction

Action	Type	Source	Target
Action1	30	source1	target1
Action3	30	source3	target3

### ModuleAdminExecuteSequence

Action	Sequence	BaseAction	After	Condition
Action2		Action1	1	true
Action3				true

### ModuleInstallExecuteSequence

Action	Sequence	BaseAction	After	Condition
Action1	1			true

To fix these errors try the following:

- Remove the sequence number for the custom action Action1 and use the BaseAction and After fields instead.
- Enter values into the Sequence, BaseAction, or After fields for the custom action Action3. Leave the BaseAction and After fields empty for standard action Action2.
- Do not leave the Sequence field empty for standard action Action2.

## **See Also**

[Merge Module ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICEM13

ICEM13 verifies that the merge module does not contain publisher policy and configuration assemblies. Publisher policy and configuration assemblies should not be included in merge modules intended for redistribution because this can affect other applications on a user's computer.

This ICEM is available in the Mergemod.cub file provided in the Windows Installer 2.0 SDK and later. For details, see [Windows SDK Components for Windows Installer Developers](#).

## Result

ICEM13 posts a warning message if it finds a component specified in the Component field of the merge module's [MsiAssembly Table](#) that is a publisher policy or configuration assembly.

## Example

ICEM13 posts the following warning message if it finds a row in the MsiAssembly Table with a component '[1]' specified in the Component field that is a publisher policy or configuration assembly that has been included in the merge module.

```
This entry Component_=`[1]` in MsiAssembly Table is a Policy
```

It is possible to install publisher policy and configuration assemblies using the Windows Installer, it is not recommended that these assemblies be redistributed in merge modules.

## See Also

[Merge Module ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ICEM14

ICEM14 validates the Value Column of the [ModuleSubstitution table](#).

## Result

ICEM14 posts the following errors.

Error	Meaning
The replacement string in ModuleSubstitution.Value column in row [1].[2].[3] is not found in ModuleConfiguration table.	[1].[2].[3] refers to a <i>table.row.column</i> primary key for a row in the ModuleSubstitution table. The formatting template in the Value field of this row does not correspond to a row of configurable attributes in the <a href="#">ModuleConfiguration Table</a> .
In ModuleSubstitution table in row [1].[2].[3], a configurable item is indicated in the table '%s'. The table '%s' must not contain configurable items.	One of the following tables is listed in the Table column of the ModuleSubstitution table: <a href="#">ModuleSubstitution</a> , <a href="#">ModuleConfiguration</a> , <a href="#">ModuleExclusion</a> , or <a href="#">ModuleSignature</a> . These tables cannot contain configurable fields.
In ModuleSubstitution table in row [1].[2].[3], an empty replacement string is specified.	The formatting template in the Value field of this row does not correspond to a row of configurable attributes in the <a href="#">ModuleConfiguration Table</a> .

ICEM14 posts the following warning.

Warning	Meaning
ModuleSubstitution table exists but ModuleConfiguration table is missing	The ModuleConfiguration table is absent.

## **Table Used During Execution**

[ModuleSubstitution table](#)

[ModuleConfiguration table](#)

## **See Also**

[Merge Module ICE Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# CMSM Special Format

Certain values used with configurable merge modules require special text handling. A text string described as being in "CMSM Special Format" treats the semicolon (;) and equals (=) characters as reserved characters used by the client merge tool or Mergemod.dll.

CMSM Special format is currently used in the following locations:

- The Row column of the [ModuleSubstitution table](#).
- The Value column of the [ModuleSubstitution table](#).
- The ContextData column of the [ModuleConfiguration table](#) when Bitfield is the value in the Format column.
- The ContextData column of the [ModuleConfiguration table](#) when Text is the value in the Format column and Enum is the value in the Type column.
- The DefaultValue column of the [ModuleConfiguration table](#) when Key is the value in the Format column.
- Configurable items in the Key format used by the [ProvideTextData method](#).

To enter literal semicolons or equal characters into a value in CMSM special format, prefix the character with a backslash character ('\'). A literal backslash can be represented by two backslashes. A single character prefixed by a single backslash is translated into the single character, even if escaping the character is not required.

If a semicolon or equals character is not prefixed by a backslash yet does not have a defined behavior in the context of the value, the resulting string is undefined. For example, the DefaultValue column of the ModuleConfiguration table is in CMSM special format for all Key items because the semicolon character is the column delimiter. Although the equal character has no special meaning in this string, literal equal characters must still be escaped in this string.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Semantic Types

The following entries in the Format, Type, and ContextData columns of the [ModuleConfiguration table](#) specify the semantic type of information being substituted into the configurable item specified in the Name column of this table.

## Text Format Types

Format	Type	ContextData	Description
Text			Arbitrary text. See <a href="#">Arbitrary Text Type</a> .
Text	Enum	<A>=<a>; <B>=<b>; <C>=<c>	Value selected from a set. See <a href="#">Enum Type</a> .
Text	Formatted		Value meeting the definition of Formatted Text in the installer. See <a href="#">Formatted Type</a> .
Text	RTF		An RTF text string. See <a href="#">RTF Type</a> .
Text	Identifier		A text string conforming to a Windows Installer <a href="#">Identifier</a> .

## Integer Format Types

Format	Type	ContextData	Description
Integer			Any integer value. See <a href="#">Arbitrary Integer Type</a> .

## Key Format Types

Format	Type	ContextData	Description
Key	File	AssemblyContext	Enable users to configure foreign keys to Win32 or common language runtime assemblies. See <a href="#">File Type</a> .

Key	Binary	Bitmap	Foreign key to a Binary table row holding a bitmap for use in UI. See <a href="#">Binary Type</a> .
Key	Binary	Icon	Foreign key to a Binary table row holding an Icon for use in UI. See <a href="#">Binary Type</a> .
Key	Binary	EXE	Foreign key to a Binary table row holding a 32bit EXE. See <a href="#">Binary Type</a> .
Key	Binary	EXE64	Foreign key to a Binary table row holding a 32 or 64bit EXE. See <a href="#">Binary Type</a> .
Key	Icon	ShortcutIcon	Foreign key to an Icon table row holding an Icon for use by a shortcut. See <a href="#">Icon Type</a> .
Key	Dialog	DialogNext	Foreign key to a Dialog table row. See <a href="#">Dialog Type</a> .
Key	Dialog	DialogPrev	Foreign key to a Dialog table row. See <a href="#">Dialog Type</a> .
Key	Directory	IsolationDir	Foreign key to a Directory table row where isolated files belong. See <a href="#">Directory Type</a> .
Key	Directory	ShortcutLocation	Foreign key to a Directory table row where a shortcut should be installed. See <a href="#">Directory Type</a> .
Key	Property		Foreign key to a property row. See <a href="#">Property Type</a> .
Key	Property	Public	Foreign key to a property row. See <a href="#">Property Type</a> .
Key	Property	Private	Foreign key to a property row. See <a href="#">Property Type</a> .

## Bitfield Format Types

Format	Type	ContextData	Description
Bitfield		<mask>;<A>= <a>;<B>=b	Changes a subset of bits in a column. See <a href="#">Arbitrary Bitfield Type</a> .

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Text Format Types

The text format types of configurable data may be text strings of any length. Embedded nulls are not allowed.

The following Text Format types exist:

[Arbitrary Text](#)

[RTF](#)

[Formatted](#)

[Enum](#)

[Identifier Type](#)

Configurable items of the Text Format Type are used in non-binary database fields and in general could be replaced by any string of any length. However, particular configurable items also have semantic restrictions. For example, a configurable item that is required to be a foreign key into a specific table has additional semantic restrictions. Such semantic restrictions are not enforced by Mergemod.dll and therefore module authors should be prepared to handle any string that satisfies the specified Text Format type.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Arbitrary Text Type

The Arbitrary Text Type of [semantic type](#) is one of the [Text Format Types](#). This type consists of an arbitrary text string of any length provided by the user. The merge tool substitutes this arbitrary string into the templates specified in the Value column of the [ModuleSubstitution table](#).

To specify a configurable item of this type, module authors should enter the name of the text string into the Name column, enter "0" into the Format column, and leave blank the Type and ContextData columns of the [ModuleConfiguration table](#). The arbitrary text string may be in any language compatible with the code page of the database. For details, see [Code Page Handling \(Windows Installer\)](#). Null is a valid value for the text string unless the msmConfigItemNonNullable has been included in the Attributes field of the ModuleConfiguration table.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RTF Type

The RTF Type of [semantic type](#) is one of the [Text Format Types](#). This type consists of an arbitrary text string in the Rich Text Format (RTF) of any length provided by the user. The merge tool substitutes this string into the templates specified in the Value column of the [ModuleSubstitution table](#).

To specify a configurable item of this type, module authors should enter the name of the text string into the Name column, enter "0" into the Format column, enter "RTF" into the Type column, and leave blank the ContextData column of the [ModuleConfiguration table](#). The string may be in any language compatible with the code page of the database. See [Code Page Handling \(Windows Installer\)](#). Null is a valid value for the text string unless the msmConfigItemNonNullable has been included in the Attributes field of the ModuleConfiguration table.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Formatted Type

The Formatted Type of [semantic type](#) is one of the [Text Format Types](#). This type consists of an arbitrary text string of any length provided by the user and in the Windows Installer formatted text format. For details, see [Formatted](#). The merge tool substitutes this string into the templates specified in the Value column of the [ModuleSubstitution table](#).

To specify a configurable item of this type, module authors should enter the name of the text string into the Name column, enter "0" into the Format column, enter "Formatted" into the Type column, and leave blank the ContextData column of the [ModuleConfiguration table](#). The string may be in any language compatible with the code page of the database. For details, see [Code Page Handling \(Windows Installer\)](#). Null is a valid value for the text string unless the `msmConfigItemNonNullable` has been included in the Attributes field of the ModuleConfiguration table.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Enum Type

The Enum Type of [semantic type](#) is one of the [Text Format Types](#). This type consists of a text string chosen by the user from a set of choices. The merge tool substitutes the selected string selected into the templates specified in the Value column of the [ModuleSubstitution table](#).

To specify a configurable item of this type, module authors should enter the name of the text string into the Name column, enter "0" into the Format column, enter "Enum" into the Type column, and enter the list of possible strings in the ContextData column of the [ModuleConfiguration table](#). The list of possible strings must be provided as a list of strings delimited by semicolons. Each choice must be in the form "Name=Value". A literal semicolon can be added to the value by prefixing the semicolon with a backslash character. Null is a valid value unless the msmConfigItemNonNullable has been included in the Attributes field of the ModuleConfiguration table.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Identifier Type

The Identifier Type of [semantic type](#) is one of the [Text Format Types](#). This type consists of an text string provided by the user in the format of a Windows Installer [Identifier](#). The merge tool substitutes this string into the templates specified in the Value column of the [ModuleSubstitution table](#).

To specify a configurable item of this type, module authors should enter the name of the text string into the Name column, enter "0" into the Format column, enter "Identifier" into the Type column, and leave blank the ContextData column of the [ModuleConfiguration table](#). The string may be in any language compatible with the code page of the database. See [Code Page Handling \(Windows Installer\)](#). Null is a valid value for the text string unless the `msmConfigItemNonNullable` has been included in the Attributes field of the ModuleConfiguration table.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Integer Format Types

The Integer Format Types of configurable data may be used in either text or integer database fields. The `msmConfigItemNonNullable` attribute is implicit in this data format and does not need to be set. Null is never a valid value for this type.

The following Integer Format type exists.

## [Arbitrary Integer Type](#)

Configurable items of the Integer Format Type are used in either text or integer columns and in general could be replaced by any positive or negative integer. However, particular configurable items may also have characteristic semantic restrictions. For example, a particular configurable item required to be an integer between 0 and 100 has an additional semantic restriction. Such restrictions are not enforced by `Mergemod.dll` and therefore module authors should be prepared to handle any string that satisfies the specified Integer Format type.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Arbitrary Integer Type

The Arbitrary Integer Type of [semantic type](#) is one of the [Integer Format Types](#). This type of configurable item is an integer provided by the user. The merge tool substitutes this integer into the templates specified in the Value column of the [ModuleSubstitution table](#).

To specify a configurable item of this type, module authors should enter the name of the text string into the Name column, enter "2" into the Format column, and leave blank the Type and ContextData columns of the [ModuleConfiguration table](#). The Type and ContextData columns are reserved and must be null.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Key Format Types

The Key Format Types of configurable data may be used in text fields to provide a key into a database table. The `msmConfigItemNonNullable` attribute is implicit in this data format and does not need to be set. Null is never a valid value for this type. Responses to all Key format items must be in [CMSM Special Format](#).

The following Key Format types exist:

[Directory Type](#)

[File Type](#)

[Property Type](#)

[Dialog Type](#)

[Binary Type](#)

[Icon Type](#)

Configurable items of the Key Format Type are used in text columns to provide a database key and in general could be replaced by any text string. The individual types may have additional syntactic restrictions, but these restrictions are not enforced by `Mergemod.dll`. Particular configurable items may also have characteristic semantic restrictions. For example, a particular configurable item may be required to be a key into the [Binary table](#) to a row containing a bitmap image. Such restrictions are not enforced by `Mergemod.dll` and therefore module authors should be prepared to handle any string that satisfies the specified Key Format type. Unless otherwise specified by semantic meaning or attributes, null is a valid response.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Directory Type

The Directory Type of [semantic type](#) is one of the [Key Format Types](#), which consists of a foreign key into the [Directory table](#) provided by the user.

The merge tool must substitute a valid Windows Installer [Identifier](#) for items of this type. Mergemod.dll does not enforce this restriction and it is up to the merge tool to ensure that the user provides a valid key into the Directory table.

A configurable item of the Directory type should only modify the destination directory of the installation and not modify the source image. A configurable item of this type should therefore only modify foreign keys to the Directory table and not modify the Directory table directly.

Because the Directory\_ column of the [Component table](#) is non-nullable, null is an invalid value for a configurable item of this type even if the msmConfigItemNonNullable is not set in the Attributes column.

The Directory type may be used with two kinds of ContextData.

## IsolationDirectory ContextData

A configurable merge module may use this type to enable the user to provide a destination directory for files in the module. The merge tool substitutes the directory's identifier into the templates in the Value column of the [ModuleSubstitution table](#). To specify a configurable item of this type, module authors should enter the name of the directory into the Name column, enter "1" into the Format column, enter "Directory" into the Type column, and enter "IsolationDirectory" into the ContextData column of the [ModuleConfiguration table](#).

## ShortcutLocation ContextData

A configurable merge module may use this type to enable the user to provide a destination directory for shortcuts in the module. The merge tool substitutes the shortcut's identifier into the templates in the Value column of the [ModuleSubstitution table](#). To specify a configurable item of this type, module authors should enter the name of the directory into the Name column, enter "1" into the Format column, enter "Directory" into the Type column, and enter "ShortcutLocation" into the ContextData column

of the [ModuleConfiguration](#) table.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# File Type

The File Type of [semantic type](#) is one of the [Key Format Types](#). This type consists of a foreign key into the [File table](#) provided by the user.

The File type may be used with the following kinds of ContextData.

## **AssemblyContext ContextData**

This type may be used to enable users to configure foreign keys to Win32 or common language runtime assemblies. The merge tool must substitute a Windows Installer [Identifier](#) for items of this type into the template in the Value column of the [ModuleSubstitution table](#).

Mergemod.dll does not enforce this and it is up to the merge tool to ensure that the user provides a valid key into the File table.

Null is a valid value for this type unless the msmConfigItemNonNullable has been included in the Attributes field of the [ModuleConfiguration table](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Property Type

The Property Type of [semantic type](#) is one of the [Key Format Types](#). This type consists of a foreign key into the [Property table](#) provided by the user.

The merge tool must substitute a valid Windows Installer [Identifier](#) for items of this type. Mergemod.dll does not enforce this restriction and it is up to the merge tool to ensure that the user provides a valid key into the Property table. The primary keys of the Property table are the property names.

Null is a valid value for this type unless the `msmConfigItemNonNullable` has been included in the `Attributes` field of the [ModuleConfiguration table](#).

The Property type may be used with the following kinds of `ContextData`.

## Null ContextData

A configurable merge module may use this type to enable the user to provide a property name to a database table in the module. The merge tool substitutes the property's identifier into the templates in the `Value` column of the [ModuleSubstitution table](#). To specify a configurable item of this type, module authors should enter the name of the configurable item into the `Name` column, enter "1" into the `Format` column, enter "Property" into the `Type` column, and leave blank the `ContextData` column of the [ModuleConfiguration table](#).

## Public ContextData

A configurable merge module may use this type to enable the user to provide the name of a [public property](#) to a database table in the module. The merge tool substitutes the property's identifier into the templates in the `Value` column of the [ModuleSubstitution table](#). To specify a configurable item of this type, module authors should enter the name of the configurable item into the `Name` column, enter "1" into the `Format` column, enter "Property" into the `Type` column, and enter "Public" into the `ContextData` column of the `ModuleConfiguration` table.

## Private ContextData

A configurable merge module may use this type to enable the user to provide the name of a [private property](#) to a database table in the module. The merge tool substitutes the property's identifier into the templates in

the Value column of the [ModuleSubstitution table](#). To specify a configurable item of this type, module authors should enter the name of the configurable item into the Name column, enter "1" into the Format column, enter "Property" into the Type column, and enter "Private" into the ContextData column of the ModuleConfiguration table.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Dialog Type

The Dialog Type of [semantic type](#) is one of the [Key Format Types](#). This type consists of a foreign key into the [Dialog table](#) provided by the user.

The merge tool must substitute a valid Windows Installer [Identifier](#) for items of this type. Mergemod.dll does not enforce this restriction and it is up to the merge tool to ensure that the user provides a valid key into the Dialog table.

Null is a valid value for this type unless the `msmConfigItemNonNullable` has been included in the `Attributes` field of the [ModuleConfiguration table](#).

The Dialog type may be used with the following kinds of ContextData.

## **DialogNext ContextData**

A configurable merge module may use this type to enable the user to provide a foreign key into the Dialog Table. To specify a configurable item of this type, module authors should enter the name of the configurable item into the `Name` column, enter "1" into the `Format` column, enter "Dialog" into the `Type` column, and enter "DialogNext" into the `ContextData` column of the [ModuleConfiguration table](#).

## **DialogPrev ContextData**

A configurable merge module may use this type to enable the user to provide a foreign key into the Dialog Table. To specify a configurable item of this type, module authors should enter the name of the configurable item into the `Name` column, enter "1" into the `Format` column, enter "Dialog" into the `Type` column, and enter "DialogPrev" into the `ContextData` column of the [ModuleConfiguration table](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Binary Type

The Binary Type of [semantic type](#) is one of the [Key Format Types](#). This type consists of a key into the [Binary table](#) provided by the user.

The merge tool must substitute a valid Windows Installer [Identifier](#) for items of this type. Mergemod.dll does not enforce this restriction and it is up to the merge tool to ensure that the user provides a valid key into the Binary table.

Null is a valid value for this type unless the `msmConfigItemNonNullable` has been included in the `Attributes` field of the [ModuleConfiguration table](#).

The Binary type may be used with the following kinds of `ContextData`.

## Bitmap ContextData

A configurable merge module may use this type to enable the user to provide a foreign key to a row in the Binary Table that contains a bitmap image. Mergemod.dll does not guarantee any specific size or type of bitmap and the merge tool must ensure that the data is a valid image. To specify a configurable item of this type, module authors should enter the name of the configurable item into the `Name` column, enter "1" into the `Format` column, enter "Binary" into the `Type` column, and enter "Bitmap" into the `ContextData` column of the `ModuleConfiguration` table.

## Icon ContextData

A configurable merge module may use this type to enable the user to provide a foreign key to a row in the Binary Table that contains an icon image. Mergemod.dll does not guarantee any specific size or type of icon and the merge tool must ensure that the data is a valid image. To specify a configurable item of this type, module authors should enter the name of the configurable item into the `Name` column, enter "1" into the `Format` column, enter "Binary" into the `Type` column, and enter "Icon" into the `ContextData` column of the `ModuleConfiguration` table. This type is not appropriate for use in an advertisement table.

## EXE ContextData

A configurable merge module may use this type to enable the user to provide a foreign key to a row in the Binary Table that contains a 32-bit executable image. Mergemod.dll does not validate the data is valid and

the merge tool must ensure that the data is a valid PE file. To specify a configurable item of this type, module authors should enter the name of the configurable item into the Name column, enter "1" into the Format column, enter "Binary" into the Type column, and enter "EXE" into the ContextData column of the ModuleConfiguration table.

### **EXE64 ContextData**

A configurable merge module may use this type to enable the user to provide a foreign key to a row in the Binary Table that contains either a 32-bit or 64-bit executable image. Mergmod.dll does not validate the data is valid and the merge tool must ensure that the data is a valid PE file. To specify a configurable item of this type, module authors should enter the name of the configurable item into the Name column, enter "1" into the Format column, enter "Binary" into the Type column, and enter "EXE64" into the ContextData column of the ModuleConfiguration table.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Icon Type

The Icon Type of [semantic type](#) is one of the [Key Format Types](#). This type consists of a key into the [Icon table](#) provided by the user.

The merge tool must substitute a valid Windows Installer [Identifier](#) for items of this type. Mergemod.dll does not enforce this restriction and it is up to the merge tool to ensure that the user provides a valid key into the Icon table.

Null is a valid value for this type unless the `msmConfigItemNonNullable` has been included in the `Attributes` field of the [ModuleConfiguration table](#).

The Binary type may be used with the following kinds of ContextData.

## ShortcutIcon ContextData

A configurable merge module may use this type to enable the user to provide a foreign key to a row in the [Icon Table](#) that contains an image suitable for use as a shortcut icon. To specify a configurable item of this type, module authors should enter the name of the configurable item into the `Name` column, enter "1" into the `Format` column, enter "Icon" into the `Type` column, and enter "ShorcutIcon" into the `ContextData` column of the `ModuleConfiguration` table. This type is not appropriate for use in a user interface table. To modify a key to the Icon table in these tables see [Icon ContextData](#) under [Binary Type](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Bitfield Format Types

The Bitfield Format Types of configurable data may be used in integer database fields. The user should choose a value from a specified subset. This is by convention only and is not enforced by Mergemod.dll. The `msmConfigItemNonNullable` attribute is implicit in this data format and does not need to be set. Null is never a valid value for this type.

The following Bitfield Format type exists.

## [Arbitrary Bitfield Type](#)

Configurable items of the Bitfield Format Type are used in integer columns and in general could be replaced by any integer. The user should choose a value from a specified subset. This is by convention only, however, and is not enforced by Mergemod.dll. The author should prepare the module to handle any value.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Arbitrary Bitfield Type

The "Bitfield" type with no context requests that the user provide an integer whose value is used to set one or more bits in a bitfield. This text may be in any language compatible with the code page of the database.

The Arbitrary Bitfield Type of [semantic type](#) is one of the Bitfield Types. This type consists of an integer chosen by the user from a set of choices. The merge tool substitutes the selected integer into the templates specified in the Value column of the [ModuleSubstitution table](#). To specify a configurable item of this type, module authors should enter the name of the item into the Name column, enter "3" into the Format column, leave the Type column blank, and enter the list of possible integers in the ContextData column of the [ModuleConfiguration table](#).

The Type column is reserved and must be null. The entry in the ContextData column for all Bitfield Format types must be in the form "<mask>;<Name>=<value>;<Name>=<value>....", where <mask> is an integer value indicating the bits of interest, <Name> is a localizable display name for the choice, and <value> is a decimal integer value. The context column is in use [CMSM Special Format](#) and for all bitfield types. A literal "=" or ";" character can be entered in the <Name> field by prefixing it with a backslash ('\') character.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Configurable Component Example

In this example, the following ModuleConfiguration and ModuleSubstitution tables allows the module consumer to independently configure the Password attribute for an edit control, the checksum attribute for a file, and the compressed attribute for the same file.

## ModuleSubstitution Table

Table	Row	Column	Value
Control	Dialog1;Edit1	Attributes	[=Password]
File	File1	Attributes	[=Checksum][=Compressed]

## ModuleConfiguration Table

Name	Format	Type	ContextData
Password	Bitfield		2097152;True=2097152;False=0
Checksum	Bitfield		1024;Checksum=1024;No Checksum=0
Compressed	Bitfield		24576;Compressed=16384;Uncompressed=8192

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Windows Installer Bootstrapping

[Bootstrapping](#)

[Internet Download Bootstrapping](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Bootstrapping

Currently every installation that attempts to use the Windows Installer begins by checking whether the installer is present on the user's computer, and if it is not present, whether the user and computer are ready to install Windows Installer. A setup application [Instmsi.exe](#) is available with the Windows Installer SDK that contains all logic and functionality to install Windows Installer. However, a bootstrapping application must manage this installation.

The bootstrapping application must first check to see whether Windows Installer is currently installed. Applications can get the version of Windows Installer currently installed by using **DllGetVersion**. If Windows Installer is not currently installed, the bootstrapping application must query the operating system to determine which version of the [Instmsi.exe](#) is required. Once the installation of Windows Installer has initiated, the bootstrapping application must handle return codes from the [Instmsi.exe](#) application and handle any reboot that is incurred during the Windows Installer installation. For more information, see [Determining the Windows Installer Version](#)

The following example demonstrates how the setup application which installs Microsoft Office 2000 checks the user's system and configures the Windows Installer installation. This example is specifically written to install Office 2000 and should be used as a general reference only.

When a user inserts an Office 2000 CD-ROM into their computer, Setup.exe attempts to launch the maintenance mode, the setup application, or does nothing at all, according to the user's needs. The following section describes how the Office 2000 setup application, named Setup.exe, qualifies the user and their computer, constructs a command line and installs Windows Installer using the Msiexec.exe application.

## How Setup.exe Bootstraps the Windows Installer when Installing Office 2000

1. The user inserts an Office 2000 CD-ROM into their computer. The Windows operating system initiates Setup.exe using the /autorun switch and the Autorun.inf file. The Autorun.inf file is found at the

root of the Office 2000 CD-ROM and contains the following sections:

[Autorun]

[Office Features]

[Product Information]

[ServicePack].

The [Autorun] section contains a command line that executes the Setup.exe application, executes the icon used to display the disc, and contains information to add an "Install" option and a "Configure" option to the context menu for the CD-ROM.

The [Office Features] section contains a list of features and feature name pairs.

The [Product Information] section specifies the name and version of the application.

The [ServicePack] section allows a network administrator to set the minimum required service pack level. The network administrator can use this section to author the text of an alert message displayed if the local operating system does not have the required service pack.

The following is a sample Autorun.inf.

```
[autorun]
OPEN=setup.EXE /AUTORUN /KEY:Software\Microsoft\Office
ICON=setup.EXE,1
shell\configure=&Configure
shell\configure\command=setup.EXE
shell\install=&Install
shell\install\command=setup.EXE
[OfficeFeatures]
Feature1=ACCESSFiles
Feature2=OfficeFiles
Feature3=WORDFiles
Feature4=EXCELFiles
```

```
Feature5=PPTFiles
[ProductInformation]
DisplayName=Microsoft Office 9
Version=9.0
ProductCode={product guid}
[ServicePack]
MessageText="The operating system does not have a requ
SPLevel=3
```

2. The Setup.exe application checks for the `_MsiPromptForCD` mutex. Windows Installer creates this mutex when it prompts the user to insert the CD-ROM. The presence of the mutex indicates that Windows Installer is running an installation that has requested the Office 2000 CD-ROM. In this case, the Setup.exe application exits immediately and allows the Office 2000 installation to continue. If the mutex is absent, the Setup.exe application continues at step 3 where a registry key is evaluated to determine if Office 2000 is installed.
3. The Setup.exe application checks the presence of the Office9 registry key:  
**HKCU/Software/Microsoft/Office/9.0/Common/General/InstallP**  
If this registry key does not exist, the Setup.exe application continues at step 6 where the operating system is checked to determine if it qualifies for the installation of Office 2000.
4. If the Office 2000 registry key exists, the Setup.exe application checks the current installation state by calling **MsiQueryProductState**. A return state of `InstallState_Default` indicates that Office 2000 is already installed and the Setup.exe application continues at step 5 where the Office 2000 is checked for run from source.  
If Office 2000 is not installed, the Setup.exe application continues at step 6 where the operating system is checked to determine if it qualifies for the installation of Office 2000.

5. The Setup.exe application calls **MsiQueryFeatureState** for each of the features in the *[OfficeFeatures]* section of the Autorun.inf file. If any of these features returns INSTALLSTATE\_SOURCE, this indicates that the feature is being run from source and the Setup.exe application exits immediately.  
If none of the features returns INSTALLSTATE\_SOURCE, the Setup.exe application launches the installer application, Msiexec.exe, and presents the Windows Installer maintenance mode before exiting.
6. The Setup.exe application determines whether the operating system qualifies for an installation of Office 2000. Windows 2000 or Windows XP is required to install Office 2000. If the operating system requires a service pack update to qualify for Office 2000, the Setup.exe application displays the text specified in the Autorun.inf file. If the operating system does not qualify for Office 2000 or an upgrade of Office 2000, the Setup.exe application displays a message that prevents the user from continuing.  
If the operating system qualifies for Office 2000, the Setup.exe application continues at step 7, which determines whether Windows Installer is installed on the user's computer.
7. If Windows Installer exists on the user's machine, the Setup.exe application launches the Msiexec.exe application and passes the Office 2000 .msi file to it.  
If Windows Installer is not installed on the local machine, the Setup.exe application continues at step 8, which determines whether the operating system qualifies to have Windows Installer installed.
8. Windows Installer can be installed on Windows 2000. Installation on Windows 2000 requires administrative privileges. If the user does not have administrative privileges, the Setup.exe application

displays an error message.

9. If the local computer is eligible to have Windows Installer installed, the Setup.exe application runs the correct version of the Instmsi.exe installer application for the platform. Setup.exe may pass the "/q" command line switch to suppress the user interface and prevent the user from changing any installation configuration options.
10. The Setup.exe application loads the newly installed Msi.dll file and performs a call to the **MsiInstallProduct** function to install the user's application.

## Setup.exe Command Line Parameters

The Setup.exe application enables administrators and users to pass command line options to the Msiexec.exe application. For more information, see [Command Line Options](#). The following table lists the command options that can be used with Setup.exe.

Option	Usage	Meaning
/autorun	setup.exe /autorun	Runs the Autorun.inf described above.
/a	setup.exe /a	Initiates an administrative installation.
/j	[u m]Package or [u m]Package /t Transform List or [u m]Package /g LanguageID	Advertises a product. This option ignores any property values entered on the command line. u – Advertise to the current user. m – Advertise to all users of machine. g – Language identifier t – Applies transform to advertised package.
/I	setup.exe /I	Specifies the .msi file that

	Office9.msi /t ProgramMgmt.mst	Setup.exe is to install. If the /I option is not included, Setup.exe uses the Office9.msi file.
<i>/o&lt;property=value&gt;</i>	setup.exe /o CDKEY=111111- 1111	Sets properties in the .msi file. Setup.exe passes these it to msiexec as written.
<i>/q</i>	setup.exe /q	Set the UI level the installation. /q – no UI (/qn – for msiexec.) /qb – basic UI /qr – reduced UI.
<i>/m#</i>	setup.exe /m4	Supports multiple licenses in accordance with Select agreements. This property is used in by the License Verification custom action to write the LV certificate. The /m option must be followed by the number of unlocks allowed. The value specified by the /m option should be set as the "M" property in the Office9.msi file. If no value is specified, but the /m option is used with setup, the value of 0 should be set. The /m option is required to support Select customers using a CD or network.
<i>/settings</i>	setup.exe /settings mysettings.ini	Enables administrators to specify an .ini file containing all of the customized settings to be passed during Office 2000 setup. See the description of the .ini file below.

## Using an .ini File

Creating an initialization file may be easier than creating a long command line. Using the /settings option, the Setup.exe application reads the specified .ini file and constructs a command line to pass to the Msiexec.exe application. Only properties supported on the command line are supported in the .ini file. If a property or value is found in both the .ini file and on the command line, the command line settings override the .ini file settings.

The format of the .ini file is:

```
[msi]
[mst]
[options]
[Display]
```

The [msi] section of the .ini file specifies the path to the installation package for the installation. This corresponds to the /I option on the command line.

The [mst] section of the .ini file specifies the path to transforms used with this installation. This corresponds to the /j option on the command line. Multiple transforms are each indicated on a different line, using MST1 — MST(N). When parsed into the command line, the list in the .ini file is turned from left to right. Note that the number associated with the MST(N) title is present only to maintain unique identifiers and has no programmatic meaning.

The [options] section allows network administrators to set and override properties in the .msi or .mst files. Options set in the .ini file are added to the command line using the /o option. Each option in the option section must have a property name and a value.

The [Display] section is used to set the user interface level used during setup. This corresponds to the /q option on the command line. Valid values are — none, basic, reduced, and full.

Sample .ini file

```
[MSI]
MSI=\\sourceshare\Office2000\Office2000.msi
[MST]
MST1=\\sourceshare\Office2000\trns1.mst
MST2=\\sourceshare\Office2000\trns2.mst
```

[Options]  
PUBLICPROPERTY=your value

[Display]  
Display=None

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Internet Download Bootstrapping

With Windows Installer, a configurable bootstrap executable (Setup.exe) and configuration tool ([Msistuff.exe](#)) is included in the [Windows SDK Components for Windows Installer Developers](#). By using Msistuff.exe to configure the resources in Setup.exe, developers can easily create a web installation of a Windows Installer package.

The minimum installer version required by the bootstrap executable is Windows Installer version 2.0. Applications can get the installer version by using **DllGetVersion**. For more information see, [Determining the Windows Installer Version](#)

The bootstrap executable provided with the Windows Installer SDK does the following:

- Calls **WinVerifyTrust** to verify the digital signature of the .msi file. Windows Installer version 2.0 and later versions provides a Subject Interface Package (SIP) to enable signing of Windows Installer packages.
- If necessary upgrades the version of the Windows Installer on the machine.

The following resources of Setup.exe can be displayed or configured using [Msistuff.exe](#).

Resource ID	Description
ISETUPPROPNAME_BASEURL	The base URL location of Setup.exe. If no value is specified the location of Setup.exe defaults to removable media. Only URL-based installs are subject to a security check with <b>WinVerifyTrust</b> . The trailing forward slash on the URL is optional. This resource is optional.
ISETUPPROPNAME_DATABASE	The name of the .msi file. This is a relative path to the .msi file specified relative to the location of the Setup.exe.

	<p>program. This resource is required if resource ISETUPPROPNAME_PATCH is not specified.</p> <p>ISETUPPROPNAME_DATABASE ISETUPPROPNAME_PATCH can both be specified. Only one of the tv can be specified.</p>
<p>ISETUPPROPNAME_OPERATION</p>	<p>The type of operation to perform. Possible values are INSTALL, MINPATCH, MAJPATCH and INSTALLUPD. The INSTALL value equates to the first time installation of package. The MINPATCH value indicates that the patch specified in the PATCH property is a small or minor upgrade patch. The MAJPATCH value indicates that the patch specified in the PATCH property is a major update patch. The INSTALLUPD indicates that the installation could be a first time installation or an update. Use of INSTALLUPD has the bootstrap check for the presence of the product. If the product is present, then the recache package flag is set and a reinstall is performed; otherwise, a regular installation is performed. Note that INSTALLUPD should only be used for small and minor upgrade packages where the product code is not changing. In a major upgrade, REINSTALL is not required because the product code changes, in essence creating a brand new product. The Upgrade table authorizes actions in addition to the FindRelatedProducts, MigrateFeatureStates, and RemoveExistingProducts actions have upgrading of the original product. T</p>

	<p>the INSTALL option is sufficient. If resource is missing, the INSTALL operation is assumed if the ISETUPPROPNAME_DATABASE property is authored. Otherwise, MINPATCH is assumed if the ISETUPPROPNAME_PATCH prop is authored. This is an optional value</p>
ISETUPPROPNAME_PRODUCTNAME	<p>The name of the product. This is the name used in the banner text in the downloaded user interface. This resource is optional. If omitted, the name defaults to "the product".</p>
ISETUPPROPNAME_MINIMUM_MSI	<p>The minimum version of the Windows Installer required. If the minimum version of the Windows Installer is not installed on the computer, the application <a href="#">Instmsi.exe</a> is called to upgrade the Windows Installer. The value of this property has the same format as the PID_PAGECOUNT value. For more information, see the <a href="#">Page Count Summary</a> property. This value must be at least 200, the value that identifies Windows Installer version 2.0. This resource is required.</p>
ISETUPPROPNAME_INSTLOCATION	<p>The base URL location of the Windows Installer upgrade executables. This resource may be omitted. If this value is omitted, the default location of the upgrade executables is the location of Setup.exe.</p>
ISETUPPROPNAME_INSTMSIA	<p>The name of the ANSI version of the Windows Installer upgrade executable. This is a relative path to the ANSI version of Instmsi.exe relative to the location specified by</p>

	<p>ISETUPPROPNAME_INSTLOCAL</p> <p>This resource is required.</p>
ISETUPPROPNAME_INSTMSIW	<p>The name of the Unicode version of Windows Installer upgrade executable. This is a relative path to the Unicode version of Instmsi.exe relative to the location specified by ISETUPPROPNAME_INSTLOCAL. This resource is required.</p>
ISETUPPROPNAME_PATCH	<p>The name of the .msp file. This is a relative path to the .msp file specified relative to the location of the Setup program. This resource is required if ISETUPPROPNAME_DATABASE is not specified. ISETUPPROPNAME_DATABASE and ISETUPPROPNAME_PATCH can both be specified. Only one of the two can be specified.</p>
ISETUPPROPNAME_PROPERTIES	<p>The PROPERTY=VALUE strings. They are the PROPERTY=VALUE pairs that are included on the command line. This resource is optional.</p>

For example, the following ISETUPPROPNAME\_BASEURL and ISETUPPROPNAME\_DATABASE values would be used for the following hypothetical package locations.

Actual package location	Resource
http://www.blueyonderairlines.com/Products/Product1/product1.msi	<p>ISETUPPROPNAME_BASEURL http://www.blueyonderairlines.com/Products/Product1/product1.msi</p> <p>ISETUPPROPNAME_DATABASE http://www.blueyonderairlines.com/Products/Product1/product1.msi</p>
http://www.blueyonderairlines.com/Products/Product1/product1.msi	<p>ISETUPPROPNAME_BASEURL http://www.blueyonderairlines.com/Products/Product1/product1.msi</p>

	ISETUPPROPNAME_DATABASE ../Products
e:\product1.msi	Omit ISETUPPROPNAME_DATABASE ISETUPPROPNAME_PATCH
e:\setup\product1.msi (The currently running location of Setup.exe is e:\setup.exe.)	Omit ISETUPPROPNAME_DATABASE ISETUPPROPNAME_PATCH Setup\product1.msi

## Setup.exe Command Line Parameters

The following table lists the command options that can be used with the Setup.exe that is shipped with the [Windows SDK Components for Windows Installer Developers](#). When invoked without any command line options, Setup.exe defaults to the behavior specified via the configured resources. The command line options are mutually exclusive — only one of them can be used at a time.

Option	Usage	Meaning
/a	setup.exe /a	Initiates an administrative installation. If the ISETUPPROPNAME_DATABASE property is configured in Setup.exe, then this option can be used to perform an administrative installation of the product. This command line option is not supported if the ISETUPPROPNAME_PATCH resource is configured.
/a	setup.exe /a <full-path to an existing administrative install>	If the ISETUPPROPNAME_PATCH property is configured in Setup.exe, then this option can be used to patch an existing administrative installation of the product. This command line option is not supported if the ISETUPPROPNAME_DATABASE resource is configured.
/v	setup.exe /v <full path to	Verifies the signature on the file using WinVerifyTrust. No UI is displayed when this

	<i>a file</i> >	option is used. In this case, the result of the trust check is the return value obtained from the execution of Setup.exe.
/?		Displays a help dialog indicating the valid arguments. Note: The help dialog is also displayed if invalid arguments are supplied.

For more information, see [A URL Based Windows Installer Installation Example](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Windows Installer on 64-bit Operating Systems

On 64-bit operating systems, Windows Installer installs and manages applications consisting of 32-bit or 64-bit Windows Installer components. The following sections describe Windows Installer on 64-bit systems.

[About Windows Installer on 64-Bit Operating Systems](#)

[Using 64-Bit Windows Installer Packages](#)

For information about 64-bit merge modules, see [Using 64-bit Merge Modules](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# About Windows Installer on 64-Bit Operating Systems

Windows Installer runs as a service on computers using 32-bit or 64-bit Windows. Versions of the installer earlier than version 2.0 can install and manage 32-bit Windows Installer Packages only on 32-bit operating systems. Note that you cannot advertise or install a 64-bit application on a 32-bit operating system.

A Windows Installer package must be specified as either a 32-bit or a 64-bit package; it cannot be specified as neutral. On a computer using a 64-bit operating system, the Windows Installer service is hosted in a 64-bit process that installs both 32-bit and 64-bit packages. Windows Installer installs three types of Windows installer packages on a computer running a 64-bit operating system:

- 32-bit packages that contain only 32-bit components.
- 64-bit packages containing some 32 bit components.
- 64-bit packages containing only 64 bit components.

The follow sections describe the two types of Windows Installer packages and the new installer properties provided by Windows Installer for 64-bit packages.

[32-bit Windows Installer Packages](#)

[64-bit Windows Installer Packages](#)

## See Also

[Using 64-Bit Windows Installer Packages](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# 32-bit Windows Installer Packages

A 32-bit package consists of only 32-bit [Windows Installer components](#) and must have the value "Intel" entered in the platform field of the [Template Summary](#) Property.

For more information, see [Windows Installer on 64-bit Operating Systems](#) and [64-bit Windows Installer Packages](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# 64-bit Windows Installer Packages

A 64-bit package consists partially or entirely of 64-bit [Windows Installer](#) components. The following list identifies the requirements for every 64-bit Windows Installer package:

- The value "Intel64" must be entered in the platform field of the [Template Summary](#) property if and only if the package runs on an Intel64 processor.
- The value "x64" must be entered in the platform field of the [Template Summary](#) property if and only if the package runs on an x64 processor.
- The [Page Count Summary](#) property must be set to the integer 200 or greater, because Windows Installer 2.0 is the minimum version that is capable of installing 64-bit components.
- Each 64-bit Windows Installer component in the package must include the msidbComponentAttributes64bit bit in the Attributes column of the [Component Table](#).

For more information, see [Windows Installer on 64-bit Operating Systems](#) and [32-bit Windows Installer Packages](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using 64-Bit Windows Installer Packages

When you create [64-bit Windows Installer Packages](#) or applications that call Windows Installer to install 64-bit packages, do the following:

- Use a Windows Installer database schema of 200 or higher. Specify that version 2.0 is the minimum version of the installer required to install the package by setting the **Page Count Summary** property to the integer 200. Earlier Windows Installer versions reject attempts to install 64-bit packages.
- Indicate in the **Template Summary** property of the package summary information stream that this is a 64-bit package. Enter "Intel64" into the platform field of the **Template Summary** property if the package is to be run on an Intel64 processor. Enter "x64" if the package is to be run on a 64-bit extended processor. A package cannot be marked as supporting both Intel64 and x64 platforms, a **Template Summary** property value of "Intel64,x64" is invalid. A package cannot be marked as supporting both 32-bit and 64-bit platforms, the **Template Summary** property values of "Intel,x64" or "Intel,Intel64" are invalid.
- Identify every 64-bit component by setting the **msidbComponentAttributes64bit** in the Attributes column of the [Component](#) table.
- Use optional conditional statements that check the version of the 64-bit operating system by referencing the **VersionNT64** property. Windows Installer sets this property to the 64-bit Windows version and leaves VersionNT64 undefined if the operating system is not 64-bit Windows. For more information, see [Using Properties in Conditional Statements](#).
- Use optional conditional statements that check the numeric

processor level of the computer by referencing the [Intel64](#) or [Msix64](#) property. The Windows Installer sets these properties to the current numeric processor level of the computer and leaves the **Intel64** Property undefined if this is not an Itanium-based processor. For more information, see [Using Properties in Conditional Statements](#).

- Use the [AppSearch Table](#) and [AppSearch Action](#) to do optional searches of the registry for existing 64-bit components. To search for existing 64-bit components, include the **msidbLocatorType64bit** bit in the Type column of the [RegLocator Table](#). For more information, see [Searching for Existing Applications, Files, Registry Entries or .ini File Entries Property](#)
- Obtain the paths to system folders by referencing the [System64Folder](#) Property, [ProgramFiles64Folder](#) Property, and [CommonFiles64Folder](#) Property for the 64-bit folders and the [SystemFolder](#) Property, [ProgramFilesFolder](#) Property, and [CommonFilesFolder](#) Property for the 32-bit folders.
- Verify that the application uses the correct GUID when referencing a 64-bit component. If there are 32-bit and 64-bit versions of a specific component, these should have different component ID GUIDs.
- Determine whether any new environment variables need to be defined when installing 64-bit applications.
- If a 64-bit ODBC Driver Manager is to be installed, the component that carries it should be named ODBCDriverManager64. The ODBC Driver Manager must be authored in the installer package and a component named ODBCDriverManager64 must be included. The manager will be installed if necessary.
- Verify that the application only calls 32-bit services that run as executables. Applications should not call 32-bit services that run in DLLs.
- If the application installs coexisting 32-bit and 64-bit versions of a component, verify that the application shares .ini file information

correctly.

- Verify that the application only applies 32-bit patches to 32-bit binaries and 64-bit patches to 64-bit binaries.
- Consider future upgrade scenarios for both 32-bit and 64-bit versions and maintain upgrade codes. For more information, see [Patching and Upgrades](#).
- When using a [bootstrapping](#) application to install a 64-bit Windows Installer Package, compile the bootstrapping application as a 64-bit application.
- To disable Registry Reflection for registry keys that are affected by a particular component, set the `msidbComponentAttributesDisableRegistryReflection` bit in the `Attributes` field of the `Component` table. This may be necessary to have 32-bit and 64-bit copies of the same application coexist. If this bit is set, the Windows Installer calls the **RegDisableReflectionKey** function on each key that is being accessed by the component. This bit is available with Windows Installer version 4.0. This bit is ignored on 32-bit systems. This bit is ignored on the 64-bit versions of Windows XP and Windows 2000.

**Note** The value of the numeric registry root returned by the *lpPathBuf* parameter of the [MsiGetComponentPath](#) function distinguishes between components on 32-bit and 64-bit operating systems. For more information, see **MsiGetComponentPath** function.

## See Also

[64-Bit Custom Actions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# 64-Bit Custom Actions

On 64-bit operating systems, Windows Installer may call custom actions that have been compiled for 32-bit or 64-bit systems.

A 64-bit custom action based on [Scripts](#) must be explicitly marked as a 64-bit custom action by adding the `msidbCustomActionType64BitScript` bit to the custom actions numeric type in the Type column of the [CustomAction](#) table.

Constant	Hexadecimal	Decimal	Meaning
<code>msidbCustomActionType64BitScript</code>	0x0001000	4096	This is a 64-bit custom action written in <a href="#">Scripts</a> .

Custom actions based on [Executable files](#) or [Dynamic-Link Libraries](#) that have been compiled for a 64-bit operating systems do not require including this additional bit in the Type column of the CustomAction table.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Windows Installer and Logo Requirements

The Certified for Microsoft Windows Logo identifies products that have been verified through independent testing to comply with the Application Specification for Windows. This specification was developed by Microsoft in cooperation with customers and other developers to provide a road map for building reliable and manageable applications. Software vendors who comply with the specification qualify for the Certified for Microsoft Windows logo and then license the logo for use on their product packaging, advertising, collateral, and other marketing materials.

For more information about Windows Vista, Windows XP, and Windows Server 2008 certification for your applications, see the [Microsoft Partner Program](#).

For more information about the Windows Server 2003 and Windows 2000 certification for your applications, see <http://msdn.microsoft.com/certification>.

If you are authoring an installation package for your application, you may take advantage of the capabilities of the Microsoft Windows Installer to help satisfy several of the requirements for certification. The following table provides links to information in Windows Installer documentation pertinent for some of the certification requirements. Note that the not every requirement for certification discussed in the Application Specification is dependent upon Windows Installer.

<b>Requirement for Certification</b>	<b>See the following sections</b>
--------------------------------------	-----------------------------------

1.1 Application should perform primary functionality and maintain	If you provide an installation package, your application can use the installer service available on all 32-bit Windows platforms. Using the Installer can improve the functionality and stability of your application. See <a href="#">Resiliency Overview of Windows Installer</a> <a href="#">Roadmap to Windows Installer Documentation</a> .
--	---

stability.	
1.3 Application should support Long File Names and UNC paths	The installer supports installing to and from both UNC and drive network and always uses the long file names provided in your package except if the <b>SHORTFILENAMES</b> Property is set or if the target volume does not support long file names. See <a href="#">Filename</a> , <a href="#">File Table</a> , <a href="#">Directory Table</a> , <b>SHORTFILENAMES</b> Property.
1.5 The application should not read from or write to Win.ini, System.ini, Autoexec.bat or Config.sys.	Your installation program should add entries to the registry <i>not</i> to the Win.ini, System.ini, Autoexec.bat or Config.sys. Windows Installer supports informational keys in the registry. When your application uses the installer, these registry keys become available as installer properties. You can author the package for your application such that it is easy to check and set the values of these keys. If your application uses information that you do not want to put in the registry, create a private initialization file and place it in the directory with the application's executable files. You can easily manage the installation of a private .ini file, as well as add or remove information from existing .ini files, by using Windows Installer to install your application. See <a href="#">Modifying the Registry</a> , <a href="#">Registry Tables Group</a> , <a href="#">Registry Table</a> , <a href="#">RemoveRegistry Table</a> , <a href="#">WriteRegistryValues Action</a> , <a href="#">RemoveRegistryValues Action</a> , <a href="#">Uninstall Registry Key, Properties</a> , <a href="#">File Table</a> , <a href="#">Directory Table</a> , <a href="#">Using the Directory Table</a> , <a href="#">IniFile Table</a> , <a href="#">RemoveIniFile Table</a> , <a href="#">IniLocator Table</a> , <a href="#">WriteIniValues Action</a> , <a href="#">RemoveIniValues Action</a> , <a href="#">RemoveIniFile Table</a> .
1.7 The application should perform Windows version checking correctly.	When you prepare an installation package for your application, you must include information about the product's operating system version requirements. On initialization, the installer automatically sets certain properties to the version of the current operating system. Your setup program can use these installer properties to provide easy version checking. See <a href="#">Properties</a> , <a href="#">Version9X</a> Property, <a href="#">VersionDatabase</a> Property, <a href="#">VersionNT</a> Property, <a href="#">WindowsBuild</a> Property, <a href="#">ProductCode</a> Property,

	<p><a href="#">ProductName</a> Property, <a href="#">ProductVersion</a> Property, <a href="#">Summary Information Stream</a>, <a href="#">Revision Number Summary</a> Property.</p>
<p>2.1 The application should install using a Windows Installer package that passes validation testing.</p>	<p>You can check that your installation package passes validation testing by using a specific set of validation rules known as <a href="#">Internal Consistency Evaluators - ICEs</a>. These ICEs are contained in the file Logo.cub. To qualify for certification, an installation package must not produce any errors when validated using these rules. Warnings are acceptable, but generally should be corrected.</p> <p>The Windows Installer SDK includes the Logo.cub, Darice.cub, and Mergemod.cub files. The ICEs in the Logo.cub file are included in the Darice.cub file. If your package passes validation using Darice.cub, it will pass with Logo.cub. For more information, see <a href="#">Package Validation</a>.</p>
<p>2.2 The installation package used to install the application should follow the rules for creating components.</p>	<p>An installer component is a part of an application that is always installed or removed as one piece. There is a set of rules to help you decide how best to divide your application into components. If the components in your installation package are correctly defined, the installer can install and remove them safely. See <a href="#">Organizing Applications into Components</a>, <a href="#">Components and Features</a>, <a href="#">Changing the Component Code</a>, <a href="#">What happens if the component rules are broken?</a>, <a href="#">Working with Features and Components</a>, <a href="#">Component Table</a>.</p>
<p>2.3 The application's installation package should identify shared components.</p>	<p>If you provide an installation package that correctly organizes your application into components, and set msidbComponentAttributesSharedDllRefCount in the Component table, Windows Installer can track shared components using the reference count in the shared DLL registry of the component's key file. If you do not use the installer, then your application must keep track of its use of shared DLLs by incrementing a usage counter for the DLL in the SharedDLLs registry key.</p> <p>For more information, see <a href="#">Directory Table</a>, <a href="#">Using the Directory Table</a>, <a href="#">Component Table</a>. For the discussion of the</p>

	<p>SharedDllRefCount bit, see <a href="#">Component Table, Organizing Applications into Components</a>.</p>
<p>2.4 The application should install to the Program Files folder by default.</p>	<p>On initialization, the installer sets a property to the full path of the Program Files folder. When you prepare the installation package you can specify that the installer install your application to this folder by default. If you also include a UI in your package, the installer can provide users with an option to select another installation location. For more information, see <a href="#">Directory Table, Using the Directory Table, TARGETDIR Property, MsiSetTargetPath, Properties, ProgramFilesFolder Property</a>.</p>
<p>2.5 The application should support Add/Remove Program Files properly.</p>	<p>You can supply all the information that is needed by Add/Remove Programs in the Control Panel by setting the values of certain installer properties in your application's Windows Installer package. Setting these properties automatically writes the corresponding values into the registry. See <a href="#">Properties, Required Properties, ProductName Property, ARPINSTALLLOCATION Property, Manufacturer Property, ProductVersion Property</a>.</p>
<p>2.6 The application should ensure that Windows Installer package supports advertising.</p>	<p>Support advertising by organizing your application for advertising and by including all the information needed for advertisement in the application's Windows Installer package. For more information, see <a href="#">Advertisement, Installation-On-Demand, Publishing Products, Features, and Components, Platform Support of Advertisement, Components and Features, Suggested AdvtExecuteSequence, Suggested AdvtUISequence, Advertise Property, Advertise Action, PublishProduct Action, Feature Table, Shortcut Table, Class Table, Extension Table, Icon Table, MIME Table, ProgId Table, TypeLib Table, Verb Table, MsiConfigureFeature, MsiConfigureProduct</a>.</p>
<p>2.7 The application's installation</p>	<p>If you use the installer to install your application, you do not need to create a separate uninstaller. You can author an installation package that enables the installer to install, uninstall, and repair your application. See <a href="#">Overview of</a></p>

<p>package should ensure correct uninstall support.</p>	<p><a href="#">Windows Installer, Organizing Applications into Components, Standard Actions, About Standard Actions, Using Standard Actions, Standard Actions Reference, Installing an Application, <b>MsiInstallProduct</b>, INSTALL Action.</a></p>
<p>3.1 On Windows 2000 do not attempt to replace files that are protected by Windows File Protection.</p>	<p>Windows Installer adheres to Windows File Protection (WFP) when installing essential system files on Windows 2000. Windows Installer never attempts to install or replace a protected file. If a protected system file is modified by an unattended installation of an application, WFP restores the file to the verified file version. For more information, see <a href="#">Using Windows Installer and Windows Resource Protection</a>.</p>
<p>3.2 Authors writing new redistributable components must use side-by-side sharing techniques so their components can be installed into the application directory.</p>	<p>Authors of Windows Installer packages can specify that the installer copy the shared files (commonly shared DLLs) of an application into the application's folder rather than to a shared location. This private set of files (DLLs) are then used only by the application. See <a href="#">Isolated Components</a>.</p>
<p>3.3 For Windows 2000 and Windows 98 Second Edition, any</p>	<p>Authors of Windows Installer packages can specify that the installer copy the shared files (commonly shared DLLs) of an application into the application's folder rather than to a shared location. This private set of files (DLLs) are then used only by the application. See <a href="#">Isolated Components</a> or <a href="#">Installation of Isolated Components</a>.</p>

side-by-side DLLs that your application depends on must be installed into your application directory:

4.3  
The application should degrade gracefully if access is denied.

The objective of this requirement is to ensure that if the user is denied access to resources, the application fails in a manner that maintains a secure environment.

Windows Installer handles privileges only during installation. The application must handle privileges at run time. The installer can set certain properties on initialization to the user's privilege level or to a level specified by System Policy. You can then author your Windows Installer package such that the installer checks the user's access privileges before installation begins.

You can author the installation package so that the installer determines whether there is sufficient disk space. If you author a user interface (UI) for the application's package, it can display options to users that run out of disk space.

If the installation is unsuccessful, the installer can switch into its rollback mode and automatically restore the original state of the computer.

For more information, see [Overview of Windows Installer](#), [Resiliency](#), [Source Resiliency](#), [Privileged Property](#), [AdminUser Property](#), [System Policy](#), [InstallValidate Action](#), [File Costing](#), [DiskCost Dialog](#), [OutOfDiskSpace Property](#), [OutOfNoRbDiskSpace Property](#), [AllocateRegistrySpace Action](#), [VolumeCostList Control](#), [Rollback](#), [Installation Mechanism](#), [PROMPTROLLBACKCOST Property](#), [DISABLEROLLBACK Property](#), [EnableRollback](#)

	<a href="#">ControlEvent</a> .
4.5 The application should adhere to system-level Group Policy settings.	Windows Installer can follow policies pertaining to installation. For more information, see <a href="#">System Policy</a> .
7.1 Applications should continue to function after upgrade to Microsoft Windows 2000 Professional without reinstall	The installer has a component attribute that can facilitate preparation of your application for migration to Windows 2000. For more information, see <a href="#">Using Transitive Components</a> .

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Windows Installer Examples

The following sections present an example of authoring installation package for an application. An example of a minimal user interface for this sample is provided in the [Windows SDK Components for Windows Installer Developers](#) as the file Uisample.msi. If you have the SDK, you have access to all the tools and data necessary to reproduce this sample installation package and user interface.

For more information, see the following detailed examples:

- [An Installation Example](#)
- [An Upgrade Example](#)
- [A Customization Transform Example](#)
- [A Small Update Patching Example](#)
- [A Database and Patch Example](#)
- [A Localization Example](#)
- [A MUI Shortcut Example](#)
- [A URL Based Windows Installer Installation Example](#)
- [Windows Installer Scripting Examples](#)
- [Single Package Authoring Example](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# An Installation Example

This example illustrates how to create a simple Windows Installer package that installs an application. The sample installs Notepad, a text editor included with Windows, and several text files describing events and admissions at the imaginary Red Park Arena.

The sample has the following specifications:

- The application is provided to users as a self-installing Windows Installer package that installs all the required files, shortcuts, and registry information.
- The installation package may present a UI wizard to the user during setup to collect user information.
- During setup, users have the option of selecting individual features to be installed to run-locally, to run-from-source, or to not be installed.
- One of the features can be presented to users as an install-on-demand feature.
- The same package uninstalls the application and removes all the application files and registry information from the user's computer.
- The package is prepared to receive a major upgrade that includes changing its product code.

To reproduce the example, you need a software tool capable of creating and editing a blank Windows Installer database. Several package creation tools are available from independent software vendors. A Windows Installer database editor called Orca is provided in the [Windows SDK Components for Windows Installer Developers](#).

To complete the example, follow these steps:

[Planning the Installation](#)

[Importing a Blank Database](#)

[Specifying Directory Structure](#)

[Specifying Components](#)  
[Specifying Files and File Attributes](#)  
[Specifying Source Media](#)  
[Specifying Features](#)  
[Specifying Feature-Component Relationships](#)  
[Adding Registry Information](#)  
[Specifying Shortcuts](#)  
[Specifying Properties](#)  
[Importing the InstallExecuteSequence](#)  
[Importing the InstallUISequence](#)  
[Importing the AdminExecuteSequence](#)  
[Importing the AdminUISequence](#)  
[Importing the AdvtExecuteSequence](#)  
[Adding Summary Information](#)  
[Importing the User Interface](#)  
[Validating an Installation Database](#)  
[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Planning the Installation

When the installation of an existing application is moved to Windows Installer from another setup technology, the setup developer may start authoring a Windows Installer package using the source and target file images of the existing installation. A detailed plan of how the files and other resources are organized at the source and target is also a good starting point for developing a package for a new application.

The sample installation package takes the following files that are stored at the source location for the application and installs them to the target on the user's computer.

<b>File</b>	<b>Description</b>	<b>Path to source</b>
Redpark.exe	Text editor executable file.	C:\Sample\notepad\Redpark.exe
Readme.txt	An informational file.	C:\Sample\notepad\Readme.txt
Help.txt	Help manual	C:\Sample\notepad\Help.txt
Baseball.txt	Baseball game schedule for year 2000.	C:\Sample\notepad\Events\Baseball.txt
Football.txt	Football game schedule for year 2000.	C:\Sample\notepad\Events\Football.txt
Dance.txt	Dance performances for year 2000.	C:\Sample\notepad\Events\Dance.txt
Concert.txt	Music performances	C:\Sample\notepad\Events\Concert.txt

	for year 2000.	
January.txt	Admissions in January of year 2000.	C:\Sample\notepad\Gate\January.txt
NewYears.txt	Admissions on New Years Day of year 2000.	C:\Sample\notepad\Gate\Holidays\NewYears.txt

The sample writes the following values in the user's registry under **HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\notepad Sample**.

<b>Name</b>	<b>Value</b>
IfCharSet	0
IfClipPrecision	2
IfFaceName	FixedSys
IfItalic	0
IfOrientation	0
IfOutPrecision	1
fSavePageSetting	0
IfPitchAndFamily	49
iPointSize	120
IfQuality	2
IfStrikeOut	0
IfWeight	400
fWrap	0

The sample installs the following shortcuts. One of these shortcuts may

be selected during setup as an advertised shortcut so that the user can install-on-demand the Baseball feature.

<b>Name</b>	<b>Shortcut location</b>	<b>Shortcut target</b>
sNotepad	[ProgramFilesFolder]\Red_Park\Menu\	[ProgramFilesFolder]\Red
sReadme	[ProgramFilesFolder]\Red_Park\Menu\	[ProgramFilesFolder]\Red
sHelp	[ProgramFilesFolder]\Red_Park\Menu\	[ProgramFilesFolder]\San
sBaseball	[ProgramFilesFolder]\Red_Park\Menu\	[ProgramFilesFolder]\Red
sFootball	[ProgramFilesFolder]\Red_Park\Menu\	[ProgramFilesFolder]\Red
sDance	[ProgramFilesFolder]\Red_Park\Menu\	[ProgramFilesFolder]\Red
sConcert	[ProgramFilesFolder]\Red_Park\Menu\	[ProgramFilesFolder]\Red
sJanuary	[ProgramFilesFolder]\Red_Park\Menu\	[ProgramFilesFolder]\Red
sNewYears	[ProgramFilesFolder]\Red_Park\Menu\	[ProgramFilesFolder]\Red

To reproduce the sample, begin by creating the source directory structure given in the first table. You can make a copy of your system's Notepad.exe file and then rename this copy Redpark.exe. Use the Notepad editor to create the remaining text files. The directory structure of the target, the registry values, and the shortcuts are added by authoring the installation database.

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Importing a Blank Database

To reproduce the sample you need to copy, or create with a software tool, a Windows Installer database file. A totally blank installation database, Schema.msi, is provided with the [Windows SDK Components for Windows Installer Developers](#). The SDK also provides a partially blank database, uisample.msi, that contains the suggested sequence tables and data required for a simple user interface. Make a copy of uisample.msi and move it into the same directory containing the Notepad folder you created in [Planning the Installation](#). Rename the file MNP2000.msi. The installation database file and the source files must both be located at the root of the same directory. For example:

```
C:\Sample\MNP2000.msi
```

```
C:\Sample\Notepad\
```

If you do not use Uisample.msi, then you must obtain a blank database, such as Schema.msi, and enter information for the installation using a database editing tool such as Orca, which is provided with the SDK, or another database editor.

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Specifying Directory Structure

The installer keeps information about the installation directory structure in the [Directory Table](#). See the [Core Tables Group](#). In this section you add directory structure information for the Notepad sample to the empty database you created in [Importing a Blank Database](#). Use the database editor Orca that is provided with the SDK, or another editor, to open the Directory Table in MNP2000.msi. Use the editor to enter the following data into the blank Directory table.

## Directory Table

Directory	Directory_Parent	DefaultDir
<b>TARGETDIR</b>		SourceDir
<b>ProgramFilesFolder</b>	<b>TARGETDIR</b>	.
ARTSDIR	NOTEPADDIR	Arts:Events
HOLDIR	MONDIR	.:Holidays
MENUDIR	NOTEPADDIR	Menu
MONDIR	NOTEPADDIR	Gate
NOTEPADDIR	<b>ProgramFilesFolder</b>	Red_Park:Notepad
SPORTDIR	NOTEPADDIR	Sports:Events

Entering this data into the Directory table specifies the source and target directory structures. See the [Directory Table](#) and [Using the Directory Table](#) topics. Note that the **TARGETDIR** property must be the name of one root in the Directory table of every installation.

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Specifying Components

The Windows Installer installs and removes blocks of resources referred to as [Windows Installer Components](#). For more information, see [Core Tables Group](#) and [Components and Features](#).

In this section you add information about the components used by the Notepad example to the [Component Table](#) you created in [Importing a Blank Database](#). For more information, see [Organizing Applications into Components](#) and [Defining Installer Components](#).

The Notepad sample uses eight components to control resources.

<b>Component Resources</b>	
Baseball	Baseball.txt, sBaseball
Concert	Concert.txt, sConcert
Dance	Dance.txt, sDance
Football	Football.txt, sFootball
Help	Help.txt, sHelp
January	January.txt, sJanuary
NewYears	NewYears.txt, sNewYears
Notepad	Redpark.exe, Readme.txt, sReadme, sNotepad, <b>HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Notepad Sample</b>

Every component should be identified with a unique component ID [GUID](#). If you are reproducing the sample, do not reuse the same component ID GUIDs in the following table. Instead use a utility such as Guidgen.exe to generate new GUIDs for your components.

Be sure that you use a GUID string consistent with the Windows Installer GUID data type. For more information, see [Changing the Component Code](#) and [What happens if the component rules are broken?](#)

Use Orca or another database editor to enter the following data into the blank [Component Table](#) of MNP2000.msi. Do not reuse the GUIDs

shown below in the ComponentId column in your sample.

Component	ComponentId	Directory_	Attributes	Condition	Keyph
Baseball	{F54ABAC0-33F2-11D3-91D7-00C04FD70856}	SPORTDIR	2		Baseb
Concert	{76FA7A80-33F6-11D3-91D8-00C04FD70856}	ARTSDIR	2		Conce
Dance	{CCF834A1-33F8-11D3-91D8-00C04FD70856}	ARTSDIR	2		Dance
Football	{CCF834A0-33F8-11D3-91D8-00C04FD70856}	SPORTDIR	2		Footb
Help	{AD10EB50-33C1-11D3-91D6-00C04FD70856}	NOTEPADDIR	2		Help.t
January	{CF0BC690-33C9-11D3-91D6-00C04FD70856}	MONDIR	2		Janua
NewYears	{A42D9140-33D8-11D3-91D6-00C04FD70856}	HOLDIR	2		NewY
Notepad	{19BED232-30AB-11D3-91D3-00C04FD70856}	NOTEPADDIR	2		Redpa

The source and target directories for each component is specified by the value entered into the Directory\_ column. The installer resolves the location of this directory using the information in the Directory table. The installer uses the key path files specified in the KeyPath column to detect each component. The remote execution attributes are set in the sample so that the components can be run-from-source or run-locally.

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Specifying Files and File Attributes

The installation and removal of each file is determined by the [Windows Installer Component](#) that controls the file. Once the grouping of resources into components has been specified, file attribute information can be added to the installation database. In this section, you add file information to the installation database for the Notepad sample. See the [File Tables Group](#).

The files in the Notepad sample are uncompressed. See [Compressed and Uncompressed Sources](#) for information on how to add cabinet files to packages. This sample does not contain file versioning information. For more information about file versioning, see [File Versioning Rules](#) and [Default File Versioning](#).

Use your database editor to open MNP2000.msi and enter the following data into the empty File table.

## File Table

File	Component_	FileName	FileSize	Version	Language	Attr
Baseball.txt	Baseball	Baseball.txt	1000			0
Concert.txt	Concert	Concert.txt	1000			0
Dance.txt	Dance	Dance.txt	1000			0
Football.txt	Football	Football.txt	1000			0
Help.txt	Help	Help.txt	1000			0
January.txt	January	January.txt	1000			0
NewYears.txt	NewYears	NewYears.txt	1000			0
Redpark.exe	Notepad	Redpark.exe	45328			0
Readme.txt	Notepad	Readme.txt	1000			0

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Specifying Source Media

The Media table describes the set of disks that make up the source media for the installation. See the [File Tables Group](#). In this section, you add information about the source media for the Notepad sample.

Use your database editor to open MNP2000.msi and enter the following data into the empty Media table.

## Media Table

DiskId	LastSequence	DiskPrompt	Cabinet	VolumeLabel	Source
1	1				

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Specifying Features

The Microsoft Installer enables users to install and remove blocks of application functionality that is referred to as [Windows Installer Features](#). In this section you add information to the installation database about the features that are available for the Notepad sample. For more information, see [Core Tables Group](#) and [Components and Features](#).

The Notepad sample installs features in a hierarchy of parent and child features. In the following list, child features are indented relative to their parent feature. The features should display in this order in the [SelectionTree Control](#) of the user interface (UI).

## Notepad

- Readme
- Help

## Gate

- January
  - NewYears

## Sport

- Baseball
- Football

## Arts

- Concert
- Dance

Use a database editor to open MNP2000.msi and enter the following data into the empty [Feature Table](#).

Feature	Feature_Parent	Title	Description	Display	Level	Directory
Arts		Arts	Arts events	20	3	NOTEPA1

			at Red Park.			
Baseball	Sport	Baseball	Baseball Games	17	3	SPORTDI
Concert	Arts	Concert	Concert events at Red Park	21	3	ARTSDIR
Dance	Arts	Dance	Dance events at Red Park	23	3	ARTSDIR
Football	Sport	Football	Football Games	19	3	SPORTDI
Gate		Gate	Red Park's Admissions	6	3	NOTEPAI
Help	Notepad	Help	Help file.	5	3	NOTEPAI
January	Gate	January	January Admissions	10	3	MONDIR
NewYears	January	New Years Day	New Years Day Admissions	11	3	HOLDIR
Notepad		Notepad	Notepad Editor	1	3	NOTEPAI
Readme	Notepad	Readme	Readme File	3	3	NOTEPAI
Sport		Sport Events	Sport Events at Red Park	14	3	NOTEPAI

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Specifying Feature-Component Relationships

Each [Windows Installer Feature](#) uses one or more [Windows Installer Components](#), and features may share components. The [FeatureComponents table](#) defines the relationship between features and components. See the [Core Tables Group](#) and [Components and Features](#) in the Windows Installer overview. In this section you add information to the FeatureComponents table of the Notepad sample.

Use your database editor to open MNP2000.msi and enter the following data into the empty FeatureComponents table.

## FeatureComponents Table

Feature_	Component_
Baseball	Baseball
Concert	Concert
Dance	Dance
Football	Football
Help	Help
January	January
NewYears	NewYears
Notepad	Notepad
Readme	Notepad

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Adding Registry Information

The [Registry table](#), and related tables, of the installation database holds the registry information the application needs written in the system registry. See the [Registry Tables Group](#). In this section you add the information that is to be registered on the user's computer by the Notepad sample.

Use your database editor to open MNP2000.msi and enter the following data into the empty Registry table.

## Registry Table

Registry	Root	Key	Name	V
CharSet	2	SOFTWARE\Microsoft\Notepad Sample	lfCharSet	#
ClipPrecision	2	SOFTWARE\Microsoft\Notepad Sample	lfClipPrecision	#
Escapement	2	SOFTWARE\Microsoft\Notepad Sample	lfFaceName	F
Italic	2	SOFTWARE\Microsoft\Notepad Sample	lfItalic	#
Orientation	2	SOFTWARE\Microsoft\Notepad Sample	lfOrientation	#
OutPrecision	2	SOFTWARE\Microsoft\Notepad Sample	lfOutPrecision	#
PageSettings	2	SOFTWARE\Microsoft\Notepad Sample	fSavePageSetting	#
PitchAndFamily	2	SOFTWARE\Microsoft\Notepad Sample	lfPitchAndFamily	#
PointSize	2	SOFTWARE\Microsoft\Notepad Sample	iPointSize	#
Quality	2	SOFTWARE\Microsoft\Notepad Sample	lfQuality	#

StrikeOut	2	SOFTWARE\Microsoft\notepad Sample	lfStrikeOut	#
Weight	2	SOFTWARE\Microsoft\notepad Sample	lfWeight	#
Wrap	2	SOFTWARE\Microsoft\notepad Sample	fWrap	#

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Specifying Shortcuts

The [Shortcut table](#) and related tables of the installation database hold information needed to install shortcuts. See the [Program Information Tables Group](#) and [Editing Installer Shortcuts](#).

In this section you add information that specifies advertised and non-advertised shortcuts for the Notepad sample.

Use your database editor to open MNP2000.msi and enter the following data into the Shortcut table.

## Shortcut Table

Shortcut	Directory_	Name	Component_	Target	Argur
sBaseball	MENUDIR	Baseball.txt	Baseball	Baseball	
sConcert	MENUDIR	Concert.txt	Concert	[#Concert.txt]	
sDance	MENUDIR	Dance.txt	Dance	[#Dance.txt]	
sFootball	MENUDIR	Football.txt	Football	[#Football.txt]	
sHelp	MENUDIR	Help.txt	Help	[#Help.txt]	
sJanuary	MENUDIR	January.txt	January	[#January.txt]	
sNewYears	MENUDIR	NewYears.txt	NewYears	[#NewYears.txt]	
sNotepad	MENUDIR	Redpark.exe	Notepad	[#Redpark.exe]	
sReadme	MENUDIR	Readme.txt	Notepad	[#Readme.txt]	

The sample installation needs to enable installation of an advertised shortcut for the Baseball feature. This requires specifying a key to the Icon table in the Icon\_ column of the Shortcut table. For the purposes of this example you may copy the icon for the Orca database editor provided with the Windows Installer SDK. Export the [Icon table](#) from Orca.msi and then merge this table into the MNP2000.msi database using Orca or another merge tool. Orca also creates a directory named Icon in the directory containing MNP2000.msi, and adds the icon binary data file orca\_icon.exe.ibd. See the Data column in Icon table. The completed Icon table should look as follows when viewed in Orca.

## Icon Table

Name	Data
orca_icon.exe	[Binary Data]

[Continue](#)

## See Also

[Editing Installer Shortcuts](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Specifying Properties

Windows Installer properties are global variables the installer uses during an installation. See the sections under [Properties](#). If in the section [Importing a Blank Database](#) you used uisample.msi from the Windows Installer SDK, the [Property table](#) in your copy of MNP2000.msi already contains many properties that are used by the user interface. In this section you add additional information to the Property table specific to the installation of the Notepad sample. See also the [Program Information Tables Group](#).

There are five properties that are required in every installation package, and these must be updated for the Notepad sample in the [Property table](#) of MNP2000.msi:

- [ProductCode](#)
- [ProductLanguage](#)
- [Manufacturer](#)
- [ProductVersion](#)
- [ProductName](#)

Although not required by all installation packages, applications that may in the future receive an upgrade should also have an [UpgradeCode](#) property. See [Preparing an Application for Future Major Upgrades](#).

Use your database editor to open MNP2000.msi and enter the following data into the Property table. The list provides links to the reference topics for built-in installer properties. The property names that are not links are author-defined properties. Many of the properties imported from uisample.msi are for the sample user interface. The later section [User Interface for the Installation Sample](#) discusses the user interface.

## Property Table

Property	Value
<a href="#">ARPHELPLINK</a>	<a href="http://www.microsoft.com/management">http://www.microsoft.com/management</a>
BannerBitmap	bannrbmp

ButtonText_Back	< &Back
ButtonText_Browse	Br&owse
ButtonText_Cancel	Cancel
ButtonText_Exit	&Exit
ButtonText_Finish	&Finish
ButtonText_Ignore	&Ignore
ButtonText_Install	&Install
ButtonText_Next	&Next >
ButtonText_No	&No
ButtonText_OK	OK
ButtonText_Remove	&Remove
ButtonText_Reset	&Reset
ButtonText_Resume	&Resume
ButtonText_Retry	&Retry
ButtonText_Return	&Return
ButtonText_Yes	&Yes
CompleteSetupIcon	completi
ComponentDownload	ftp://anonymous@microsoft.com/components/
CustomSetupIcon	custicon
<b>DefaultUIFont</b>	DlgFont8
DialogBitmap	dlgbmp
DlgTitleFont	{&DlgFontBold8}
ErrorDialog	ErrorDlg
ExclamationIcon	exclamic
False	0
Iagree	No

InfoIcon	info
InstallerIcon	insticon
<b>INSTALLLEVEL</b>	3
InstallMode	Typical
<b>Manufacturer</b>	Microsoft
<b>PIDTemplate</b>	12345<###-%%%%%%%%>@@@@@
<b>ProductCode</b>	{18A9233C-0B34-4127-A966-C257386270BC}
<b>ProductID</b>	none
<b>ProductLanguage</b>	1033
<b>ProductName</b>	MNP2000
<b>ProductVersion</b>	01.40.0000
Progress1	Installing
Progress2	installs
<b>PROMPTROLLBACKCOST</b>	P
RemoveIcon	removico
RepairIcon	repairic
Setup	Setup
True	1
<b>UpgradeCode</b>	{908E378A-9551-4772-BF1D-5CFAF6FD9CB4}
Wizard	Setup Wizard

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Importing the InstallExecuteSequence

The [InstallExecuteSequence table](#) lists the actions that are executed when the installer executes the top-level `INSTALL` action. See [Installation Procedure Tables Group, Using a Sequence Table](#), and the [Sequence Table Detailed Example](#).

If in the section [Importing a Blank Database](#) you used `uisample.msi` from the Windows Installer SDK, the sequence tables in your copy of `MNP2000.msi` already contains the suggested action sequences described in [Using a Sequence Table](#). No changes to these sequences are necessary to author the Notepad installation package.

Use your database editor to open `MNP2000.msi` and enter the following data into the `InstallExecuteSequence` table.

## InstallExecuteSequence Table

Action	Condition	Sequence
AllocateRegistrySpace	NOT Installed	1550
AppSearch		400
BindImage		4300
CCPSearch	NOT Installed	500
CostFinalize		1000
CostInitialize		800
CreateFolders		3700
CreateShortcuts		4500
DeleteServices	VersionNT	2000
DuplicateFiles		4210
FileCost		900
FindRelatedProducts		200
InstallFiles		4000

InstallFinalize		6600
InstallInitialize		1500
InstallODBC		5400
InstallServices	VersionNT	5800
InstallValidate		1400
LaunchConditions		100
MigrateFeatureStates		1200
MoveFiles		3800
PatchFiles		4090
ProcessComponents		1600
PublishComponents		6200
PublishFeatures		6300
PublishProduct		6400
RegisterClassInfo		4600
RegisterComPlus		5700
RegisterExtensionInfo		4700
RegisterFonts		5300
RegisterMIMEInfo		4900
RegisterProduct		6100
RegisterProgIdInfo		4800
RegisterTypeLibraries		5500
RegisterUser		6000
RemoveDuplicateFiles		3400
RemoveEnvironmentStrings		3300
RemoveExistingProducts		6700
RemoveFiles		3500

RemoveFolders		3600
RemoveIniValues		3100
RemoveODBC		2400
RemoveRegistryValues		2600
RemoveShortcuts		3200
RMCCPSearch	NOT Installed	600
SelfRegModules		5600
SelfUnregModules		2200
SetODBCFolders		1100
StartServices	VersionNT	5900
StopServices	VersionNT	1900
UnpublishComponents		1700
UnpublishFeatures		1800
UnregisterClassInfo		2700
UnregisterComPlus		2100
UnregisterExtensionInfo		2800
UnregisterFonts		2500
UnregisterMIMEInfo		3000
UnregisterProgIdInfo		2900
UnregisterTypeLibraries		2300
ValidateProductID		700
WriteEnvironmentStrings		5200
WriteIniValues		5100
WriteRegistryValues		5000

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Importing the InstallUISequence

The [InstallUISequence table](#) lists actions that are executed when the top-level [INSTALL action](#) is executed and the internal user interface level is set to full UI or reduced UI. The installer skips the actions in this table if the user interface level is set to basic UI or to no UI. See [User Interface](#) and [User Interface Levels](#). See [Installation Procedure Tables Group](#), [Using a Sequence Table](#), and the [Sequence Table Detailed Example](#).

If in the section [Importing a Blank Database](#) you used uisample.msi from the Windows Installer SDK, the sequence tables in your copy of MNP2000.msi already contains the suggested action sequences described in [Using a Sequence Table](#). No changes to these sequences should be necessary to author the Notepad installation package.

Use your database editor to open MNP2000.msi and enter the following data into the InstallUISequence table.

## InstallUISequence Table

Action	Condition	Sequence
AppSearch		400
CCPSearch	NOT Installed	500
CostFinalize		1000
CostInitialize		800
ExecuteAction		1300
ExitDlg		-1
FatalErrorDlg		-3
FileCost		900
LaunchConditions		100
MaintenanceWelcomeDlg	Installed AND NOT RESUME AND NOT Preselected	1250
PrepareDlg		140
ProgressDlg		1280

ResumeDlg	Installed AND (RESUME OR Preselected)	1240
RMCCPSearch	NOT Installed	600
UserExitDlg		-2
WelcomeDlg	NOT Installed	1230
MigrateFeatureStates		1200
FindRelatedProducts		200

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Importing the AdminExecuteSequence

The [AdminExecuteSequence table](#) lists actions that the installer executes when it calls the top-level **ADMIN** action. See [Installation Procedure Tables Group](#), [Using a Sequence Table](#), and the [Sequence Table Detailed Example](#).

If in the section [Importing a Blank Database](#) you used uisample.msi from the Windows Installer SDK, the sequence tables in your copy of MNP2000.msi already contains the suggested action sequences described in [Using a Sequence Table](#). No changes to these sequences should be necessary to author the Notepad sample installation package.

Use your database editor to open MNP2000.msi and enter the following data into the AdminExecuteSequence table.

## AdminExecuteSequence Table

Action	Condition	Sequence
CostFinalize		1000
CostInitialize		800
FileCost		900
InstallAdminPackage		3900
InstallFiles		4000
InstallFinalize		6600
InstallInitialize		1500
InstallValidate		1400

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Importing the AdminUISequence

The [AdminUISequence table](#) lists actions that the installer calls when it executes the top-level [ADMIN action](#) and the internal user interface level is set to full UI or reduced UI. The installer skips the actions in this table if the user interface level is set to basic UI or no UI. See [User Interface](#) and [User Interface Levels](#). See [Installation Procedure Tables Group, Using a Sequence Table](#), and the [Sequence Table Detailed Example](#).

If in the section [Importing a Blank Database](#) you used uisample.msi from the Windows Installer SDK, the sequence tables in your copy of MNP2000.msi already contains the suggested action sequences described in [Using a Sequence Table](#). No changes to these sequences should be necessary to install the Notepad sample.

Use your database editor to open MNP2000.msi and enter the following data into the AdminExecuteSequence table.

## AdminUISequence Table

Action	Condition	Sequence
AdminWelcomeDlg		1230
CostFinalize		1000
CostInitialize		800
ExecuteAction		1300
ExitDlg		-1
FatalErrorDlg		-3
FileCost		900
PrepareDlg		140
ProgressDlg		1280
UserExitDlg		-2

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Importing the AdvtExecuteSequence

The [AdvtExecuteSequence table](#) lists actions the installer calls when it executes the top-level [ADVERTISE action](#). See [Installation Procedure Tables Group](#), [Using a Sequence Table](#), and the [Sequence Table Detailed Example](#).

If in the section [Importing a Blank Database](#) you used uisample.msi from the Windows Installer SDK, the sequence tables in your copy of MNP2000.msi already contains the suggested action sequences described in [Using a Sequence table](#). No changes to these sequences should be necessary to author the Notepad sample installation package.

Use your database editor to open MNP2000.msi and enter the following data into the AdvtExecuteSequence table.

## AdvtExecuteSequence Table

Action	Condition	Sequence
CostFinalize		1000
CostInitialize		800
CreateShortcuts		4500
InstallFinalize		6600
InstallInitialize		1500
InstallValidate		1400
PublishComponents		6200
PublishFeatures		6300
PublishProduct		6400
RegisterClassInfo		4600
RegisterExtensionInfo		4700
RegisterMIMEInfo		4900
RegisterProgIdInfo		4800

The [AdvtUISequence table](#) is not used by the installer. This table should not exist or be left empty in the installation database.

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Adding Summary Information

The following summary information properties must be defined in every installation package, using a software tool to access the [Istream](#) interface of the [Summary Information Stream](#). For example, you can use the tool Msiinfo.exe provided with the Windows Installer SDK to set these properties. If these properties are not set, the package will not pass [Package Validation](#).

Summary information property	Data	Notes
<b>Template</b> (Platform and Language)	;1033	Platform and language used by the database. If the platform specification is missing in the <b>Template</b> Summary property value, the installer assumes the Intel architecture. The <b>ProductLanguage</b> property from the database is typically used for this summary property. The sample's Language ID indicates that the package uses U.S. English.
<b>Revision Number</b> (Package Code)	{4966AEC4-3C59-4B07-9B98-1B6A7103C0D3}	This is the package code <b>GUID</b> that uniquely identifies the sample package. If you reproduce this sample, use a utility such as GUIDGEN to generate a different GUID for your package. The results of GUIDGEN contain lowercase characters, note that you must change all lowercase characters to uppercase for a valid package code. See <a href="#">Package Codes</a> .
<b>Page Count</b> (Minimum Installer Version)	200	For a minimum Windows Installer 2.0, this property should be set to the integer 200.
<b>Word Count</b>	0	The global source type for the package is long file names and uncompressed. See

(Type of Source)	<a href="#">Compressed and Uncompressed Sources</a> and the description of the Attributes column of the <a href="#">File table</a> for more information.
------------------	--

The remaining summary information stream properties are not required, but should be set for the MNP2000.msi sample.

<b>Summary information property</b>	<b>Data</b>	<b>Notes</b>
<b>Title</b>	Installation Database	Informs users that this database is for an installation rather than a transform or a patch.
<b>Subject</b>	MNP2000	File browsers can display this as the product to be installed with this database.
<b>Keywords</b>	Installer, MSI, Database	File browsers that are capable of keyword searching can search for these words.
<b>Author</b>	Microsoft Corporation	Name of the product's manufacturer.
<b>Comments</b>	This installer database contains the logic and data required to install Notepad.	Informs users about the purpose of this database.
<b>Creating Application</b>	Orca	Application used to create the installation database. The sample specifies the Orca database editor as an example.
<b>Security</b>	0	The sample database is unrestricted read-write.

You do not need to set the [Last Saved By](#), [Last Saved Time/Date](#), [Create Time/Date](#), [Last Printed](#), [Character Count](#), and [Codepage](#)

summary properties to complete this sample database. The sample relies upon the database editing tool to set and update these optional properties.

For example, to use MsiInfo to add the summary information to the sample, change to the directory containing the database and use the following command line. Do not reuse the example package ID shown below.

```
Msiinfo.exe MNP2000.msi -T "Installation Database" -J Subject -A "Microsoft Corporation" -K "Installer, MSI, Database" -O "This installer database contains the logic and data required to install Notepad." -P ;1033 -V {4966AEC4-3C59-4B07-9B98-1B6A7103C0D3} -G 200 -W 0 -N Orca -U 0
```

For more details about summary information, see [About the Summary Information Stream](#), [Using the Summary Information Stream](#), and [Summary Information Stream Reference](#).

See the [Summary Information Stream Property Set](#) for a complete list of all the summary information properties and [Summary Property Descriptions](#) for their description.

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Importing the User Interface

In addition to information discussed in previous sections, uisample.msi also contains data for a sample user interface. If you merged uisample.msi into MNP2000.msi in the section [Importing a Blank Database](#), then this information is present in MNP2000.msi as well. The information for the sample user interface is in the following tables.

- [ActionText table](#)
- [Binary table](#)
- [Control table](#)
- [ControlEvents table](#)
- [Dialog table](#)
- [Error table](#)
- [EventMapping table](#)
- [RadioButton table](#)
- [TextStyle table](#)
- [UIText table](#)

The database editor Orca provided with the installer includes a dialog preview option that you can use to preview the dialogs of the user interface that is specified by the data in the above tables.

The sample installation package MNP2000.msi is now ready for package validation. Always run validation on a new package before attempting to install the package for the first time. This is discussed in [Validating the Installation Sample](#).

If you do not want to include the user interface in the sample package, omit or remove the all information in the tables shown above except for the [TextStyle table](#) (which is needed to define the [DefaultUIFont](#) property). You should also remove user interface properties from the [Property Table](#). An example Property table for the Notepad sample, without a UI, is shown below. Do not reuse the GUIDs shown in the table if you copy this example.

## [Property Table](#)

<b>Property</b>	<b>Value</b>
<b>DefaultUIFont</b>	DlgFont8
<b>INSTALLLEVEL</b>	3
<b>LIMITUI</b>	1
<b>Manufacturer</b>	Microsoft
<b>ProductCode</b>	{18A9233C-0B34-4127-A966-C257386270BC}
<b>ProductLanguage</b>	1033
<b>ProductName</b>	MNP2000
<b>ProductVersion</b>	01.40.0000
<b>UpgradeCode</b>	{908E378A-9551-4772-BF1D-5CFAF6FD9CB4}

A package without a user interface can be installed from the command line or from a program. To install a package from the command line use the methods described in [Command Line Options](#). To install a package from a program use the methods described in [Using Installer Functions](#). Always run validation on a new package before attempting to install a new package for the first time.

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Validating an Installation Database

Authors of installation packages should always run validation on their packages before attempting to install the package for the first time and rerun validation whenever making any changes to the package.

Validation scans the database for errors that may appear valid individually but that cause incorrect behavior in the context of the whole database. Attempting to install a package that fails validation can damage the user's system. See the sections [Package Validation](#) and [Internal Consistency Evaluators - ICEs](#).

You can validate the sample package using Orca or Msival2.exe (both are provided with the Windows installer SDK). To view the help for Msival2.exe change directories and enter on the command line.

```
Msival2 -?
```

The .cub file darice.cub is also provided with the SDK. This file contains the ICE custom actions needed by Msival2.exe to perform validation. To validate the MNP2000.msi enter

```
msival2 MNP2000.msi Darice.cub
```

For a description of the error and warning messages returned by validation see the [ICE Reference](#). Correct all the errors in the package and rerun validation as necessary until the package passes validation without errors.

Once the package passes validation, you can install the sample package by clicking on the MNP2000.msi icon or from the command line using the [Command Line Options](#).

This completes the sample installation.

## Next example

[An Upgrade Example](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# An Upgrade Example

The following sections present an example of authoring an upgrade package for the application described in [An Installation Example](#). An example of a minimal user interface for this sample is provided in the [Windows SDK Components for Windows Installer Developers](#) as the file Uisample.msi. If you have the SDK, you have access to all the tools and data necessary to reproduce the sample installation package, user interface, and sample upgrade package.

This example illustrates how to create a Windows Installer package that upgrades the hypothetical product MNP2000 to a new product called MNP2001. The example upgrade package applies a major upgrade to the product which requires changing the product code. For more information about major upgrades, see the topic on [Major Upgrades](#) in the [Patching and Upgrades](#) section.

The sample upgrade package has the following specifications:

- To qualify to receive this upgrade to MNP2001, a user must have previously installed the 1.0 to 1.4 (inclusive) versions of English language MNP2000 using Windows Installer.
- When a user attempts to install the upgrade package, the upgrade functionality of Windows Installer searches the user's computer for any products that qualify for the upgrade.
- Windows Installer migrates all the of the original product's feature settings to the upgraded product.
- The installer removes all obsolete features from the user's computer.
- The installer installs all new features belonging to the upgrade.
- An uninstall of the upgrade package removes the product from the user's computer, and does not restore the earlier version of the product.
- The sample upgrade updates shortcuts to new files and features.

[Planning a Major Upgrade](#)

Importing Original Installation Database  
Updating Directory Structure for an Upgrade  
Updating Files and File Attributes for an Upgrade  
Updating Components for an Upgrade  
Updating Features for an Upgrade  
Updating Shortcuts for an Upgrade  
Updating Upgrade Table for an Upgrade  
Updating Properties for an Upgrade  
Updating Sequence Tables for an Upgrade  
Updating Summary Information for an Upgrade  
Validating an Installation Upgrade

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Planning a Major Upgrade

If Windows Installer is used for the installation and setup of an application, later upgrades of that application can be handled by installing an upgrade package. Setup developers may choose to author an upgrade package by modifying the original installation package. This approach is illustrated by the following upgrade example.

The installation of the original product, MNP2000, followed by the installation of the upgrade package provides the user with the following files required by the product MNP2001.

File	Description	Path to source
Redpark.exe	Text editor executable file. Unchanged from previous products.	C:\Sample\notepad\Redpark.exe
Readme.txt	An information file. Unchanged from previous products.	C:\Sample\notepad\Readme.txt
Help.txt	Help manual. Unchanged from previous products.	C:\Sample\notepad\Help.txt
Baseba01.txt	Baseball game schedule for year 2001.	C:\Sample\notepad\Events\Baseba01.txt

Footba01.txt	Football game schedule for year 2001.	C:\Sample\notepad\Events\Footba01.txt
Basket01.txt	Basketball game schedule for year 2001.	C:\Sample\notepad\Events\Basket01.txt
Dance01.txt	Dance performances for year 2001.	C:\Sample\notepad\Events\Dance01.txt
Concert01.txt	Music performances for year 2001.	C:\Sample\notepad\Events\Concer01.txt
Opera01.txt	Opera performances for year 2001.	C:\Sample\notepad\Events\Opera01.txt
Januar01.txt	Admissions in January of year 2001.	C:\Sample\notepad\Gate\Januar01.txt
NewYea01.txt	Admissions on New Years Day of year 2001.	C:\Sample\notepad\Gate\Holidays\NewYea01.txt
Memori01.txt	Admissions on Memorial Day of year 2001.	C:\Sample\notepad\Gate\Holidays\Memori01.txt

Installation of the upgrade package removes all the features installed with the original product that are not being used by the upgraded product.

For example, when upgrading from MNP2000, installation of the upgrade removes the following files from the user's computer:

- Baseball.txt
- Football.txt
- Dance.txt
- Concert.txt
- January.txt
- NewYears.txt

Installation of the upgrade package writes the following values in the user's registry under:

**HKEY\_LOCAL\_MACHINE\SOFTWARE\Microsoft\Notepad Sample**

<b>Name</b>	<b>Value</b>
IfCharSet	0
IfClipPrecision	2
IfFaceName	FixedSys
IfItalic	0
IfOrientation	0
IfOutPrecision	1
fSavePageSetting	0
IfPitchAndFamily	49
iPointSize	120
IfQuality	2
IfStrikeOut	0
IfWeight	400
fWrap	0

The upgrade updates old shortcuts to the following shortcuts. One of these shortcuts may be selected during setup as an advertised shortcut so that the user can install-on-demand the Baseball feature.

<b>Name</b>	<b>Shortcut location</b>	<b>Shortcut target</b>
sNotepad	[ProgramFilesFolder]\Red_Park\Menu\	[ProgramFilesFolder]\Red
sReadme	[ProgramFilesFolder]\Red_Park\Menu\	[ProgramFilesFolder]\Red
sHelp	[ProgramFilesFolder]\Red_Park\Menu\	[ProgramFilesFolder]\San
sBaseball	[ProgramFilesFolder]\Red_Park\Menu\	[ProgramFilesFolder]\Red
sFootball	[ProgramFilesFolder]\Red_Park\Menu\	[ProgramFilesFolder]\Red
sBasketball	[ProgramFilesFolder]\Red_Park\Menu\	[ProgramFilesFolder]\Red
sDance	[ProgramFilesFolder]\Red_Park\Menu\	[ProgramFilesFolder]\Red
sConcert	[ProgramFilesFolder]\Red_Park\Menu\	[ProgramFilesFolder]\Red
sOpera	[ProgramFilesFolder]\Red_Park\Menu\	[ProgramFilesFolder]\Red
sJanuary	[ProgramFilesFolder]\Red_Park\Menu\	[ProgramFilesFolder]\Red
sNewYears	[ProgramFilesFolder]\Red_Park\Menu\	[ProgramFilesFolder]\Red
sMemorial	[ProgramFilesFolder]\Red_Park\Menu\	[ProgramFilesFolder]\Red

When a user uninstalls the upgrade package, Windows Installer completely removes all versions of the product from the user's computer. The user is not left with any parts of MNP2000 or MNP2001.

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Importing Original Installation Database

Begin authoring the upgrade package by copying the installation package and source directory tree of the original product. Instructions for authoring the installation package for MNP2000 are given in [An Installation Example](#). You should copy the contents of the following folders:

C:\Sample\MNP2000.msi

C:\Sample\notepad\

C:\Sample\Binary\

C:\Sample\Icon\

Update the contents of the Notepad folder so that they match the source described in [Planning a Major Upgrade](#). Remove all obsolete source files, such as Baseball.txt, and replace with the updated files, such as Baseba01.txt. Add the additional new files provided by the upgrade, such as Opera01.txt.

Rename MNP2000.msi to MNP2001.msi. In subsequent steps you will use a table editor to modify this database into the .msi file for the upgrade. Database tables that are not explicitly modified in the subsequent sections are identical to the tables in the original product's database, MNP2000.msi.

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Updating Directory Structure for an Upgrade

Use the database editor Orca that is provided with the SDK, or another editor, to open the [Directory table](#) in MNP2001.msi. The directory structure of the sample upgrade is the same as the directory structure of the original product and no changes are required to the original Directory table described in [Specifying Directory Structure](#).

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Updating Files and File Attributes for an Upgrade

Because the upgrade updates the files used by the application, you must modify the [File table](#) of the database. Use the database editor Orca that is provided with the SDK, or another editor to open MNP2001.msi and enter the following data into the [File table](#).

## File Table

File	Component_	FileName	FileSize	Version	Language	At
Baseba01.txt	Baseball	Baseba01.txt	1000			0
Concer01.txt	Concert	Concer01.txt	1000			0
Dance01.txt	Dance	Dance01.txt	1000			0
Opera01.txt	Opera	Opera01.txt	1000			0
Footba01.txt	Football	Footba01.txt	1000			0
Basket01.txt	Basketball	Basket01.txt	1000			0
Help.txt	Help	Help.txt	1000			0
Januar01.txt	January	Januar01.txt	1000			0
NewYea01.txt	NewYears	NewYea01.txt	1000			0
Memori01.txt	Memorial	Memori01.txt	1000			0
Redpark.exe	Notepad	Redpark.exe	45328			0
Readme.txt	Notepad	Readme.txt	1000			0

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Updating Components for an Upgrade

By design, users of the fictitious MNP2000 product should never use upgraded files such as Baseba01.txt. Therefore the updated files are by definition incompatible with the original product and Windows Installer components, such as Baseball, containing these files must be assigned new component codes. New files, such as Opera01.txt, are introduced as a part of a new component with a unique component code. Because the original product and upgrade use the same Notepad component, the component code of this component is unchanged. For more information about when component code must be changed, see [Changing the Component Code](#).

Use Orca or another database editor to enter the following data into the [Component table](#) of MNP2001.msi. Do not reuse the GUIDs shown below in the ComponentId column in your sample.

## Component Table

Component	ComponentId	Directory_	Attributes	Condition	Key
Baseball	{2951190A-6AF8-4D7F-AA16-D256405C277A}	SPORTDIR	2		Bas
Basketball	{E1AAB6B0-FEC6-4F18-B765-3B05A81CEACB}	SPORTDIR	2		Bas
Concert	{C28C5064-AA84-4431-AC69-FC1321DF18AF}	ARTSDIR	2		Cor
Dance	{1AC2B14D-D5F4-4642-9F7A-EE81BF59B3E2}	ARTSDIR	2		Dar
Opera	{C2DABF7E-1EF6-458D-84B1-	ARTSDIR	2		Ope

	AAC1127CED26}				
Football	{92AA30F4-7AC5-4DFA-801E-988CF3DAA4DC}	SPORTDIR	2		Foo
Help	{AD10EB50-33C1-11D3-91D6-00C04FD70856}	NOTEPADDIR	2		Hel
January	{E90CD0E6-ED8D-4F88-B000-27BD2B482C6C}	MONDIR	2		Janu
NewYears	{1EEF8C53-F7C0-405C-8FE3-2B0FE54B0114}	HOLDIR	2		New
Memorial	{BA81ACF7-4D43-424F-93B0-8845A2DF1C02}	HOLDIR	2		Mei
Notepad	{19BED232-30AB-11D3-91D3-00C04FD70856}	NOTEPADDIR	2		Red

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Updating Features for an Upgrade

The sample upgrade package adds three new features to the original product:

- Basketball, a new child feature of the Sport feature.
- Opera, a new child feature of the Arts feature.
- Memorial, a new child feature of the Gate feature.

Use your database editor to open MNP2001.msi and enter the following data into the [Feature table](#).

## Feature Table

Feature	Feature_Parent	Title	Description	Display	Level	Direct
Arts		Arts	Arts events at Red Park.	18	3	NOTE
Baseball	Sport	Baseball	Baseball Games	17	3	SPOR
Concert	Arts	Concert	Concert events at Red Park	19	3	ARTS
Dance	Arts	Dance	Dance events at Red Park	21	3	ARTS
Football	Sport	Football	Football Games	13	3	SPOR
Gate		Gate	Red Park's Admissions	6	3	NOTE
Help	Notepad	Help	Help file.	5	3	NOTE
January	Gate	January	January Admissions	7	3	MONI
NewYears	January	New Years Day	New Years Day Admissions	9	3	HOLI

Notepad		Notepad	Notepad Editor	1	3	NOTE
Readme	Notepad	Readme	Readme File	3	3	NOTE
Sport		Sport Events	Sport Events at Red Park	12	3	NOTE
Basketball	Sport	Basketball	Basketball Games	15	3	SPOR
Opera	Arts	Opera	Opera Performances	23	3	ARTS
Memorial	Gate	Memorial Day	Memorial Day Admissions	11	3	HOLI

Use your database editor to open MNP2001.msi and enter the following data into the empty [FeatureComponents Table](#).

<b>Feature_</b>	<b>Component_</b>
Baseball	Baseball
Concert	Concert
Dance	Dance
Football	Football
Help	Help
January	January
NewYears	NewYears
Notepad	Notepad
Readme	Notepad
Basketball	Basketball
Opera	Opera
Memorial	Memorial

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Updating Shortcuts for an Upgrade

The sample upgrade package updates shortcuts to the new files and features.

Use your database editor to open MNP2001.msi and enter the following data into the [Shortcut table](#).

Shortcut	Directory_	Name	Component_	Target	Arg
sBaseball	MENUDIR	Baseba01.txt	Baseball	Baseball	
sConcert	MENUDIR	Concer01.txt	Concert	[#Concer01.txt]	
sDancell	MENUDIR	Dance01.txt	Dance	[#Dance01.txt]	
sFootball	MENUDIR	Footba01.txt	Football	[#Footba01.txt]	
sHelp	MENUDIR	Help.txt	Help	[#Help.txt]	
sJanuary	MENUDIR	Januar01.txt	January	[#Januar01.txt]	
sNewYears	MENUDIR	NewYea01.txt	NewYears	[#NewYea01.txt]	
sNotepad	MENUDIR	Redpark.exe	Notepad	[#Redpark.exe]	
sReadme	MENUDIR	Readme.txt	Notepad	[#Readme.txt]	
sMemorial	MENUDIR	Memori01.txt	Memorial	[#Memori01.txt]	
sBasketball	MENUDIR	Basketba01.txt	Basketball	[#Basketba01.txt]	
sOpera	MENUDIR	Opera01.txt	Opera	[#Opera01.txt]	

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Updating Upgrade Table for an Upgrade

To apply a major upgrade using Windows Installer, the original product installation package must specify an **UpgradeCode** Property, described in [Preparing an Application for Future Major Upgrades](#), and the upgrade package must have an [Upgrade table](#).

For more information about major upgrades, see [Major Upgrades in Patching and Upgrades](#).

The installation package of MNP2000.msi was assigned an **UpgradeCode** property, as described in the section [Specifying Properties](#).

Windows Installer applies the upgrade if the user has already installed the 1.0 to 1.4 versions (inclusive) of English language MNP2000. Windows Installer migrates all of the original product's feature settings to the upgraded product. The installer removes the files belonging to the original products not being used by the product's upgrade.

If your copy of MNP2001.msi does not include an [Upgrade table](#), use Orca, or another table editor, to import an empty Upgrade table into the database from Schema.msi. The SDK provides a copy of Schema.msi. Use your database editor to open MNP2001.msi and enter the following data into the empty Upgrade table.

<b>UpgradeCode</b>	<b>VersionMin</b>	<b>VersionMax</b>	<b>Language</b>	<b>Attributes</b>	<b>Remov</b>
{908E378A-9551-4772-BF1D-5CFAF6FD9CB4}	01.00.0000	01.40.0000	1033	769	

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Updating Properties for an Upgrade

Because the upgrade changes the name of the .msi file and changes the component code of some components, the product code of the upgrade must be changed from that of the original product. For a description of the cases where an upgrade is required to change the **ProductCode** property, see [Changing the Product Code](#). An upgrade that changes the ProductCode is referred to as a [Major Upgrade](#).

The upgrade package's **ProductName** property, **ProductVersion** property, **ProductLanguage** property, and **UpgradeCode** property may be changed, or left unchanged, from the original product. Based upon the values of these properties, Windows Installer may determine whether to apply future upgrade packages to the current upgrade.

The property specified in the ActionProperty column of the [Upgrade table](#) must be added to the **SecureCustomProperties** Property.

Use your database editor to open MNP2001.msi and enter the following data into the [Property table](#). The list provides links to the reference topics for built-in installer properties. The property names that are not links are author-defined properties. Many of the properties were imported from Uisample.msi that comes with the SDK. For details, see [Importing the User Interface](#).

## Property Table

Property	Value
<a href="#">ARPHELPLINK</a>	http://www.microsoft.com/management
BannerBitmap	bannrbmp
ButtonText_Back	< &Back
ButtonText_Browse	Br&owse
ButtonText_Cancel	Cancel
ButtonText_Exit	&Exit
ButtonText_Finish	&Finish
ButtonText_Ignore	&Ignore

ButtonText_Install	&Install
ButtonText_Next	&Next >
ButtonText_No	&No
ButtonText_OK	OK
ButtonText_Remove	&Remove
ButtonText_Reset	&Reset
ButtonText_Resume	&Resume
ButtonText_Retry	&Retry
ButtonText_Return	&Return
ButtonText_Yes	&Yes
CompleteSetupIcon	completi
ComponentDownload	ftp://anonymous@microsoft.com/components/
CustomSetupIcon	custicon
<b>DefaultUIFont</b>	DlgFont8
DialogBitmap	dlgbmp
DlgTitleFont	{&DlgFontBold8}
ErrorDialog	ErrorDlg
ExclamationIcon	exclamic
False	0
Iagree	No
InfoIcon	info
InstallerIcon	insticon
<b>INSTALLLEVEL</b>	3
InstallMode	Typical
<b>Manufacturer</b>	Microsoft
<b>PIDTemplate</b>	12345<###-%%%%%%%%%>@@@@@

<b>ProductCode</b>	{34CF587C-1D8F-4DD5-ADFE-440F4B593987}
<b>ProductID</b>	none
<b>ProductLanguage</b>	1033
<b>ProductName</b>	MNP2001
<b>ProductVersion</b>	01.50.0000
Progress1	Installing
Progress2	installs
<b>PROMPTROLLBACKCOST</b>	P
RemoveIcon	removico
RepairIcon	repairic
Setup	Setup
True	1
<b>UpgradeCode</b>	{908E378A-9551-4772-BF1D-5CFAF6FD9CB4}
Wizard	Setup Wizard
SecureCustomProperties	OLDAPPFOUND

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Updating Sequence Tables for an Upgrade

The sequence tables of the sample upgrade package are the same as in the original product. See the following sections:

[Importing the InstallExecuteSequence](#)

[Importing the InstallUISequence](#)

[Importing the AdminExecuteSequence](#)

[Importing the AdminUISequence](#)

[Importing the AdvtExecuteSequence](#)

[Updating Summary Information for an Upgrade](#)

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Updating Summary Information for an Upgrade

The package code of the upgrade package must be changed from that of the original product. The package code is stored in the **Revision Number Summary** Property. For details, see [Package Codes](#). Use Msiinfo.exe, or another editor, to change the summary information of MNP2001.msi as described in [Adding Summary Information](#).

Summary information property	Data	Notes
<b>Template</b> (Platform and Language)	;1033	Platform and language used by the database. If the platform specification is missing in the <b>Template</b> Summary property value, the installer assumes the Intel architecture. The <b>ProductLanguage</b> property from the database is typically used for this summary property. The sample's Language ID indicates that the package uses U.S. English.
<b>Revision Number</b> (Package Code)	{A0EC5348-1D02-4FF6-93F5-61BD7AC1772E}	This is the package code <b>GUID</b> that uniquely identifies the sample package. If you reproduce this sample, use a utility such as GUIDGEN to generate a different GUID for your package. The results of GUIDGEN contain lowercase characters, note that you must change all lowercase characters to uppercase for a valid package code.
<b>Page Count</b> (Minimum Installer Version)	200	For a minimum Windows Installer version 2.0, this property should be set to the integer 200.
<b>Word</b>	0	The global source type for the package is

<b>Count</b> (Type of Source)		long file names and uncompressed. For more information, see <a href="#">Compressed and Uncompressed Sources</a> and the description of the Attributes column of the <a href="#">File table</a> .
<b>Title</b>	Installation Database	Informs users that this database is for an installation rather than a transform or a patch.
<b>Subject</b>	MNP2001	File browsers can display this as the product to be installed with this database.
<b>Keywords</b>	Installer, MSI, Database	File browsers that are capable of keyword searching can search for these words.
<b>Author</b>	Microsoft Corporation	Name of the product's manufacturer.
<b>Comments</b>	This installer database contains the logic and data required to install Notepad.	Informs users about the purpose of this database.
<b>Creating Application</b>	Orca	Application used to create the installation database. The sample specifies the Orca database editor as an example.
<b>Security</b>	0	The sample database is unrestricted read-write.

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Validating an Installation Upgrade

Whenever making any changes to the package, authors of upgrade packages should always run validation on their packages before attempting to install the package for the first time and rerun validation. Run validation on an upgrade package in the same way as for an installation package. For details, see [Validating an Installation Database](#).

This completes the sample upgrade package. To install the upgrade first install MNP2000.msi and then install MNP2001.msi. When you uninstall MNP2001 both the original and upgrade products are removed from your computer.

## Next example

[A Customization Transform Example](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# A Customization Transform Example

This example illustrates how a customization transform may be used to disable features and add new resources.

An administrator can permanently disable a feature by using a customization transform to enter a 0 into the Level column of the [Feature table](#). The application of the customization transform then prevents the installation and display of that feature even if the user selects a complete installation using the UI or by setting the **ADDLOCAL** property to ALL on the command line. For a discussion of installation level, see [Feature table](#) and **INSTALLLEVEL** property.

The resources needed to customize an application can be deployed by using a customization transform to add one or more new components. The transform must add one or more new features to contain these new components. For the rules that should be followed when deploying resources, such as files, registry keys, or shortcuts, see [Using Transforms to Add Resources](#).

This example illustrates how to create a [transform](#) to customize the installation of the application described in [An Installation Example](#). The original installation package installs all the features of the sample application, including the feature Gate, which enables users to view admissions information for the Red Park Arena. Some groups of users only need the application features that give event scheduling information, and do not need the Gate feature. These groups also need to get a special phone list. The transform must therefore do two things: 1) customize the installation so that this group only receives the application features they need and 2) provide the resources needed for the new phone list.

An example of a minimal user interface for this sample is provided in the [Windows SDK Components for Windows Installer Developers](#) as the file Uisample.msi. If you have the SDK, you have access to all the tools and data necessary to reproduce the sample installation package, user interface, and customization transform.

The customization transform has the following specifications:

- The customization transform is embedded inside the MNP2000.msi

file to guarantee that it is always available with the installation database.

- Installing MNP2000.msi with the customization transform does not install the Gate feature, child features of the Gate feature, or any of the components of the Gate feature, even if the user selects the Complete type of installation.
- Other applications may share some or all of the components of the Gate feature. The installation packages of these applications may install all their components on the user's computer.
- Removing MNP2000.msi with the customization transform does not remove any of the Gate components that have been installed by other applications.
- Installing MNP2000.msi with the customization transform also installs a new top level feature, Phone\_List, and a new component, phone, which requires the installation of the resource, Phone.txt. The user accesses the Phone\_List feature using a shortcut in the Menu directory.

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Customizing an Original Database

Make a copy of the sample Windows Installer installation package MNP2000.msi and rename this copy MNP2000t.msi. In the following steps you will customize this file using a database table editor such as Orca, which is provided with the SDK, or another database editor.

Include the new resource file for the phone list, Phone.txt, in the Notepad folder with the other source files.

File	Description	Path to source	Path to target
phone.txt	A resource for the Phone_List feature.	C:\Sample\notepad\phone.txt	[ProgramFilesFolder]\R

Use your database editor to add a record to the [File table](#) of MNP2000t.msi for the new file.

## File Table

File	Component_	FileName	FileSize	Version	Language	Attributes
Phone.txt	Phone	Phone.txt	1000			0

As explained in the section: [Using Transforms to Add Resources](#), the transform should add one or more new components to the installation database to contain the new phone list feature. Use your database editor to add the following record to the [Component table](#) of MNP2000t.msi.

The Phone component should be identified with a unique component ID [GUID](#). If you are reproducing the sample, do not reuse the same component ID GUID as in the following table. Instead use a utility such as Guidgen.exe to generate a new GUID. Be sure that you use a GUID string consistent with the Windows Installer [GUID](#) data type.

## Component Table

Component	ComponentId	Directory_	Attributes	Condition	Key
-----------	-------------	------------	------------	-----------	-----

Phone	{D152A1EC-9F7A-4E45-B0DC-ED6EE5D829F8}	NOTEPADDIR	2		Phon
-------	--	------------	---	--	------

Use your database editor to modify the data in the [Feature table](#) of MNP2000t.msi. Enter 0 into the Level column of the Gate feature record. This disables the Gate feature and its child features and hides these features from the UI. Note that because the [INSTALLLEVEL](#) property is set to 3 in the [Property table](#), the installer does not install features with a Level of 0. Add a record for the new Phone\_List feature.

### Feature Table

Feature	Feature_Parent	Title	Description	Display	Level	Director
Arts		Arts	Arts events at Red Park.	20	3	NOTEPA
Baseball	Sport	Baseball	Baseball Games	17	3	SPORTD
Concert	Arts	Concert	Concert events at Red Park	21	3	ARTSDI
Dance	Arts	Dance	Dance events at Red Park	23	3	ARTSDI
Football	Sport	Football	Football Games	19	3	SPORTD
Gate		Gate	Red Park's Admissions	6	0	NOTEPA
Help	Notepad	Help	Help file.	5	3	NOTEPA
January	Gate	January	January Admissions	10	3	MONDII
NewYears	January	New	New Years	11	3	HOLDIF

		Years Day	Day Admissions			
Notepad		Notepad	Notepad Editor	1	3	NOTEPA
Readme	Notepad	Readme	Readme File	3	3	NOTEPA
Sport		Sport Events	Sport Events at Red Park	14	3	NOTEPA
Phone_List		Phone List	Phone List	24	3	NOTEPA

Add the following record to the [FeatureComponents table](#) of MNP2000t.msi.

#### [FeatureComponents Table](#)

<b>Feature_</b>	<b>Component_</b>
Phone_List	Phone

Add a new record in the [Shortcut table](#) to create a shortcut to the Phone\_List feature.

#### [Shortcut Table](#)

<b>Shortcut</b>	<b>Directory_</b>	<b>Name</b>	<b>Component_</b>	<b>Target</b>	<b>Arguments</b>	<b>De</b>
sPhone	MENUDIR	Phone.txt	Phone	[#Phone.txt]		

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Generating a Customization Transform

You may generate a transform file by using [MsiDatabaseGenerateTransform](#) or the [GenerateTransform method](#) of the [Database object](#). An example of this is provided in the Windows Installer SDK as the utility `WiGenXfm.vbs`. The following snippet, `Gen.vbs`, also illustrates the [GenerateTransform](#) method and is for use with Windows Script Host.

```
'Gen.vbs. Argument(0) is the original database. Argument(1)
'      customized database. Argument(2) is the transform file

Option Explicit

' Check arguments
If WScript.Arguments.Count < 2 Then
    WScript.Echo "Usage is gen.vbs [original database] [cus
    WScript.Quit(1)
End If

' Connect to Windows Installer object
On Error Resume Next
Dim installer : Set installer = Nothing
Set installer = Wscript.CreateObject("WindowsInstaller.Inst
' Open databases
Dim database1 : Set database1 =
    installer.OpenDatabase(Wscript.Arguments(0), 0)
Dim database2 : Set database2 =
    installer.OpenDatabase(Wscript.Arguments(1), 0)
' Generate transform
Dim transform : transform = Database2.GenerateTransform(Dat
    Wscript.Arguments(2))
```

To generate the transform file `MNPtrans.mst` from the original `MNP2000.msi` database and the `MNP2000t.msi` database you modified in [Customizing an Original Database](#), change directories to the folder

containing Gen.vbs, the original database, and the updated installer database and enter the following command line.

**Cscript.exe Gen.vbs MNP2000.msi MNP2000t.msi MNPtrans.mst**

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Adding Summary Information to Customization Transform

To apply the customization transform during an installation of the product, you must add a [Summary Information Stream](#) to the transform file MNPtrans.mst generated in [Generating a Customization Transform](#).

You may generate summary information for a transform using [MsiCreateTransformSummaryInfo](#) or the [CreateTransformSummaryInfo Method](#). The following snippet, Sum.vbs, illustrates the [CreateTransformSummaryInfo Method](#) and is for use with Windows Script Host. Note that this example performs no validation and suppresses no error conditions.

```
'Sum.vbs. Argument(0) is the original database. Argument(1)
'      customized database. Argument(2) is the transform file

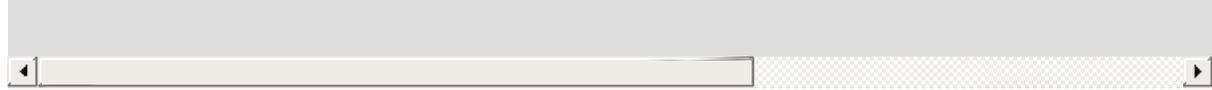
Option Explicit

' Check arguments
If WScript.Arguments.Count < 2 Then
    WScript.Echo "Usage is sum.vbs [original database] [cus
    WScript.Quit(1)
End If

' Connect to Windows Installer object
On Error Resume Next
Dim installer : Set installer = Nothing
Set installer = Wscript.CreateObject("WindowsInstaller.Inst

' Open databases and transform
Dim database1 : Set database1 =
    installer.OpenDatabase(Wscript.Arguments(0), 0)
Dim database2 : Set database2 =
    installer.OpenDatabase(Wscript.Arguments(1), 0)
Dim transform : transform = Wscript.Arguments(2)

' Create and add Summary Information
Dim transinfo : transinfo =
    Database2.CreateTransformSummaryInfo(Database1, transfo
```



To create and add summary information to the transform file MNPtrans.mst you created in [Generating a Customization Transform](#), change directories to the folder containing Gen.vbs, the original database, the updated database, and the transform, and enter the following command line.

**Cscript.exe Sum.vbs MNP2000.msi MNP2000t.msi MNPtrans.mst**

Click the MNP2000.msi icon to launch an install or use the following command line.

**msiexec /i MNP2000.msi**

This installs the product without the customizations. To install with the customization, enter the following command line. Note that the value of the **TRANSFORMS** Property refers to transform file located at the source.

**msiexec /i MNP2000.msi TRANSFORMS=MNPtrans.mst**

The Gate feature does not appear in the feature selection tree and the components of the Gate feature are not installed even if a Complete type of installation is selected in the user interface.

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Embedding Customization Transforms as Substorage

You may store the customization transform in a storage of the Windows Installer package to guarantee that the transform is always available when the installation package is available. See [Embedded Transforms](#). An example of this is provided in the Windows Installer SDK as the utility `WiSubStg.vbs`. The following snippet, `Emb.vbs`, also illustrates the use of the [Storages table](#) to add an embedded transform and is for use with Windows Script Host.

```
'Emb.vbs. Argument(0) is the original database. Argument(1)
'      path to the transform file. Argument(2) is the name of
'
Option Explicit

' Check arguments
If WScript.Arguments.Count < 2 Then
    WScript.Echo "Usage is emb.vbs [original database] [transform file]"
    WScript.Quit(1)
End If

' Connect to Windows Installer object
On Error Resume Next
Dim installer : Set installer = Nothing
Set installer = Wscript.CreateObject("WindowsInstaller.Installer")

' Evaluate command-line arguments and set open and update records
Dim databasePath: databasePath = Wscript.Arguments(0)
Dim importPath  : importPath = Wscript.Arguments(1)
Dim storageName : storageName = Wscript.Arguments(2)

' Open database and create a view on the _Storages table
Dim sqlQuery : sqlQuery = "SELECT `Name`, `Data` FROM _Storages"
Dim database : Set database = installer.OpenDatabase(databasePath)
Dim view     : Set view = database.OpenView(sqlQuery)

' Create and Insert the row.
Dim record   : Set record = installer.CreateRecord(2)
```

```
record.StringData(1) = storageName
view.Execute record

'Insert storage - copy data into stream
record.SetStream 2, importPath
view.Modify 3, record
database.Commit
Set view = Nothing
Set database = Nothing
```

To add a storage named MNPtrans1 to MNP2000.msi, and containing the transform you created in [Adding Summary Information to Customization Transform](#), change directories to the folder containing Emb.vbs, the original database, and the transform file, then enter the following command line.

### **Cscript.exe Emb.vbs MNP2000.msi MNPtrans.mst MNPtrans1**

This completes the customization transform example. After embedding the transform in MNPtrans.mst, the transform is always available with the installation package. The file MNPtrans.mst does not need to be located at the source to apply the transform.

Remove MNPtrans.mst from the folder containing the sample installation package. Click the MNP2000.msi icon to launch an install or use the following command line.

### **msiexec /i MNP2000.msi**

Note that this installs the product without the customizations. To install with the customizations, enter the following command line. Use a colon to indicate that the value of the **TRANSFORMS** Property refers to an embedded transform.

```
msiexec /i MNP2000.msi TRANSFORMS=:MNPtrans1
```

Note that the Gate feature does not appear in the feature selection tree and that the components of the Gate feature are not installed even if a Complete type of installation is selected in the user interface.

## **Next example**

## A Small Update Patching Example

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# A Small Update Patching Example

This example illustrates how to create a [patch package](#) that applies a [small update](#) to the sample installation package discussed in [An Installation Example](#). A small update makes changes to one or more application files that are judged to be too minor to warrant changing the product code. For more information see [Patching and Upgrades](#).

This example illustrates how to create a Windows Installer patch package that updates the Concert feature of the hypothetical product MNP2000 to fix an error in the original product. The example patch package applies a small update to the product that does not require changing the product code. See the topic on [Major Upgrades](#) in the [Patching and Upgrades](#) section for more information about major upgrades.

The sample upgrade package has the following specifications:

- It corrects a minor error in the Concert schedule displayed by the Concert feature.
- It updates the package code to reflect the installation package has changed.
- The small update can be applied by patching the local installation of the product.

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Planning a Small Update Patch

The Concert feature file of the original product, MNP2000, contains an error in the Concert.txt file. Because Windows Installer was used for the installation and setup of the application, minor fixes to the application may be handled by installing a small update patch package. A [small update](#) makes changes to one or more application files that are too minor to change the product code. The following example shows you how to create a Windows Installer patch package that can apply the small update and provide a quick fix to the MNP2000 product.

To create the small update first obtain a fully uncompressed image of the MNP2000 product that includes the error in Concert.txt. The image must include MNP2000.msi and all the source files described in [Planning the Installation](#). In the following discussion this is called the Target image. The Target image must be fully uncompressed because the patch creation process is unable to generate binary patches for files compressed in cabinets. Put the .msi file and all the source files of the Target image into a folder named Target.

Next, obtain a fully uncompressed image of the MNP2000 product with a Concert.txt file that is fixed. This is called the Upgraded image in the following discussion. Use an installation database editing tool, such as Orca, to update the .msi file. For example, if the size of the corrected Concert.txt is smaller than the original, be sure to enter the new size in the FileSize field of the File table of the upgraded image. Note that because the package has changed you must assign a new package code in the [Revision Number Summary](#) Property. Put the .msi file and all the source files of the Upgraded image into a folder named Upgraded.

For the purposes of this example, assume that the size of the Concert.txt file changes. This means that FileSize fields in the File tables of the Target and Upgraded database contain different data.

The following [File Table](#) identifies the record from the Target Image.

File	Component_	FileName	FileSize	Version	Language	Attribut
Concert.txt	Concert	Concert.txt	1000			0

The following File Table identifies the record from the Upgraded Image.

File	Component_	FileName	FileSize	Version	Language	Attribut
Concert.txt	Concert	Concert.txt	900			0

**Note** The file must have the same key in the File Tables of both the target image and the updated image. The string values in the File column of both tables must be identical. Uppercase and lowercase must be identical also.

Follow the guidelines described in [Creating a Patch Package](#). Do not author a package with File Table keys that differ only by case, because [Msimsp.exe](#) and [Patchwiz.dll](#) call Makecab.exe, which is case-insensitive and patch generation fails.

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Creating a Patch Creation Properties File

To reproduce the sample patch package, you need a software tool capable of creating and editing a Windows Installer patch package. Several patch package creation tools are available from independent software vendors. The example discussed in the following sections uses a Windows Installer database editor called Orca to author a patch creation properties file (.pcp extension). The .pcp file may be used with the utilities [Msimsp.exe](#) and [Patchwiz.dll](#) to generate a Windows Installer patch package (.msp extension). Orca, Msimsp.exe, and Patchwiz.dll are provided in the [Windows SDK Components for Windows Installer Developers](#).

A blank patch creation properties file, template.pcp, is also provided with the SDK. Make a copy of template.pcp and rename this copy MNP2000.pcp. Use Orca or another database editor to enter the following data into the Properties table of MNP2000.pcp. The Properties table contains global settings for the patch package.

## Properties Table

Name	Value
AllowProductCodeMismatches	1
AllowProductVersionMajorMismatches	1
ApiPatchingSymbolFlags	0x00000000
DontRemoveTempFolderWhenFinished	1
IncludeWholeFilesOnly	0
ListOfPatchGUIDsToReplace	
ListOfTargetProductCodes	*
PatchGUID	{5406B219-A1AC-4BC4-8695-72292C8195AC}
PatchOutputPath	c:\output.msp
PatchSourceList	PatchSourceList

Use the database editor to enter the following data into the ImageFamilies table of MNP2000.pcp. The ImageFamilies table contains information to be added to the [Media table](#) during patching.

### ImageFamilies Table

Family	MediaSrcPropName	MediaDiskId	FileSequenceStart	DiskPro
MNPapps	MNPSrcPropName	2	1000	

Enter the following data into the UpgradedImages table of MNP2000.pcp. The UpgradedImages table contains information about the Upgraded image you created in [Planning a Small Update Patch](#).

### UpgradedImages Table

Upgraded	MsiPath	PatchMsiPath
MNP_fixed	C:\Note_Installer\Patch\Upgraded\MNP2000.msi	

Enter the following data into the TargetImages table of MNP2000.pcp. The TargetImages table contains information about the Target image.

### TargetImages Table

Target	MsiPath	SymbolPaths	Upg
MNP_error	C:\Note_Installer\Patch\Target\MNP2000.msi		MN

For the sample patch package, leave the following tables in MNP2000.pcp blank.

[UpgradedFiles\\_OptionalData Table](#)

[FamilyFileRanges Table](#)

[TargetFiles\\_OptionalData Table](#)

[ExternalFiles Table](#)

[UpgradedFilesToIgnore Table](#)

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Generating a Patch Package

The recommended method for generating a patch package is to use a patch creation tool such as [Msimsp.exe](#) to call [UiCreatePatchPackage](#) in [Patchwiz.dll](#). Msimsp.exe and PatchWiz.dll are provided in the Windows Installer SDK.

The following example command line uses Msimsp.exe and the .pcp file created in [Creating a Patch Creation Properties File](#) to generate the sample patch package MNP2000.msp.

```
Msimsp.exe -s C:\Note_Installer\Patch\MNP2000.pcp -p  
C:\Note_Installer\Patch\MNP2000.msp
```

An authored patch creation tool may generate the patch package by calling [UiCreatePatchPackage](#) as follows.

```
UiCreatePatchPackage ("C:\\Note_Installer\\Patch\\MNP2000.pcp
```

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Applying a Patch Package to a Local Installation

You may apply the small update to a local installation of MNP2000 with the sample patch MNPpatch.msp created in [Generating a Patch Package](#). The general procedure is discussed in the section [Applying Small Updates by Patching the Local Installation of the Product](#).

Install the original MNP2000 product locally on your computer. Verify that this has the error Concert.txt that the small update is to fix. Next launch the installation by entering the following command line.

```
msiexec /p MNP2000.msp REINSTALL=ALL  
REINSTALLMODE=omus
```

Reexamine the Concert.txt belonging to MNP2000 to verify that the installer has applied the small update to fix this file.

To apply the small update to an administrative image, see [Applying a Patch Package to an Administrative Installation](#).

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Applying a Patch Package to an Administrative Installation

You may apply the small update to an administrative source image of MNP2000.msi by installing the sample patch MNP2000.msp created in [Generating a Patch Package](#). The update can then be propagated to users by requesting that they reinstall the application from the new administrative source image.

An administrator can use the following command line to update the administrative source image located at //server/MNP2000.msi to a new source image that is the same as would be produced by an administrative installation from a fully updated CD-ROM.

```
msiexec /a //server/MNP2000.msi /p MNP2000.msp
```

Members of the workgroup using MNP2000 then must reinstall the application from the new administrative source image to receive the update.

To completely reinstall the applications and cache the updated .msi file on the user's computer, users may enter either of the following commands.

```
msiexec /fvomus //server/MNP2000.msi
```

```
msiexec /i //server/MNP2000.msi REINSTALL=ALL  
REINSTALLMODE=vomus
```

To reinstall only the updated Concert feature and cache the updated .msi file on the user's computer, users may enter the following command.

```
msiexec /i //server/MNP2000.msi REINSTALL=Concert  
REINSTALLMODE=vomus
```

## Next example

[A Localization Example](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# A Database and Patch Example

An application can use the [MsiOpenDatabase](#) function to open a new or existing installation database (.msi file) or patch package (.msp file.) The application checks the return value of **MsiOpenDatabase** before using the database handle.

The following examples use the **PMSIHANDLE** type variables defined in msi.h. It is recommended to use the **PMSIHANDLE** type because the installer closes **PMSIHANDLE** objects as they go out of scope, whereas your application must close **MSIHANDLE** objects by calling [MsiCloseHandle](#). For more information see [Use PMSIHANDLE instead of HANDLE](#) section in the [Windows Installer Best Practices](#).

The following example opens a database, sample.msi, for reading only. **MsiOpenDatabase** succeeds only if sample.msi exists in the c:\test directory. Upon success, the returned database handle can be used to query the data in the installation package using [MsiDatabaseOpenView](#) and [MsiGetSummaryInformation](#).

```
PMSIHANDLE hDbReadOnly = 0;
UINT uiStatus1 = MsiOpenDatabase(TEXT("c:\\test\\sample.msi"));
if (ERROR_SUCCESS != uiStatus1)
{
    // process error
    return uiStatus1;
}
```

The following example opens the database for reading and writing. If the application calls [MsiDatabaseCommit](#), all changes made to the database are saved. If the application does not call **MsiDatabaseCommit**, no changes are made to the database.

```
PMSIHANDLE hDbTransact = 0;
UINT uiStatus2 = MsiOpenDatabase(TEXT("c:\\test\\example.msi"));
if (ERROR_SUCCESS != uiStatus2)
{
    // process error
}
```

```
        return uiStatus2;
    }
```

The following example takes an existing database, text.msi, and creates a new database, newtest.msi. Any changes that are made can be saved in the new database by calling **MsiDatabaseCommit**. The existing database specified in the *szDatabasePath* parameter is unchanged.

```
PMSIHANDLE hDbOutput = 0;
UINT uiStatus3 = MsiOpenDatabase(TEXT("c:\\test\\test.msi"),
if (ERROR_SUCCESS != uiStatus3)
{
    // process error
    return uiStatus3;
}
```

The following example opens a Windows Installer patch package (.msp file) for reading only. The returned patch handle can be used to determine the cabinets and transform substorages included in the patch package by queries on the [\\_Streams](#) and [\\_Storages](#) tables.

**Windows Installer 2.0:** Not supported. Beginning with Windows Installer 3.0, the application can query the [MsiPatchSequence table](#) present in a patch package that uses the new patch sequencing information.

```
PMSIHANDLE hDbPatch = 0;
LPCTSTR szPersistMode = MSIDBOPEN_READONLY + MSIDBOPEN_PATCH
UINT uiStatus4 = MsiOpenDatabase(TEXT("c:\\test\\sample.msp"),
if (ERROR_SUCCESS != uiStatus4)
{
    // process error
    return uiStatus4;
}
```

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# A Localization Example

This example illustrates how to localize the Windows Installer package described in [An Installation Example](#). The original installation package is changed from an English into French language version.

The localization sample has the following specifications:

- The installation UI for the application MNP2000 should be changed from English to French.
- The French version of MNP2000 includes Readme.txt and Help.txt files written in the French language.
- The French version includes a list of travel agents in France that can provide tickets to Red Park Arena. This list (provided as a new file, Fre.txt) is not a part of the English version.

The general procedure for localizing an installation is described in the section [Localizing a Windows Installer Package](#).

Copy the English version of MNP2000.msi and all the source files described in [Planning the Installation](#) into a new folder. Change the name of the MNP2000.msi in the folder to MNPFren.msi. In the following sections you will modify this copy to localize the application into French.

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Checking the Installation Database Code Page

A *code page* is an internal table that the operating system uses to map symbols (letters, numerals, and punctuation characters) to a character number. Different code pages provide support for the character sets used in different countries or regions. Code pages are referred to by number; for example, code page 932 represents the Japanese character set, and code page 950 represents one of the Chinese character sets. See [Localizing the Error and ActionText Tables](#) for a list of ASCII code pages.

The same ASCII code page, 1252, may be used for English and French. Therefore the code page for the database MNP2000.msi (English) does not need to be reset to change the installation to French.

For more information about setting the code page see [Code Page Handling \(Windows Installer\)](#).

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Importing Localized Error and ActionText Tables

Localized language versions of the [Error table](#) and [ActionText table](#) are provided by the Windows Installer SDK. The French language versions of these tables, Error.FRA and ActionTe.FRA, are located in the Intl folder of the Windows Installer SDK.

You may use the table editor Orca or the utility Msidb.exe provided with the SDK to import the French versions of these tables into the database.

An example of using [MsiDatabaseImport](#) and the [Import Method](#) of the [Database object](#) is provided in the Windows Installer SDK as the utility Wilimport.vbs. The following snippet, Imp.vbs, also illustrates the use of the Import method and is for use with Windows Script Host.

```
'Imp.vbs. Argument(0) is the original database. Argument(1)
' path of the folder containing the file to be imported.
'
Option Explicit

' Check arguments
If WScript.Arguments.Count < 2 Then
    WScript.Echo "Usage is imp.vbs [original database] [folder]"
    WScript.Quit(1)
End If

' Connect to Windows Installer object
On Error Resume Next
Dim installer : Set installer = Wscript.CreateObject("WindowsInstaller.Installer")
Dim databasePath : databasePath = Wscript.Arguments(0)
Dim folder : folder = Wscript.Arguments(1)

' Open database and process file
Dim database : Set database = installer.OpenDatabase(databasePath)
Dim table : table = Wscript.Arguments(2)
database.Import folder, table

' Commit database changes
database.Commit 'commit changes
```

```
Wscript.Quit 0
```

To import and replace the Error table with Error.FRA you may use a command line such as the following.

**Cscript Imp.vbs MNPFren.msi C:\Note\_Installer\French Error.FRA**

To import and replace the ActionText table with ActionTe.FRA you may use a command line such as the following.

**Cscript Imp.vbs MNPFren.msi C:\Note\_Installer\French  
ActionTe.FRA**

Rerun validation on MNPFren.msi as described in [Validating an Installation Upgrade](#).

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.



CancelDlg	Text	Text	48	15	194	30	3		Êtes voul l'ins [Pro
-----------	------	------	----	----	-----	----	---	--	-------------------------------

If the installation of MNPFren.msi is canceled by the user, the **Cancel Dialog** appears displaying the text: *"Êtes-vous sûr de vouloir annuler l'installation de MNP2000?"*

When using this method to localize UI text to a different language, the localized UI must be tested to ensure that the size of controls are large enough to display the entire localized text. This should be tested using all font size settings that are available for display. Localized text can require more space than the original text and may be truncated if displayed in a control that is too small.

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Updating ProductLanguage and ProductCode Properties

The **ProductLanguage** property must be updated to the numeric language identifier (LANGID) for the new language. In the case of the localization example, the value of the **ProductLanguage** property must be changed from the LANGID for English (1033) to the LANGID for French (1036) in the [Property table](#).

The value of the **ProductCode** property in the [Property table](#) must be changed to a new, unique value when localizing a database because a localized product is considered a different product. For example, the German and English versions of an application are considered two different products and must have different product codes. See [Product Codes](#).

Use your database table editor to update the values of the ProductCode and ProductLanguage properties in the Property table. Do not reuse the GUID shown if you copy this example.

## Property Table

Property	Value
<b>ProductCode</b>	{EE389960-E426-4EEA-B669-AD8129249881}
<b>ProductLanguage</b>	1036

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Updating a Summary Information Stream

The value of the **Revision Number Summary** Property in the [summary information stream](#) must be changed to a new, unique value when localizing a database, because the installation database is being changed. See [Package Codes](#).

The value of the **Template Summary** Property in the summary information stream must be changed to the numeric language identifier (LANGID) of the new language.

If the descriptive text strings in the summary information stream are localized to the new language, the **Codepage Summary** Property must be set to the correct code page for the new language.

You may modify the summary information stream of the localized package by the same procedure as in [Adding Summary Information](#). An example demonstrating the use of the database summary information methods is also provided in the Windows Installer SDK as the utility WiSumInf.vbs. For more information about WiSumInf.vbs, see [Manage Summary Information](#).

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Adding Localized Resources

Depending upon the application, localization may require modifying or adding resources such as files or registry keys. The localization of the sample application MNP2000 requires adding one additional file to the package, Fre.txt, and the French versions of two existing files: Help.txt and Readme.txt.

The best practice in this case is to localize the package such that the English and French versions can safely coexist on the computer. This includes never installing two different files with identical file names into the same directory. Because Help.txt and Readme.txt have identical file names in the two language versions, the French package should install these files into a different directory than the English.

The French package installs Help.txt into a new subdirectory of the RedPark folder, French. Because the addition of Fre.txt adds a resource to the original Help component, the component code for the Help component must be different in the French and English packages. See the rules for component codes in [Changing the Component Code](#).

The French package installs Readme.txt into the directory French so that this file name cannot conflict with the English version. The file Readme.txt is installed with the Notepad component, but the component rules do not require changing the component code. In this example, the component code of Notepad should not be change because RedPark.exe, the registry values specified in the [Registry table](#), are shared by both language versions. See [Adding Registry Information](#).

Remove the English versions of Help.txt and Readme.txt from the source files and add the new French versions of Help.txt, Readme.txt, and Fre.txt. The localized package should map the installation of files from source to target as follows.

File	Description	Path to source	Path to target
Redpark.exe	Text editor executable file.	C:\Sample\Notepad\Redpark.exe	[ProgramFilesFc
Readme.txt	An	C:\Sample\Notepad\Readme.txt	[ProgramFilesFc

	informational file.		
Help.txt	Help manual	C:\Sample\notepad\Help.txt	[ProgramFilesFc
Fre.txt	Phone list	C:\Sample\notepad\Fre.txt	[ProgramFilesFc

Use the database editor Orca that is provided with the SDK, or another editor, to open the Directory table and add a record for the installation of the new directory, French.

### Directory Table

Directory	Directory_Parent	DefaultDir
<b>TARGETDIR</b>		SourceDir
<b>ProgramFilesFolder</b>	<b>TARGETDIR</b>	.
ARTSDIR	NOTEPADDIR	Arts:Events
HOLDIR	MONDIR	.:Holidays
MENUDIR	NOTEPADDIR	Menu
MONDIR	NOTEPADDIR	Gate
NOTEPADDIR	<b>ProgramFilesFolder</b>	Red_Park:Notepad
SPORTDIR	NOTEPADDIR	Sports:Events
FRENCHDIR	NOTEPADDIR	French:.

Use your table editor to change the ComponentId of the Help component in MNPFren.msi to a new GUID.

### Component Table

Component	ComponentId	Directory_	Attributes	Condition	Key
Baseball	{F54ABAC0-33F2-11D3-91D7-00C04FD70856}	SPORTDIR	2		Baseb

Concert	{76FA7A80-33F6-11D3-91D8-00C04FD70856}	ARTSDIR	2		Conce
Dance	{CCF834A1-33F8-11D3-91D8-00C04FD70856}	ARTSDIR	2		Dance
Football	{CCF834A0-33F8-11D3-91D8-00C04FD70856}	SPORTDIR	2		Footb
Help	{9ED21229-FE3C-4FE9-B01D-57E00224FD0B}	NOTEPADDIR	2		Help:
January	{CF0BC690-33C9-11D3-91D6-00C04FD70856}	MONDIR	2		Janua
NewYears	{A42D9140-33D8-11D3-91D6-00C04FD70856}	HOLDIR	2		NewY
Notepad	{19BED232-30AB-11D3-91D3-00C04FD70856}	FRENCHDIR	2		Redpa

Use your table editor to add Fre.txt to the [File table](#) of MNPfren.msi. Enter the LANGID for French into the Language field for the localized files. All other things being equal, if the file being installed has a different language than the file on the machine, the installer favors the file with the language that matches the product being installed. Language neutral files are treated as just another language so the product being installed is

favored again. For more information, see [File Versioning Rules](#).

## File Table

File	Component_	FileName	FileSize	Version	Language	Attr
Baseball.txt	Baseball	Baseball.txt	1000			0
Concert.txt	Concert	Concert.txt	1000			0
Dance.txt	Dance	Dance.txt	1000			0
Football.txt	Football	Football.txt	1000			0
Help.txt	Help	Help.txt	1000		1036	0
January.txt	January	January.txt	1000			0
NewYears.txt	NewYears	NewYears.txt	1000			0
Redpark.exe	Notepad	Redpark.exe	45328			0
Readme.txt	Notepad	Readme.txt	1000		1036	0
Fre.txt	Help	Fre.txt	1000		1036	0

This completes the sample localization.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# A MUI Shortcut Example

This section describes how to add resource strings to the Windows Installer [Shortcut](#) table for use with Multilingual User Interfaces (MUI).

**Windows Installer 2.0 and Windows Installer 3.0:** Not supported.  
This example requires Windows Installer 4.0.

Refer to the Multilingual User Interface (MUI) documentation for information on how to develop MUI-enabled applications .

To add the resource strings used by Windows Vista Multilingual User Interfaces to a Windows Installer package:

1. Add the information for all the language-neutral and language files to the [File Table](#). For example, the files might consist of a language-neutral file (msimsg.dll) and language files for English (msimsgen.dll.mui), Japanese (msimsgja.dll.mui), and Chinese (msimsgcs.dll.mui). Each file can belong to a different component. Each file can have both a long and short file name. In the case of this example, the following information can be added to the [File Table](#).

[File Table](#) (partial)

<b>File</b>	<b>Component_</b>	<b>FileName</b>
msimsgmuija	MSIMSG_MUI_JA	msimsgja.dll msimsg.dll.mui
msimsgmuics	MSIMSG_MUI_CS	msimsgcs.dll msimsg.dll.mui
msimsgmuien	MSIMSG_MUI_EN	msimsgen.dll msimsg.dll.mui
msimsgdll	MSIMSG	msimsg.dll

2. Add information to the [Component table](#) for these components. Each component has a unique GUID identifier that should be entered into the ComponentId field of the Component table. The

file belonging to the component can serve as the KeyPath for that component. The directory that contains each component can be specified in the Directory\_ field. The following information can be added to the Component table.

**Component Table** (partial)

<b>Component</b>	<b>Directory_</b>	<b>KeyPath</b>
MSIMSG_MUI_JA	MUIFolder_JA	msimsgmuija
MSIMSG_MUI_CS	MUIFolder_CS	msimsgmuics
MSIMSG_MUI_EN	MUIFolder_EN	msimsgmuien
MSIMSG	MUIFolder	msimsgdll

3. Edit the **Directory** table so that the components are installed into the correct directories. Be sure to include information about the directory where the shortcut will be installed. For example, the following information might be added to the Directory table of a package that installs the components and a shortcut located in the DesktopFolder directory.

**Directory Table** (partial)

<b>Directory</b>	<b>Directory_Parent</b>	<b>DefaultDir</b>
TARGETDIR		SourceDir
MsiTest	TARGETDIR	MsiTest:.
MUIFolder	MsiTest	MUI
MUIFolder_CS	MUIFolder	cs-CZ
MUIFolder_EN	MUIFolder	en-US
MUIFolder_JA	MUIFolder	ja-JP
DesktopFolder	TARGETDIR	.

4. Add a row to the [Shortcut](#) table for each shortcut. For example, the [Shortcut](#) table could contain the following information for two shortcuts, Quick1 and Quick2, installed into the DirectoryFolder directory. Each shortcut belongs to the feature specified in the Target field. The icon associated with the shortcut can be specified in the Icon\_ field and the [Icon](#) table.

[Shortcut Table](#) (partial)

<b>Shortcut</b>	<b>Directory_</b>	<b>Component_</b>	<b>Target</b>	<b>Icon</b>
Quick1	DesktopFolder	MSIMSG	FeatureChild1_Local	HelpFil
Quick2	DesktopFolder	MSIMSG	FeatureChild1_Local	HelpFil

5. Add information to the [Feature Table](#) table for the feature owns shortcut belongs. When the shortcut is activated, the installer verifies that all the components belonging to this feature are installed before launching the key file of the component specified in the Component\_ column of the [Shortcut](#) table. In the case of this example, the following information can be added to the Feature Table table for the FeatureParent1\_Local feature.

[Feature Table](#) (partial)

<b>Feature</b>	<b>Feature_Parent</b>	<b>Title</b>	<b>A</b>
FeatureParent1_Local		FeatureParent1_Local	1
FeatureChild1_Local	FeatureParent1_Local	FeatureParent1_Local	0

6. For each new shortcut, add the resource string information to the DisplayResourceDLL, DisplayResourceId, DescriptionResourceDLL, and DescriptionResourceId fields of the [Shortcut table](#). The DisplayResourceDLL and DescriptionResourceDLL fields contain the resource string in the

**Formatted** string format. The formatted string can use the `[#filekey]` convention of the **Formatted** format. Add the display and description indices for the resource strings in the `DisplayResourceId` and `DescriptionResourceId` fields.

**Shortcut Table** (partial)

<b>Shortcut</b>	<b>DisplayResourceDLL</b>	<b>DisplayResourceId</b>	<b>DescriptionResourceId</b>
Quick1	[#msimsgdll]	36	[#msimsgdll]
Quick2	[#msimsgdll]	38	[#msimsgdll]

7. After installing the package, test to ensure that the Multilingual User Interface is working as expected.

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# A URL-Based Windows Installer Installation Example

This example illustrates how to create a URL-based installation of a Windows Installer package. For more information about securing installations and using digital certificates see [Guidelines for Authoring Secure Installations](#) and [Digital Signatures and Windows Installer](#).

To reproduce this sample you need the SignTool utility. For details, see the CryptoAPI Tools Reference in the Microsoft Windows Software Development Kit (SDK). You also need [Msistuff.exe](#) and Setup.exe utilities from the [Windows SDK Components for Windows Installer Developers](#). For more information, see [Internet Download Bootstrapping](#).

The example has the following specifications:

- When users visit your Web site and click the "MySetup Installation" link, they are presented with the option to save or run from that location. If the user selects to run from that location, the Setup.exe upgrades the version of Windows Installer on their computer, if necessary, verifies the digital signature on the installer package, and installs the package on their computer.
- There is a digital certificate, Mycert.cer, provided with a private key, Mycert.pvk.
- The URL of the hypothetical Web site a customer would visit to install the package is <http://www.blueyonderairlines.com/Products/MySetup/mysetup.html>.
- The Web server layout is as follows.

URL	F
<a href="http://www.blueyonderairlines.com/Products/MySetup/">http://www.blueyonderairlines.com/Products/MySetup/</a>	S
<a href="http://www.blueyonderairlines.com/Products/MySetup/">http://www.blueyonderairlines.com/Products/MySetup/</a>	N

<a href="http://www.blueyonderairlines.com/Products/MySetup/">http://www.blueyonderairlines.com/Products/MySetup/</a>	C
<a href="http://www.blueyonderairlines.com/Products/MySetup/">http://www.blueyonderairlines.com/Products/MySetup/</a>	C
<a href="http://www.blueyonderairlines.com/Products/Common/InstMsi/Ansi">http://www.blueyonderairlines.com/Products/Common/InstMsi/Ansi</a>	Ii
<a href="http://www.blueyonderairlines.com/Products/Common/InstMsi/Unicode">http://www.blueyonderairlines.com/Products/Common/InstMsi/Unicode</a>	Ii

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Configuring the Setup.exe Resources

A configurable bootstrap executable (Setup.exe) and configuration tool ( [Msistuff.exe](#)) is included in the [Windows SDK Components for Windows Installer Developers](#). By using Msistuff.exe to configure the resources in Setup.exe, developers can easily create a Web installation of a Windows Installer package. For more information see [Internet Download Bootstrapping](#).

Entering the following command line configures the resources for the sample described in [A URL Based Windows Installer Installation Example](#).

```
MsiStuff setup.exe /u  
http://www.blueyonderairlines.com/Products/MySetup /d  
MySetup.msi /n MySetup /v 150 /i  
http://www.blueyonderairlines.com/Products/Common/InstMsi /a  
Ansi/Instmsi.exe /w Unicode/Instmsi.exe
```

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Sign Setup.exe and MySetup.msi

Before you place Setup.exe and MySetup.msi on the Web server, you should sign the files with your digital certificate and private key, Mycert.cer and Mycert.pvk, using the SignTool utility. For more information about using the SignTool utility, see [CryptoAPI Tools Reference](#) in the Microsoft Windows Software Development Kit (SDK).

[Continue](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Establish an HTML Reference to Setup.exe

The final step is to place a reference to the Setup.exe on the hypothetical MySetup Web page (MySetup.html) described in [A URL Based Windows Installer Installation Example](#). Use the following HTML script:

```
<a href="http://www.blueyonderairlines.com/Products/MySetup/
```

Clicking on the "MySetup Installation" link presents users with the option to save or run from that location. If the user selects run from that location, the Setup.exe upgrades the version of Windows Installer on the computer, if necessary, verifies the digital signature on the installer package, and installs the package on their computer.

This completes the example.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Windows Installer Scripting Examples

The [Windows SDK Components for Windows Installer Developers](#) contains VBScript files that show you how the Windows Installer automation interface is used to modify Windows Installer packages.

The script samples identified in this topic are not supported by Microsoft Corporation, and they are provided only as a potentially useful reference. Running these samples requires the Windows Script Host. For more information about Windows Script Host, see the [Windows Script Host](#) section of the Microsoft Windows Software Development Kit (SDK).

Sample Script File	Description
WiLstPrd.vbs	<a href="#">List Products, Properties, Features, and Components</a>
WiImport.vbs	<a href="#">Import Files</a>
WiExport.vbs	<a href="#">Export Files</a>
WiSubStg.vbs	<a href="#">Manage Substorages</a>
WiStream.vbs	<a href="#">Manage Binary Streams</a>
WiMerge.vbs	<a href="#">Merge Two Databases</a>
WiGenXfm.vbs	<a href="#">Generate a Transform</a>
WiUseXfm.vbs	<a href="#">Apply a Transform</a>
WiLstXfm.vbs	<a href="#">View a Transform (CSCRIPT only)</a>
WiDiffDb.vbs	<a href="#">View Differences Between Two Databases (CSCRIPT only)</a>
WiLstScr.vbs	<a href="#">View Installer Script (CSCRIPT only)</a>
WiSumInf.vbs	<a href="#">Manage Summary Information</a>
WiPolicy.vbs	<a href="#">Manage Policy Settings</a>
WiLangId.vbs	<a href="#">Manage Language and Codepage</a>
WiToAnsi.vbs	<a href="#">Copy a Unicode File to an Ansi File</a>

WiFilVer.vbs	<a href="#">Manage File Sizes and Versions</a>
WiMakCab.vbs	<a href="#">Generate File Cabinet</a>
WiRunSQL.vbs	<a href="#">Execute SQL Statements</a>
WiTextIn.vbs	<a href="#">Copy ANSI File Into a Database Field</a>
WiCompon.vbs	<a href="#">List Components</a>
WiFeatur.vbs	<a href="#">List Features</a>
WiDialog.vbs	<a href="#">Preview User Interface</a>

All these scripts display a help screen that describes their command line arguments. To display the help screen in Windows double-click the file. To display the help screen from a command line enter a ? as the first argument, or enter fewer arguments than required. Scripts return a value of 0 for success, 1 if help is invoked, and 2 if in case of failure.

These samples require Windows Script Host to run. Windows Script Host is actually two hosts:

- CScript.exe is the version that enables you to run scripts from the command prompt, and provides command-line switches for setting script properties.
- WScript.exe is the version of Windows Script Host that enables you to run scripts from Windows. For more information, see the [Windows Script Host](#) section in the Windows SDK.

The Makecab.exe utility is included with the patching samples in the [Windows SDK Components for Windows Installer Developers](#).

For information about WMI, see [Using Windows Installer with WMI](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# List Products, Properties, Features, and Components

The VBScript file `WiLstPrd.vbs` is provided in the [Windows SDK Components for Windows Installer Developers](#). The sample script connects to an **Installer** object and enumerates registered products and product information.

This sample demonstrates the use of:

- **ProductInfo** property
- **ProductState** property (Installer object)
- **Products** property
- **Features** property
- **FeatureParent** property
- **FeatureState** property
- **Components** property
- **ComponentClients** property
- **ComponentPath** property
- **LastErrorRecord** method
- **RegistryValue** method of the **Installer** object

You'll require the `CScript.exe` or `WScript.exe` version of Windows Script Host to use this sample. To use `CScript.exe` to run this sample, type a command line at the command prompt using the following syntax. Help is displayed if the first argument is `/?` or if too few arguments are specified. To redirect the output to a file, end the command line with `VBS > [path to file]`. The sample returns a value of 0 for success, 1 if help is invoked, and 2 if the script fails.

**`cscript WiLstPrd.vbs [Product Name] [options]`**

Specify the case-insensitive product name or the product ID GUID of the installed or advertised product. If no product or options are specified, the installer lists all the products installed or advertised on the system.

Note that these options are not switches so you should not prefix them with a slash (/) on the commandline. The following options may be combined by concatenating the letters. For example, "pc" to list both the products' properties and installed components.

Option	Description
no options specified	List the products' properties.
p	List the products' properties.
f	List the products' features, feature parents, and installation states
c	List the products' installed components.
d	List the value under <b>HKLM\Software\Microsoft\Windows\CurrentVersion\SharedDlls</b> for the key files of the products' component.

For more information, see [Windows Installer Scripting Examples](#) for additional scripting examples. For sample utilities that do not require Windows Script Host, see [Windows Installer Development Tools](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Import Files

The VBScript file Wilimport.vbs is provided in the [Windows SDK Components for Windows Installer Developers](#). This sample shows how to write a script to import tables into a Windows Installer database.

The script connects to an **Installer** object, opens a database, processes a list of files, and commits the changes before closing the database.

The sample demonstrates the use of:

- **OpenDatabase Method (Installer Object)**
- **LastErrorRecord method** of the **Installer object**
- **Import method**
- **Commit method** of the **Database object**

You'll require the CScript.exe or WScript.exe version of Windows Script Host to use this sample. To use CScript.exe to run this sample, use the following syntax at the command prompt.

**cscript Wilimport.vbs [path to database][path to folder][options]  
[archive file list]**

Help is displayed if the first argument is /? or if too few arguments are specified. To redirect the output to a file, end the command line with VBS > [path to file]. The sample returns a value of 0 for success, 1 if help is invoked, and 2 if the script fails.

Specify the path to a Windows installer database that is to be created or that is to receive the imported tables. Specify the path to the folder containing the archive files of the tables that are being imported. List the names of the archive files that are being imported. Wildcard file names, such as \*.idt, can be used to import multiple files.

The following options may be specified anywhere on the command line before the file list.

Option	Description
no option	Import the list of table archive files from the specified folder into the Windows Installer database.

specified	
/c	Create a Windows Installer database and then import the list of table archive files from the specified folder into the new database.

For more information, see [Windows Installer Scripting Examples](#) for additional scripting examples. For sample utilities that do not require Windows Script Host, see [Windows Installer Development Tools](#).

Note that [A Localization Example](#) also demonstrates [Importing Localized Error and ActionText Tables](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Export Files

The VBScript file `WiExport.vbs` is provided in the [Windows SDK Components for Windows Installer Developers](#). This sample shows how to write script to export tables into a Windows Installer database. The script sample connects to an **Installer** object, opens a database and exports tables to archive files.

The sample demonstrates the use of:

- **OpenDatabase Method (Installer Object)**
- **LastErrorMessage** method of the **Installer object**
- **Export method**
- **OpenView method** of the **Database object**
- **Fetch method** of the **View object**
- **StringData property** of the **Record object**

You'll require the `CScript.exe` or `WScript.exe` version of Windows Script Host to use this sample. To use `CScript.exe` to run this sample, type a command line at the command prompt using the following syntax. Help is displayed if the first argument is `/?` or if too few arguments are specified. To redirect the output to a file, end the command line with `VBS > [path to file]`. The sample returns a value of 0 for success, 1 if help is invoked, and 2 if the script fails.

**`cscript WiExport.vbs [path to database][path to folder][options] [table name list]`**

Specify the path to the installer database from which the tables are being exported. Specify the path to the folder into which the exported archive files are to be copied. List the case-sensitive names of the [database tables](#) being exported. Specify `*` to export all the tables including `_SummaryInformation`.

The following options may be specified anywhere on the command line before the table name list.

Option	Description

no option specified	Exported archive files may have a long file name.
/s	Force exported archive files to have short file names.

For additional scripting examples, see [Windows Installer Scripting Examples](#). For sample utilities that do not require Windows Script Host, see [Windows Installer Development Tools](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Manage Substorages

The VBScript file `WiSubStg.vbs` is provided in the [Windows SDK Components for Windows Installer Developers](#). This sample shows how script can be used to manage substorages in a Windows Installer database. A transform can be added to an existing Windows Installer database as a substorage.

The sample demonstrates the use of:

- [\\_Storages table](#)
- [OpenDatabase method \(Installer Object\)](#)
- [CreateRecord method](#)
- [LastErrorRecord method](#) of the [Installer object](#)
- [OpenView method](#)
- [Commit method](#) of the [Database object](#)
- [Fetch method](#)
- [Modify method](#)
- [Execute method](#) of the [View object](#)
- [StringData property](#)
- [SetStream method](#) of the [Record object](#)

You'll require the `CScript.exe` or `WScript.exe` version of Windows Script Host to use this sample. To use `CScript.exe` to run this sample, type a command line at the command prompt using the following syntax. Help is displayed if the first argument is `/?` or if too few arguments are specified. To redirect the output to a file, end the command line with `VBS > [path to file]`. The sample returns a value of 0 for success, 1 if help is invoked, and 2 if the script fails.

```
cscript WiSubStg.vbs [path to database][path to file][options]  
[substorage name]
```

Specify the path to the Windows Installer database to add or remove substorage. Specify a path to the transform or database file that is being added as substorage. To list the substorages in the Windows Installer

database, omit the path to this file. You may specify an optional substorage name, if this is omitted the substorage name defaults to the file name.

The following option may be specified.

<b>Option</b>	<b>Description</b>
<i>no option specified</i>	Add a substorage to the Windows Installer database.
/d	Remove a substorage. This option flag must be followed by the name of the substorage to be removed.

For additional scripting examples, see [Windows Installer Scripting Examples](#). For sample utilities that do not require Windows Script Host, see [Windows Installer Development Tools](#).

Note that [A Localization Example](#) demonstrates [Embedding Customization Transforms as Substorage](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Manage Binary Streams

The VBScript file `WiStream.vbs` is provided in the [Windows SDK Components for Windows Installer Developers](#). This sample shows how script can be used to manage binary streams in a Windows Installer database. The sample may be used to enter compressed file cabinets into a database. This sample demonstrates the operation of the [\\_Streams table](#) in the Windows Installer database.

The sample also demonstrates the use of:

- **OpenDatabase method (Installer Object)**
- **CreateRecord method**
- **LastErrorRecord method** of the **Installer object**
- **OpenView method**
- **Commit method** of the **Database object**
- **Fetch method**
- **Modify method**
- **Execute method** of the **View object**
- **StringData property**
- **SetStream method** of the **Record object**

You'll require the `CScript.exe` or `WScript.exe` version of Windows Script Host to use this sample. To use `CScript.exe` to run this sample, type a command line at the command prompt using the following syntax. Help is displayed if the first argument is `/?` or if too few arguments are specified. To redirect the output to a file, end the command line with `VBS > [path to file]`. The sample returns a value of 0 for success, 1 if help is invoked, and 2 if the script fails.

```
cscript WiStream.vbs [path to database][path to file][options]  
[stream name]
```

Specify the path to the Windows Installer database that is to receive the stream. Specify a path to the binary file containing the stream data. To list the streams in the installer database, omit this path. You may specify an optional stream name, if this is omitted it defaults to the file name.

The following option may be specified.

<b>Option</b>	<b>Description</b>
<i>no option specified</i>	Add a stream to the Windows Installer database.
/d	Remove a stream. This option flag must be followed by the name of the substorage being removed.

For additional scripting examples, see [Windows Installer Scripting Examples](#). For sample utilities that do not require Windows Script Host, see [Windows Installer Development Tools](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merge Two Databases

The VBScript file `WiMerge.vbs` is provided in the [Windows SDK Components for Windows Installer Developers](#). This sample script merges one Windows Installer database into another database. For more information, see [Merges and Transforms](#).

The [MsiDatabaseMerge](#) function and the [Merge](#) method of the [Database](#) object cannot be used to merge a module included in the installation package. They should not be used to merge [Merge Modules](#) into a Windows Installer package. To include a merge module in an installation package, authors of installation packages should follow the guidelines that are described in [Applying Merge Modules](#) topic.

The sample demonstrates the use of the following:

- [OpenDatabase method \(Installer Object\)](#)
- [LastErrorRecord method](#) of the [Installer object](#)
- [OpenView method](#)
- [Merge method](#)
- [Commit method](#) of the [Database object](#)
- [Fetch method](#)
- [View object](#)
- [StringData property](#) of the [Record object](#)

You must have the `CScript.exe` or `WScript.exe` version of Windows Script Host to use this sample. To use `CScript.exe` to run this sample, type a command line at the command prompt using the following syntax. Help is displayed if the first argument is `/?` or if too few arguments are specified. To redirect the output to a file, end the command line with `VBS > [path to file]`. The sample returns a value of 0 for success, 1 if help is invoked, and 2 if the script fails.

```
cscript WiMerge.vbs [path to database][path to imported database]  
[table name]
```

Specify the path to the Windows Installer database that is receiving the merge. Specify the path to the database being imported into the first. You

may specify an optional name for a table to hold the merge errors. If no table name is specified, the installer uses the name `_MergeErrors` and drops the table after displaying the contents.

For additional scripting examples, see [Windows Installer Scripting Examples](#). For sample utilities that do not require Windows Script Host, see [Windows Installer Development Tools](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Generate a Transform

The VBScript file `WiGenXfm.vbs` is provided in the [Windows SDK Components for Windows Installer Developers](#). This sample script can generate a transform from two Windows Installer databases. For more information see [Database Transforms](#).

The sample demonstrates the use of:

**[OpenDatabase method \(Installer Object\)](#)**

**[LastErrorRecord method](#)** of the **[Installer object](#)**

**[GenerateTransform method](#)** of the **[Database object](#)**

You'll require the `CScript.exe` or `WScript.exe` version of Windows Script Host to use this sample. To use `CScript.exe` to run this sample, type a command line at the command prompt using the following syntax. Help is displayed if the first argument is `/?` or if too few arguments are specified. To redirect the output to a file, end the command line with `VBS > [path to file]`. The sample returns a value of 0 for success, 1 if help is invoked, and 2 if the script fails.

**`cscript WiGenXfm.vbs [path to original database][path to revised database][path to transform file]`**

Specify the path to the original Windows Installer database. Specify the path to the revised database. Specify the path to the transform file to be created. If the path to the transform file is omitted, the two databases are only compared.

For additional scripting examples, see [Windows Installer Scripting Examples](#). For sample utilities that do not require Windows Script Host, see [Windows Installer Development Tools](#).

Note that [A Localization Example](#) demonstrates [Generating a Customization Transform](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Apply a Transform

The VBScript file `WiUseXfm.vbs` is provided in the [Windows SDK Components for Windows Installer Developers](#). This sample shows how script can be used to apply a transform to a Windows Installer database.

The sample demonstrates the use of

- **OpenDatabase method (Installer Object)**
- **LastErrorRecord method** of the **Installer object**
- **ApplyTransform method**
- **Commit method** of the **Database object**

You'll require the `CScript.exe` or `WScript.exe` version of Windows Script Host to use this sample. To use `CScript.exe` to run this sample, type a command line at the command prompt using the following syntax. Help is displayed if the first argument is `/?` or if too few arguments are specified. To redirect the output to a file, end the command line with `VBS > [path to file]`. The sample returns a value of 0 for success, 1 if help is invoked, and 2 if the script fails.

**`cscript WiUseXfm.vbs [path to original database][path to transform file][options]`**

Specify the path to the Windows Installer database. Specify the path to the transform file. If the path to the transform file is omitted, the two databases are only compared. The third argument is an optional numeric value that specifies a set of error conditions that are to be suppressed. Add these values together to suppress multiple conditions.

Value	Error condition to suppress
1	Adding a row that already exists.
2	Deleting a row that does not exist.
4	Adding a table that already exists.
8	Deleting a table that does not exist.
16	Updating a row that does not exist.

For additional scripting examples, see [Windows Installer Scripting Examples](#). For sample utilities that do not require Windows Script Host, see [Windows Installer Development Tools](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# View a Transform

The VBScript file `WiLstXfm.vbs` is provided in the [Windows SDK Components for Windows Installer Developers](#). This script sample can be used to view a transform file.

The sample demonstrates the use of:

- `_TransformView` table
- **OpenDatabase method (Installer Object)**
- **LastErrorRecord method** of the **Installer object**
- **ApplyTransform method**
- **OpenView method**
- **Commit method** of the **Database object**
- **IsNull property**
- **StringData property** of the **Record object**

Using this sample requires the `CScript.exe` version of Windows Script Host. To use `CScript.exe` to run this sample, type a command at the command prompt using the following syntax. Help is displayed if the first argument is `/?` or if too few arguments are specified. To redirect the output to a file, end the command line with `VBS > [path to file]`. The sample returns a value of 0 for success, 1 if help is invoked, and 2 if the script fails.

**cscript `WiLstXfm.vbs`** *[path to reference database][option][path to transform to be viewed]*

Specify the path to the reference Windows Installer database. Specify a list of paths to transform files that are being viewed. Each path in the list may be preceded by an optional numeric value. This value specifies a set of error conditions that are to be suppressed. You may add these values together to suppress multiple conditions. If no numeric option is specified, all the error conditions are suppressed. The arguments in this list are executed in the left-to-right order in which they appear on the command line.

<b>Value</b>	<b>Error condition to suppress</b>
--------------	------------------------------------

1	Adding a row that already exists.
2	Deleting a row that does not exist.
4	Adding a table that already exists.
8	Deleting a table that does not exist.
16	Updating a row that does not exist.
256	Mismatch of database and transform codepages.

For additional scripting examples, see [Windows Installer Scripting Examples](#). For sample utilities that do not require Windows Script Host, see [Windows Installer Development Tools](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# View Differences Between Two Databases

The VBScript file `WiDiffDb.vbs` is provided in the [Windows SDK Components for Windows Installer Developers](#). This sample script generates a temporary transform file between two Windows Installer databases and displays the transform.

The sample demonstrates the use of:

- **OpenDatabase method (Installer Object)**
- **LastErrorRecord method** of the **Installer object**
- **OpenView method**
- **SummaryInformation property (Database Object)**
- **GenerateTransform method**
- **ApplyTransform method**
- **Database object**
- **Fetch method** of the **View object**
- **IsNull property**
- **StringData property** of the **Record object**
- **\_TransformView table**

Using this sample requires the `CScript.exe` version of Windows Script Host. To use `CScript.exe` to run this sample, type a command at the command prompt using the following syntax. Help is displayed if the first argument is `/?` or if too few arguments are specified. To redirect the output to a file, end the command line with `VBS > [path to file]`. The sample returns a value of 0 for success, 1 if help is invoked, and 2 if the script fails.

**cscript WiDiffDb.vbs** *[path to original database][path to revised database]*

Specify the path to the original Windows Installer database. Specify the path to the revised database. The sample script will display the

transform.

For additional scripting examples, see [Windows Installer Scripting Examples](#). For sample utilities that do not require Windows Script Host, see [Windows Installer Development Tools](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# View Installer Script

The VBScript file `WiLstScr.vbs` is provided in the [Windows SDK Components for Windows Installer Developers](#). This sample script demonstrates the Windows Installer script viewer.

The sample demonstrates the use of:

- [OpenDatabase method \(Installer Object\)](#)
- [LastErrorRecord](#) method of the [Installer](#) object
- [OpenView](#) method of the [Database](#) object
- [Fetch](#) method of the [View](#) object
- [FormatText](#) method
- [FieldCount](#) property
- [StringData](#) property of the [Record](#) object
- [\\_TransformView](#) table

Using this sample requires the `CScript.exe` version of Windows Script Host. To use `CScript.exe` to run this sample, type a command at the command prompt using the following syntax. Help is displayed if the first argument is `/?` or if too few arguments are specified. To redirect the output to a file, end the command line with `VBS > [path to file]`. The sample returns a value of 0 for success, 1 if help is invoked, and 2 if the script fails.

**cscript `WiLstScr.vbs`** *[path to Windows Installer execution script]*

Specify the path to the installer execution script.

For additional scripting examples, see [Windows Installer Scripting Examples](#). For sample utilities that do not require Windows Script Host, see [Windows Installer Development Tools](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Manage Summary Information

The VBScript file `WiSumInf.vbs` is provided in the [Windows SDK Components for Windows Installer Developers](#). This sample script can be used to manage the [summary information stream](#) of a Windows Installer package.

The sample demonstrates the use of:

- [SummaryInformation property \(Installer Object\)](#)
- [LastErrorRecord method](#) of the [Installer object](#)

Using this sample requires the `CScript.exe` or `WScript.exe` version of Windows Script Host. To use `CScript.exe` to run this sample, type a command at the command prompt using the following syntax. Help is displayed if the first argument is `/?` or if too few arguments are specified. To redirect the output to a file, end the command line with `VBS > [path to file]`. The sample returns a value of 0 for success, 1 if help is invoked, and 2 if the script fails.

**`cscript WiSumInf.vbs [path to database][Property=value]`**

Specify the path to the Windows Installer database. If no other arguments are specified, the script lists all the summary properties for the package. Specify a list of summary properties and values to be updated using the format `Property=value`. You may specify the property by either the name or PID value shown below. Date and time fields use current locale format, or "Now" or "Date". For more information, see [Summary Information Stream Property Set](#).

PID	Name	Description
1	Codepage	ANSI codepage of text strings in summary information. For more information, see <a href="#">Codepage Summary Property</a> .
2	Title	Type of Windows Installer package. For more information, see <a href="#">Title Summary Property</a> .
3	Subject	Full Product name . For more information, see <a href="#">Subject Summary Property</a> .
4	Author	Creator, typically vendor name. For more information,

		see <a href="#">Author Summary Property</a> .
5	Keywords	List of keywords for use by file browser. For more information, see <a href="#">Keywords Summary Property</a> .
6	Comments	Description of purpose or use of package. For more information, see <a href="#">Comments Summary Property</a> .
7	Template	Supported platforms and languages. For more information, see <a href="#">Template Summary Property</a> .
8	LastAuthor	Set only by the installer. For more information, see <a href="#">Last Saved By Summary Property</a> .
9	Revision	Package code GUID. For more information, see <a href="#">Revision Number Summary Property</a> .
11	Printed	Date and time of installation image. For more information, see <a href="#">Last Printed Summary Property</a> .
12	Created	Date and time of package creation. For more information, see <a href="#">Create Time/Date Summary Property</a> .
13	Saved	Date and time of last package modification. For more information, see <a href="#">Last Saved Time/Date Summary Property</a> .
14	Pages	Minimum version of Windows Installer required for this package. For more information, see <a href="#">Page Count Summary Property</a> .
15	Words	Type of source file image. For more information, see <a href="#">Word Count Summary Property</a> . Starting with Windows Installer version 4.0 on Windows Vista and Windows Server 2008, this property includes bits to specify whether elevated privileges are required.
16	Characters	Used for transforms only. <a href="#">Character Count</a> .
18	Application	Application associated with this file, i.e. "Windows Installer". For more information, see <a href="#">Creating Application Summary Property</a> .
19	Security	Security setting. For more information, see <a href="#">Security</a>

	<b>Summary Property.</b>
--	--------------------------

For additional scripting examples, see [Windows Installer Scripting Examples](#). For sample utilities that do not require Windows Script Host, see [Windows Installer Development Tools](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Manage Policy Settings

The VBScript file `WiPolicy.vbs` is provided in the [Windows SDK Components for Windows Installer Developers](#). This sample shows how script can be used to manage [system policy](#). Policy can be configured by an administrator using the Group Policy Editor (GPE) on Windows 2000.

This sample demonstrates the Windows Installer policies.

Using this sample requires the `CScript.exe` or `WScript.exe` version of Windows Script Host. To use `CScript.exe` to run this sample, type a command at the command prompt using the following syntax. Help is displayed if the first argument is `/?` or if too few arguments are specified. To redirect the output to a file, end the command line with `VBS > [path to file]`. The sample returns a value of 0 for success, 1 if help is invoked, and 2 if the script fails.

## **cscript WiPolicy.vbs [policy][value]**

If no arguments are specified on the command line, the sample returns the current policy settings. Specify the policy to be set by using the following identifier codes. Specify the new value for the policy. To return the current value of a policy, specify an empty string `""` for the value.

<b>CODE</b>	<b>Description</b>
LM	Logging modes. For more information, see <a href="#">Logging</a> .
DO	Debug modes. For more information, see <a href="#">Debug</a> .
DI	Disable Windows Installer mode. For more information, see <a href="#">DisableMsi</a> .
WT	Wait timeout in seconds in case of no activity. <b>HKLM\Software\Policies\Microsoft\Windows\Installer\Timeout</b>
DB	Disable user browsing of source locations. For more information, see <a href="#">DisableBrowse</a> .
DP	Disable patching. For more information, see <a href="#">DisablePatch</a> .
UC	Public properties sent to install service. For more information, see <a href="#">EnableUserControl</a> .
SS	Installer safe for scripting from browser. For more information, see

	<a href="#">SafeForScripting</a> .
EM	System privileges (HKLM). For more information, see <a href="#">AlwaysInstallElevated</a> .
EU	System privileges (HKCU). For more information, see <a href="#">AlwaysInstallElevated</a> .
DR	Disable rollback policy. For more information, see <a href="#">DisableRollback</a> .
TS	Locate transforms at root of source image. For more information, see <a href="#">TransformsAtSource</a> policy.
TP	Pin secure transforms in client-side-cache. For more information, see <a href="#">TransformsSecure</a> policy.
SO	Search order of source types. For more information, see <a href="#">SearchOrder</a> .

For additional scripting examples, see [Windows Installer Scripting Examples](#). For sample utilities that do not require Windows Script Host, see [Windows Installer Development Tools](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Manage Language and Codepage

The VBScript file `WiLangID.vbs` is provided in the [Windows SDK Components for Windows Installer Developers](#). This sample shows how script is used to access a package's language information and codepage. For more information, see [Localizing a Windows Installer Package and Code Page Handling \(Windows Installer\)](#).

The sample demonstrates the use of:

- [OpenDatabase method \(Installer Object\)](#)
- [CreateRecord method](#)
- [LastErrorRecord method](#) of the [Installer object](#)
- [OpenView method](#)
- [SummaryInformation property \(Database Object\)](#) of the [Database object](#)

Using this sample requires the `CScript.exe` or `WScript.exe` version of Windows Script Host. To use `CScript.exe` to run this sample, type a command at the command prompt using the following syntax. Help is displayed if the first argument is `/?` or if too few arguments are specified. To redirect the output to a file, end the command line with `VBS > [path to file]`. The sample returns a value of 0 for success, 1 if help is invoked, and 2 if the script fails.

**`cscript WiLangID.vbs [path to database][keyword][value]`**

Specify the path to the Windows Installer database. To change a value specify one of the following keywords followed by the new value. If no value is specified the sample returns the current value.

Keyword	Description
<b>Package</b>	The language versions supported by the database. For information, see <a href="#">Template Summary</a> property.
<b>Product</b>	Language the installer should use for any strings in the user interface that are not authored into the database. For information, see <a href="#">ProductLanguage</a> property.

<b>Codepage</b>	Single ANSI code page for the string pool. For information, see <a href="#">Code Page Handling (Windows Installer)</a> .
-----------------	--

For additional scripting examples, see [Windows Installer Scripting Examples](#). For sample utilities that do not require Windows Script Host see [Windows Installer Development Tools](#).

For more information, see [Determining an Installation Database's Code Page](#) and [Setting the Code Page of a Database](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Copy a Unicode File to an ANSI File

The VBScript file `WiToAnsi.vbs` is provided in the [Windows SDK Components for Windows Installer Developers](#). This sample shows how script is used to rewrite a Unicode text file as an ANSI text file.

Using this sample requires the `CScript.exe` or `WScript.exe` version of Windows Script Host. To use `CScript.exe` to run this sample, type a command line at the command prompt using the following syntax. Help is displayed if the first argument is `/?` or if too few arguments are specified. To redirect the output to a file, end the command line with `VBS > [path to file]`. The sample returns a value of 0 for success, 1 if help is invoked, and 2 if the script fails.

**`cscript WiToAnsi.vbs [path to Unicode file][path to ANSI file]`**

Specify the Unicode text file that is being converted. Specify the ANSI text file that is being created. If no ANSI file is specified, the Unicode file is replaced.

For additional scripting examples, see [Windows Installer Scripting Examples](#). For sample utilities that do not require Windows Script Host see [Windows Installer Development Tools](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Manage File Sizes and Versions

The VBScript file `WiFilVer.vbs` is provided in the [Windows SDK Components for Windows Installer Developers](#). The sample shows you how you can use a script to report or update the file version, size, and language information.

The sample also shows you Windows Installer actions, how to access a Windows Installer database, and the use of the following:

- [Installer.OpenDatabase](#) method of the [Installer Object](#)
- [Installer.FileAttributes](#) property
- [Installer.FileHash](#) method
- [Installer.FileVersion](#) method
- [Installer.LastErrorRecord](#) method of the [Installer Object](#)
- [Database.OpenView](#) method
- [Database.SummaryInformation](#) property of the [Database Object](#)
- [Session.DoAction](#) method
- [Session.Property](#)
- [Session.SourcePath](#) property
- [Session.Mode](#) property of the [Session Object](#)
- [Record.StringData](#) property
- [Record.IntegerData](#) property of the [Record Object](#)

Using this sample requires the `CScript.exe` or `WScript.exe` version of Windows Script Host. To use `CScript.exe` to run this sample, type a command at the command prompt by using the following syntax:

```
cscript WiFilVer.vbs [path to database][optional source locations]
```

Also be aware of the following:

- Help is displayed if the first argument is `/?` or if too few arguments are specified.
- To redirect the output to a file, end the command line with `VBS >`

[*path to file*].

- The sample returns a value of 0 (zero) for success, 1 (one) if help is invoked, and 2 (two) if the script fails.

Specify the Windows Installer database that you want to be updated, which must be located at the source file root. However, you can specify sources for the database at separate locations. If the source is compressed, all the files are opened at the root.

The following options can be specified at any location on the command line.

<b>Option</b>	<b>Description</b>
<i>no option specified</i>	Display the file information of the database.
/u	Update the file size, version, and language information in the database from the source.

For more information, see [Windows Installer Scripting Examples](#) and [Windows Installer Development Tools](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Generate File Cabinet

The VBScript file `WiMakCab.vbs` is provided in the [Windows SDK Components for Windows Installer Developers](#). This sample shows how script is used to generate file cabinets from a Windows Installer database.

This sample demonstrates:

- **OpenDatabase method (Installer Object)** and the **LastErrorRecord method** of the **Installer Object**
- **Commit method**, the **OpenView method** and **SummaryInformation property (Database Object)** of the **Database Object**
- **Fetch method**, **Execute method** and **Modify method** of the **View Object**
- **StringData property** and **IntegerData property** of the **Record Object**
- **DoAction method**, the **Property property (Session Object)**, and the **Mode property** of the **Session Object**

You'll require the `CScript.exe` or `WScript.exe` version of Windows Script Host to use this sample. To use `CScript.exe` to run this sample, type a command at the command prompt using the following syntax. Help is displayed if the first argument is `/?` or if too few arguments are specified. To redirect the output to a file, end the command line with `VBS > [path to file]`. The sample returns a value of 0 for success, 1 if help is invoked, and 2 if the script fails.

**`cscript WiMakCab.vbs [path to database][base name][optional source locations]`**

In order to generate a cabinet, `Makecab.exe` must be on the `PATH`. The `Makecab.exe` utility is included in the [Windows SDK Components for Windows Installer Developers](#). Note that the sample does not update the [Media table](#) to handle multiple cabinets. To replace an embedded cabinet, include the options: `/R /C /U /E`.

Specify the path to the installer database. This must be located at the root of the source tree. Specify the case-sensitive base name for the generated cabinet files. If the source type is compressed, all files are opened at the root. The following options may be specified at any point on the command line.

Option	Description
<i>no option specified</i>	
/C	Run compression. If /C is not specified, WiMakCab.vbs only generates the DDF file.
/L	Use LZX compression instead of MSZIP
/F	Limit cabinet size to 1.44 MB floppy size rather than CD-ROM
/U	Update the database to reference the generated cabinet
/E	Embed the cabinet file in the installer package as a stream
/S	Use sequence numbers in the File table ordered by directories
/R	Revert to non-cabinet install, remove cabinet if /E is specified (The /R option removes the compressed bit - SummaryInfo property 15 & 2)

For additional scripting examples, see [Windows Installer Scripting Examples](#). For sample utilities that do not require Windows Script Host, see [Windows Installer Development Tools](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Execute SQL Statements

The VBScript file `WiRunSQL.vbs` is provided in the [Windows SDK Components for Windows Installer Developers](#). This sample shows how script is used to run SQL queries and updates against a Windows Installer database. For more information, see [SQL Syntax](#) and [Examples of Database Queries Using SQL and Script](#).

The sample script demonstrates:

- **OpenDatabase method (Installer Object)** and the **LastErrorMessage method** of the **Installer Object**
- **OpenView method**, and the **Commit method** of the **Database Object**
- **Execute method** of the **View Object**
- **StringData property** property of the **Record Object**

Using this sample requires the `CScript.exe` or `WScript.exe` version of Windows Script Host. To use `CScript.exe` to run this sample, type a command at the command prompt using the following syntax. Help is displayed if the first argument is `/?` or if too few arguments are specified. To redirect the output to a file, end the command line with `VBS > [path to file]`. The sample returns a value of 0 for success, 1 if help is invoked, and 2 if the script fails.

## **cscript WiRunSQL.vbs [path to database][SQL queries]**

Specify the path to a Windows Installer database. Specify the SQL queries that are to be executed. Note that the SQL query must be enclosed in double quotes. `SELECT` queries display the rows of the result list specified in the query. Binary data columns selected by a query are not displayed.

For additional scripting examples, see [Windows Installer Scripting Examples](#). For sample utilities that do not require Windows Script Host, see [Windows Installer Development Tools](#).

For more information, see [Working with Queries](#) and [Examples of Database Queries Using SQL and Script](#). The sample `A Localization`

[Example](#) demonstrates using SQL for [Localizing Database Columns](#) and [Updating a Summary Information Stream](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Copy ANSI File Into a Database Field

The VBScript code sample file `WiTextIn.vbs` is provided in the [Windows SDK Components for Windows Installer Developers](#). The sample shows how a script can be used to copy a file into a text field of a Windows Installer database, and demonstrates the processing of primary key data.

The code sample also shows you the following:

- **OpenDatabase method (Installer Object)** and the **LastErrorMessage method** of the **Installer Object**
- **OpenView method**, the **Commit method**, and the **PrimaryKeys property** of the **Database Object**
- **Fetch method** and the **Modify method** of the **View Object**
- **StringData property** and **ReadStream method** of the **Record Object**

To use the code sample you need the `CScript.exe` or `WScript.exe` version of Windows Script Host.

## ▶ To use `CScript.exe` to run this sample

- At the command prompt, type the following syntax:  
**`cscript WiTextIn.vbs [path to database][table name][primary key values][column name][path to file]`**

**Note** Help is displayed if the first argument is `/?` or if too few arguments are specified.

## ▶ To redirect the output to a file

- End the command line with the following: **`VBS > [path to file]. T`**

**Note** The sample returns a value of 0 (zero) for success, 1 (one) if Help is invoked, and 2 (two) if the script fails.

The following list identifies the items that you must specify:

- Specify the path to the Windows Installer database.

- Specify the name of the database table.
- Specify all the primary key values for the row, in order, and concatenated with colons.
- Specify a column name that is not a key column. This is the column that you want to receive the data.
- Specify the path to the text file that is being copied.

**Note** If the last argument is omitted, the current value in the field is displayed.

For more scripting examples, see [Windows Installer Scripting Examples](#). For sample utilities that do not require the Windows Script Host, see [Windows Installer Development Tools](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# List Components

The VBScript file `WiCompon.vbs` is provided in the [Windows SDK Components for Windows Installer Developers](#). This sample script can be used to list the components in a Windows Installer database.

This sample demonstrates using the various primary key in the [Component table](#).

The sample also demonstrates:

- [OpenDatabase method \(Installer Object\)](#), the [CreateRecord method](#), and the [LastErrorRecord method](#) of the [Installer Object](#).
- [OpenView method](#), the [TablePersistent property](#), and the [PrimaryKeys property](#) of the [Database Object](#).
- [Execute method](#) and the [Fetch method](#) of the [View Object](#).
- [StringData property](#) property of the [Record Object](#).

Using this sample requires the `CScript.exe` or `WScript.exe` version of Windows Script Host. To use `CScript.exe` to run this sample, type a command at the command prompt using the following syntax. Help is displayed if the first argument is `/?` or if too few arguments are specified. To redirect the output to a file, end the command line with `VBS > [path to file]`. The sample returns a value of 0 for success, 1 if help is invoked, and 2 if the script fails.

**`cscript WiCompon.vbs [path to database][component name]`**

Specify path to the Windows Installer database. Specify the name of the component. The name must be listed in the Component column of the [Component table](#). If the name of the component is omitted all the components are listed. If an asterisk (\*) is used as the component name, `WiCompon.vbs` lists the composition of all components. Note that large databases are better displayed using `CScript` rather than `WScript`.

For additional scripting examples, see [Windows Installer Scripting Examples](#). For sample utilities that do not require Windows Script Host, see [Windows Installer Development Tools](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# List Features

The VBScript file `WiFeatur.vbs` is provided in the [Windows SDK Components for Windows Installer Developers](#). This sample shows how script is used to list the features in a Windows Installer database. This sample demonstrates adding temporary columns to a read-only Windows Installer database.

This sample demonstrates:

- [OpenDatabase method \(Installer Object\)](#), the [CreateRecord method](#), and the [LastErrorRecord method](#) of the [Installer Object](#)
- [Execute method](#) and the [Fetch method](#) of the [View Object](#)
- [OpenView method](#), the [TablePersistent property](#), and the [PrimaryKeys property](#) of the [Database Object](#)
- [FieldCount property](#), [IntegerData property](#), and the [StringData property](#) of the [Record Object](#)

Using this sample requires the `CScript.exe` or `WScript.exe` version of Windows Script Host. To use `CScript.exe` to run this sample, type a command at the command prompt using the following syntax. Help is displayed if the first argument is `/?` or if too few arguments are specified. To redirect the output to a file, end the command line with `VBS > [path to file]`. The sample returns a value of 0 for success, 1 if help is invoked, and 2 if the script fails.

**`cscript WiFeatur.vbs [path to database][feature name]`**

Specify path to the Windows Installer database. Specify the name of the feature. This must be listed in the Feature column of the [Feature table](#). If the name of the feature is omitted, all the features are listed as a feature tree. If an asterisk (\*) is used as the feature name, `WiFeatur.vbs` lists the composition of all features. Note that large databases are better displayed using `CScript` rather than `WScript`.

For more information, see [Windows Installer Scripting Examples](#) for additional scripting examples. For sample utilities that do not require Windows Script Host see [Windows Installer Development Tools](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Preview User Interface

The VBScript file `WiDialog.vbs` is provided in the [Windows SDK Components for Windows Installer Developers](#). This sample shows how script is used to preview dialogs in a Windows Installer database.

This sample demonstrates:

- [OpenDatabase method \(Installer Object\)](#), the [CreateRecord method](#), and the [LastErrorRecord method](#) of the [Installer Object](#)
- [OpenView method](#) and the [EnableUIpreview method](#) of the [Database Object](#)
- [Execute method](#) and the [Fetch method](#) of the [View Object](#)
- [StringData property](#) property of the [Record Object](#)

This sample requires the `CScript.exe` or `WScript.exe` version of Windows Script Host. To use `CScript.exe` for this sample, type a command at the command prompt using the following syntax. Help is displayed if the first argument is `/?` or if too few arguments are specified. To redirect the output to a file, end the command line with `VBS > [path to file]`. The sample returns a value of 0 for success, 1 if help is invoked, and 2 if the script fails.

**cscript WiDialog.vbs** *[path to database][Dialog names]*

Specify the path to a Windows Installer database. Specify the name of a dialog. This name must be listed in the Dialog column of the [Dialog table](#). To view a billboard, append the dialog's name with the control's name from the [Control table](#) and append this to the name of the billboard in the [Billboard table](#). If no dialogs are specified, all dialogs in Dialog table are displayed sequentially.

For additional scripting examples, see [Windows Installer Scripting Examples](#). For sample utilities that do not require Windows Script Host, see [Windows Installer Development Tools](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Single Package Authoring Example

The sample PUASample.msi is an example of a dual-purpose Windows Installer 5.0 package that is capable of being installed in either the per-user or per-machine [installation context](#) on Windows Server 2008 R2 and Windows 7. This sample package follows the development guidelines described in [Single Package Authoring](#).

## Obtaining a copy of the sample

A copy of this sample and a Windows Installer database table editor, Orca.exe, are in the [Windows SDK Components for Windows Installer Developers](#). The sample and table editor are provided with the Windows Software Development Kit for Windows Server 2008 R2 and Windows 7 as the Windows Installer installation files PUASample1.msi and Orca.msi.

## System Requirements

The database editor, [Orca.exe](#), requires Windows Server 2008 R2 and earlier and Windows 7 and earlier. The dual-purpose package, PUASample1.msi, can be installed in either the per-machine or per-user installation context on Windows Server 2008 R2 and Windows 7. PUASample1.msi can be installed only in the per-machine context on Windows Server 2008 and earlier and Windows Vista and earlier. You can install the database editor to examine the contents of PUASample1.msi without installing the sample. To install the sample or editor packages, ensure that the [DisableMSI](#) policy is not set to a value that blocks application installations.

## Identifying a Dual-Purpose Package

Dual-purpose packages should initialize the value of the [MSIINSTALLPERUSER](#) property to 1. This identifies the package as capable of being installed in either the per-machine or per-user context on Windows Server 2008 R2 and Windows 7. Set the [MSIINSTALLPERUSER](#) property in the package only if it has been written following the development guidelines described in [Single Package](#)

[Authoring](#) and if you intend to provide users with the option to install the package in either the per-user or per-machine context. A dual-purpose package should also initialize the value of the **ALLUSERS** property to 2. This specifies per-user as the default installation context for the application. If the value of the **ALLUSERS** property is any value other than 2, Windows Installer ignores the **MSIINSTALLPERUSER** property.

Use a Windows Installer database editor, such as Orca.exe, to examine the contents of PUASample1.msi. The [Property](#) table in the sample package contains the following two entries.

Property Table (partial)

Property	Value
ALLUSERS	2
MSIINSTALLPERUSER	1

## Custom Dialog Box for Installation Context

The [user interface](#) of the sample package includes an example of a custom dialog box, VerifyReadyDialog, that enables users to select either the per-user or per-machine installation context at installation time. The [Dialog](#) table contains a record that describes the VerifyReadyDialog dialog box. The value entered in the Attributes field is 39 because this dialog box uses the msidbDialogAttributesVisible (1), msidbDialogAttributesModal (2), msidbDialogAttributesMinimize (4), and msidbDialogAttributesTrackDiskSpace (32) [dialog style bits](#). The title bar of the dialog box displays a title given by the value of the **ProductName** property.

Dialog Table (partial)

Dialog	HCentering	VCentering	Width	Height	Attributes	Ti
VerifyReadyDialog	50	50	480	280	39	[P

The [Control](#) table contains entries for the [controls](#) displayed by the VerifyReadyDialog dialog box. The dialog box displays [PushButton](#)

controls and a [Text](#) control. All the controls use the [msidbControlAttributesEnabled](#) (2) and [msidbControlAttributesVisible](#) (1) [control attributes](#). The [InstallPerMachine](#) control also uses the [ElevationShield](#) control attribute, [msidbControlAttributesElevationShield](#) (8388608.) This control attribute adds the [User Account Control](#) (UAC) elevation icon (shield icon) to the [InstallPerMachine](#) control and informs the user that UAC credentials are required to install the application in the per-machine context. The value in the Text field of the Control table is the text-style and text displayed by the control. See the description of the Text field in the Control table topic for more information about adding text to a control using predefined styles.

#### Control Table (partial)

<a href="#">Dialog_</a>	<a href="#">Control</a>	<a href="#">Type</a>	<a href="#">Attribute</a>	<a href="#">Text</a>
VerifyReadyDialog	Cancel	PushButton	3	{\Tahoma10}8
VerifyReadyDialog	Previous	PushButton	3	{\Tahoma10}<<&Previous
VerifyReadyDialog	Next	PushButton	3	{\Tahoma10}8>>
VerifyReadyDialog	Text2	Text	3	Are you ready complete your suspended installation?
VerifyReadyDialog	InstallPerUser	PushButton	3	{\Tahoma10}I Only for &Me
VerifyReadyDialog	InstallPerMachine	PushButton	8388611	{\Tahoma10}I for &Everyone
VerifyReadyDialog	Cancel	PushButton	3	{\Tahoma10}8

The [ControlEvent](#) table specifies the [ControlEvents](#), or actions, the installer performs when the user interacts with a control. When a user activates the [InstallPerUser](#) pushbutton, the user interface displays an [OutOfDisk](#) dialog box if the [OutOfDiskSpace](#) property is 1, sets the value of the [MSIINSTALLPERUSER](#) property to 1, sets the value of the

**ALLUSERS** property to 2, sets the **MSIFASTINSTALL** property to 1, and returns . Because the **MSIFASTINSTALL** property is set, no System Restore point is generated for the installation. When a user activates the InstallPerMachine pushbutton, the user interface displays an OutOfDisk dialog box if the **OutOfDiskSpace** property is 1, sets the value of the **ALLUSERS** property to 1, and returns.

ControlEvent Table (partial)

Dialog_	Control_	Event	Argument
VerifyReadyDialog	InstallPerUser	SpawnDialog	OutOfDisk
VerifyReadyDialog	InstallPerUser	EndDialog	Return
VerifyReadyDialog	InstallPerUser	[MSIINSTALLPERUSER]	1
VerifyReadyDialog	InstallPerUser	[ALLUSERS]	2
VerifyReadyDialog	InstallPerMachine	SpawnDialog	OutOfDisk
VerifyReadyDialog	InstallPerMachine	EndDialog	Return
VerifyReadyDialog	InstallPerMachine	[ALLUSERS]	1
VerifyReadyDialog	InstallPerUser	[MSIFASTINSTALL]	1

The InstallPerUser control should be removed from the user interface of any installation using a Windows Installer version earlier than Windows Installer Windows Installer 5.0. The **ControlCondition** table in the sample package contains four entries that disable and hide the InstallPerUser control if the current version is less than Windows Installer 5.0. The table uses the value of the **VersionMsi** property and the **conditional statement syntax** to define this condition. The action specified in the Action field is performed only if the statement in the Condition field is true.

ControlCondition Table (partial)

Dialog_	Control_	Action	Condition

VerifyReadyDialog	InstallPerUser	Enable	VersionMsi >= "5.00"
VerifyReadyDialog	InstallPerUser	Disable	VersionMsi < "5.00"
VerifyReadyDialog	InstallPerUser	Show	VersionMsi >= "5.00"
VerifyReadyDialog	InstallPerUser	Hide	VersionMsi < "5.00"

## Specifying Directory Structure

Use the database editor to examine the [Directory](#) table of PUASample1.msi. The record of the Directory Table having an empty string in its Directory\_Parent field represents the root directory of both the source and target directory trees. If the **TARGETDIR** property is undefined, the installer sets its value at installation time to the value of the **ROOTDRIVE** property. If the **SourceDir** property is undefined, the installer sets its value to the location of the directory containing the Windows Installer package (.msi file.) The directory names are specified using the short|long format.

Directory Table (partial)

Directory	Directory_Parent	DefaultDir
TARGETDIR		SourceDir
ProgramFilesFolder	TARGETDIR	.
ProgramMenuFolder	TARGETDIR	.
INSTALLLOCATION	MyVendor	Sample1 MSDN-PUASample1
MyVendor	ProgramFilesFolder	Msft Microsoft

At the source, this Directory table resolves to the following directory paths.

```
[SourceDir]\Msft\Sample1
[SourceDir]
```

At the target, the Directory table resolves to the paths in the following table. The installer sets the values of the **ProgramFilesFolder** and

**ProgramMenuFolder** properties to locations that depend upon the installation context and whether the system is the 32-bit or 64-bit versions of Windows Server 2008 R2 and Windows 7. The paths to the target folders depend on whether the user selects a per-user or per-machine installation.

<b>Installation</b>		
<b>Context</b>	<b>System</b>	<b>Example Paths</b>
Per-Machine	Windows Server 2008 R2 and Windows 7 32-bit version	%ProgramFiles%\Msft\Sample1 %ALLUSERSPROFILE%\Microsoft\Windows\Start Menu\Programs
Per-Machine	Windows Server 2008 R2 and Windows 7 64-bit version	%ProgramFiles(x86)%\Msft\Sample1 %ALLUSERSPROFILE%\Microsoft\Windows\Start Menu\Programs
Per-User	Windows Server 2008 R2 and Windows 7 32-bit or 64-bit version	%USERPROFILE%\AppData\Local\Programs\Microsoft\Windows\Start Menu\Programs %APPDATA%\Microsoft\Windows\Start Menu\Programs

Per-user applications should be stored in subfolders under the Programs folder specified by the value of **ProgramFilesFolder** property. Typically, the path to the application takes the following form.

%LOCALAPPDATA%\Programs\SV *name*\AppName.

Per-user configuration data should be stored in the Programs folder specified by the value of the **ProgramMenuFolder** property. Typically, this folder is located at the following path.

%APPDATA%\Microsoft\Windows\Start Menu\Programs

If installing [32-bit Windows Installer Package](#) components, use the **ProgramFilesFolder** and **CommonFilesFolder** property in the Directory

table. If installing [64-bit Windows Installer Package](#) components, use the [ProgramFiles64Folder](#) and [CommonFiles64Folder](#) properties. If your application contains 32-bit and 64-bit versions of the same component, with the same name, ensure that these versions are saved in different directories or give them different names.

The following Directory table provides an example of a directory layout compatible with a package that includes 32-bit and 64-bit components and includes some components that are shared across applications.

<b>Directory</b>	<b>Directory_Parent</b>	<b>DefaultDir</b>
TARGETDIR		SourceDir
ProgramFilesFolder	TARGETDIR	.:Prog32
ProgramFiles64Folder	TARGETDIR	.:Prog64
CommonFilesFolder	TARGETDIR	.:Share32
CommonFiles64Folder	TARGETDIR	.:Share64
ProgramMenuFolder	TARGETDIR	.:Sample1 MSDN-PUASample1
INSTALLLOCATION	MyVendor	Sample1 MSDN-PUASample1
INSTALLLOCATIONX64	Vendorx64	Sample1 MSDN-PUASample1
SHAREDLOCATION	ShVendor	Sample1 MSDN-PUASample1
SHAREDLOCATIONX64	ShVendorx64	Sample1 MSDN-PUASample1
MyVendor	ProgramFilesFolder	Msft Microsoft
Vendorx64	ProgramFiles64Folder	Msft Microsoft
ShVendor	CommonFilesFolder	Msft Microsoft
ShVendorx64	CommonFiles64Folder	Msft Microsoft
Shrx86	SHAREDLOCATION	x32 32-bit components

Shrx64	SHAREDLOCATIONX64	x64 64-bit components
Binx86	INSTALLLOCATION	x32 32-bit components
Binx64	INSTALLLOCATIONX64	x64 64-bit components
App32	Binx86	myapp unshared 32-bit components
App64	Binx64	myapp unshared 64-bit components
Share32	Shrx86	shared shared 32-bit components
Share64	Shrx64	shared shared 64-bit components

At the source, this Directory table resolves to the following directory paths.

[SourceDir]Prog32\Msft\Sample1\x32\myapp

[SourceDir]Share32\Common Files\Msft\Sample1\x32\shared

[SourceDir]Prog64\Msft\Sample1\x64\myapp

[SourceDir]Share64\Common Files\Msft\Sample1\x64\shared

[SourceDir]Sample1

At the target, this Directory table resolves to the following directory paths. The target paths depend on the installation context and system.

<b>Installation</b>		
<b>Context</b>	<b>System</b>	<b>Example Paths</b>
Per-Machine	Windows Server 2008 R2 and Windows 7 32-bit version	%ProgramFiles%\Msft\Sample1\x32\myapp %ProgramFiles%\Common Files\Msft\Sample1\ %ProgramFiles(x86)%\Msft\Sample1\x64\myapp %ProgramFiles(x86)%\Common Files\Msft\Sam

		%ProgramData%\Microsoft\Windows\Start Menu
Per-Machine	Windows Server 2008 R2 and Windows 7 64-bit version	%ProgramFiles(x86)%\Msft\Sample1\x32\myapp %ProgramFiles(x86)%\Common Files\Msft\Sam %ProgramFiles%\Msft\Sample1\x64\myapp %ProgramFiles%\Common Files\Msft\Sample1\ %ProgramData%\Microsoft\Windows\Start Menu
Per-User	Windows Server 2008 R2 and Windows 7 32-bit or 64-bit version	%LOCALAPPDATA%\Programs\Msft\Sample1\ %LOCALAPPDATA%\Programs\Common\Msft %LOCALAPPDATA%\Programs\Msft\Sample1\ %LOCALAPPDATA%\Programs\Common\Msft %APPDATA%\Microsoft\Windows\Start Menu\I

## Application Registration

The PUASample.msi adds a subkey to the App Paths registry key for the application and performs registrations that enable application information to be saved in the registry under this key. For more information about App Paths and application registration, see the PerceivedTypes, SystemFileAssociations, and Application Registration in the [shell extensibility](#) section of the Shell Developer's Guide. At installation time, the user makes the decision to install the application in either the per-user or per-machine installation context. At the time the dual-purpose package is authored, the package developer cannot know if the registrations should be performed under the HKEY\_LOCAL\_MACHINE or HKEY\_CURRENT\_USER keys.

The package developer defines the file identifier for the application's executable file in the File field of the [File](#) Table.

File Table (partial)

File	Component_	FileName	FileSize

MyAppFile	ProductComponent	PUASAMP1.EXE PUASample1.exe	81920
-----------	------------------	-----------------------------	-------

Values to be saved in the registry can be specified in the Value field of the [Registry](#) table as a [Formatted](#) string. Use the file identifier defined in the File field of the File table, and the `[#filekey]` convention of the Formatted type, to specify the default value for the App Paths registry key. The top-level [INSTALL](#) action performs the actions in the [InstallExecuteSequence](#) table. After the [CostInitialize](#), [FileCost](#), and [InstallFinalize](#) actions in this table have completed, the Windows Installer replaces the formatted substring `[#MyAppFile]` in the Registry table with the full path to the application file.

The sample defines a custom property, `RegRoot`, to contain the location of the root key and uses a custom action to reset the property value if the user chooses a per-machine installation. Use the custom property, `RegRoot`, in any formatted string values that reference the root location. In the Property table the `PUASample.msi` package defines the custom property and sets the value of `RegRoot` to `HKCU`. This initializes the value of the property for the per-user installation context, the recommended default context for dual-purpose packages.

Property Table (partial)

Property	Value
RegRoot	HKCU

In the [CustomAction](#) table the package defines a custom action named `Set_RegRoot_HKLM`. The value in the Type field identifies this as a [Custom Action Type 51](#) standard custom action. The meaning of the Source and Target fields in the CustomAction table depend upon the custom action type. For more information about the standard types of custom actions, see [Custom Action Types](#). The Source field for the `Set_RegRoot_HKLM` custom action specifies that the value of the `RegRoot` property. If the installer performs the `Set_RegRoot_HKLM` custom action, this resets the value of the `RegRoot` property to `HKLM`.

CustomAction Table (partial)

Action	Type	Source	Target
--------	------	--------	--------

Set_RegRoot_HKLM	51	[RegRoot]	HKLM
------------------	----	-----------	------

The top-level INSTALL action performs the actions in the InstallExecuteSequence table, in the sequence specified in the Sequence field of that table. The value authored in the Sequence field for the Set\_RegRoot\_HKLM custom action (1501) specifies that this custom action be performed after the [InstallInitialize](#) action (1500) and before the [ProcessComponents](#) action (1600.) This sequence ensures that the record for the Set\_RegRoot\_HKLM custom action is evaluated at installation time. For more information about the recommended sequence of actions in the InstallExecuteSequence table, see the [Suggested InstallExecuteSequence](#) topic. The conditional statement syntax authored in the Condition field specifies that the Set\_RegRoot\_HKLM action be performed only if the value of the **ALLUSERS** property evaluates to 1 at installation time. An **ALLUSERS** property value of 1 specifies a per-machine installation.

InstallExecuteSequence Table (partial)

Action	Condition	Sequence
Set_RegRoot_HKLM	ALLUSERS=1	1501

The following records in the Registry table perform the registrations if the ProductComponent component is installed. The value -1 in the Root field is required to perform the registration under HKEY\_LOCAL\_MACHINE for a per-user installation and under HKEY\_CURRENT\_USER for a per-user installation. The record with an empty string in the Registry field adds a subkey for the application under the AppPaths registry key and sets the "(Default)" value to the full path of the application's executable file. The MyAppPathAlias registration maps the executable file to an application alias and enables the application to be launched if the user types the alias "puapct" at a command line prompt. The MyAppPathRegistration registration maps the name of the executable file to the file's full path.

Registry	Root	Key

	-1	Software\Microsoft\MyAppPathRegistrationLo
MyAppPathAlias	-1	Software\Microsoft\Windows\CurrentVersion\A Paths\PUAPCT.exe
MyAppPathRegistration	-1	Software\Microsoft\Windows\CurrentVersion\A Paths\PUASample1.exe

## AutoPlay Cancel Registration

The PUASample.msi performs registrations that enable the application user to prevent Hardware Autoplay from launching for selected devices. For information about registering a handler to cancel Autoplay in response to an event, see the Preparing Hardware and Software for Use with AutoPlay topic in the [shell extensibility](#) section of the Shell Developer's Guide. The following record registers the handler specified in the Name field when the ProductComponent component is installed. The value -1 in the Root field is required to specify to the Windows Installer that the registration should be redirected to a location that depends upon the installation context.

### Registry Table

Registry	Root	Key
MyAutoplayCancelRegistration	-1	SOFTWARE\Microsoft\Windows\Curre

## Preview Handler Registration

The PUASample.msi performs registrations that are required to install a preview handler that enables a read-only preview of .pua files without launching the application. For information about registering preview handlers, see the Registering Preview Handlers topic in the [shell extensibility](#) section of the Shell Developer's Guide. The following records

in the Registry table register the handler when the ProductComponent component is installed. The value -1 in the Root field is required to specify to the Windows Installer that the registration should be redirected to a location that depends upon the installation context.

### Registry Table

<b>Registry</b>	<b>Root</b>	<b>Key</b>
MyPreviewHandlerRegistration1	-1	Software\Classes\.pua
MyPreviewHandlerRegistration2	-1	Software\Microsoft\Windows\Curren
MyPreviewHandlerRegistration3	-1	Software\Classes\puafile\ShellEx\{880d564250836f}
MyPreviewHandlerRegistration4	-1	Software\Classes\CLSID\{1531d583-d23bbd169f22}
MyPreviewHandlerRegistration5	-1	Software\Classes\CLSID\{1531d583-d23bbd169f22}
MyPreviewHandlerRegistration6	-1	Software\Classes\CLSID\{1531d583-d23bbd169f22}
MyPreviewHandlerRegistration7	-1	Software\Classes\CLSID\{1531d583-d23bbd169f22}
MyPreviewHandlerRegistration8	-1	Software\Classes\CLSID\{1531d583-d23bbd169f22}\InProcServer32
MyPreviewHandlerRegistration9	-1	Software\Classes\CLSID\{1531d583-d23bbd169f22}\InProcServer32
MyPreviewHandlerRegistration10	-1	Software\Classes\CLSID\{1531d583-d23bbd169f22}\InProcServer32

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Windows Installer Reference

The following sections contain reference information about the Windows Installer service and the Windows Installer API.

- [Automation Interface](#)
- [Installer Functions](#)
- [Installer Structures](#)
- [Installer Database](#)
- [Released Versions, Tools, and Redistributables](#)
- [Errors Reference](#)
- [Glossary](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Automation Interface

This section is intended for developers who are writing their own setup programs and who want to learn more about the Windows Installer database tables.

For information about automation and access to C++ Libraries, see [About the Automation Interface](#).

For information about using automation and creating the **installer object**, see [Using the Automation Interface](#).

For reference material for installer objects, see [Automation Interface Reference](#).

For information about WMI, see [Using Windows Installer with WMI](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# About the Automation Interface

An [Installer object](#) must be created initially to load the automation support required to access the installer components through COM. This object provides wrappers to create the top-level objects and access their methods. These wrappers simply provide parameter translations to expose the installer functions in a manner consistent with Microsoft Visual Basic without changing the behavior of the methods.

Whenever possible, a pair of Get and Set C++ methods are exposed to Visual Basic as a single property. In some cases C++ methods taking an index argument are exposed as an indexed property. Many C++ methods return the result through a parameter because the return value is used for the error return. However, in Visual Basic, errors are handled by a separate mechanism, and the result is always passed in the return value.

For information about using automation and creating the Installer object, see [Using the Automation Interface](#).

For reference material for the Windows Installer objects, see [Automation Interface Reference](#).

## See Also

[Windows Installer Scripting Examples](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using the Automation Interface

To access automation, the DLL must be self registered. The COM ProgId used to create the **installer object** is **WindowsInstaller.Installer**.

For information about automation and access to C++ Libraries, see [About the Automation Interface](#).

For reference material for the Windows Installer objects, see [Automation Interface Reference](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Automation Interface Reference

The following are the Windows Installer objects:

- [Client Object](#)
- [Component Object](#)
- [ComponentInfo Object](#)
- [Database Object](#)
- [FeatureInfo Object](#)
- [Installer Object](#)
- [Patch Object](#)
- [Product Object](#)
- [Record Object](#)
- [RecordList Object](#)
- [Session Object](#)
- [StringList Object](#)
- [SummaryInfo Object](#)
- [UIPreview Object](#)
- [View Object](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Client Object

The Client object represents a relationship between a component and client product.

**Windows Installer 4.5 or earlier:** Not supported. This object is available beginning with Windows Installer 5.0.

## Methods

The **Client** object does not define any methods.

## Properties

The **Client** object defines the following properties.

Property	Access type	Description
<b>ComponentCode</b>	Read-only	The component code of the component in question.
<b>Context</b>	Read-only	The context of the product.
<b>ProductCode</b>	Read-only	The client product of the component in question.
<b>UserSID</b>	Read-only	The user SID for the component.

## Requirements

<b>Version</b>	Windows Installer 5.0 or later.
<b>DLL</b>	Msi.dll
	IID_IClient is defined as 000C1098-0000-0000-

## See Also

[Using the Automation Interface](#)  
[Windows Installer Scripting Examples](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Client.ProductCode Property

The client product of the component in question.

**Windows Installer 4.5 or earlier:** Not supported. This property is available beginning with Windows Installer 5.0.

## Syntax

Script

## Requirements

<b>Version</b>	Windows Installer 5.0 or later.
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IClient is defined as 000C1098-0000-0000-C000-000000000046

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Client.ComponentCode Property

The component code of the component in question.

**Windows Installer 4.5 or earlier:** Not supported. This property is available beginning with Windows Installer 5.0.

## Syntax

Script

## Requirements

<b>Version</b>	Windows Installer 5.0 or later.
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IClient is defined as 000C1098-0000-0000-C000-000000000046

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Client.UserID Property

The user SID for the component.

**Windows Installer 4.5 or earlier:** Not supported. This property is available beginning with Windows Installer 5.0.

## Syntax

Script

## Requirements

<b>Version</b>	Windows Installer 5.0 or later.
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IClient is defined as 000C1098-0000-0000-C000-000000000046

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Client.Context Property

The context of the product.

**Windows Installer 4.5 or earlier:** Not supported. This property is available beginning with Windows Installer 5.0.

## Syntax

Script

## Requirements

<b>Version</b>	Windows Installer 5.0 or later.
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IClient is defined as 000C1098-0000-0000-C000-000000000046

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ComponentInfo Object

The ComponentInfo object represents additional details about a component that may be obtained via a call from MsiGetComponentPathEx.

**Windows Installer 4.5 or earlier:** Not supported. This object is available beginning with Windows Installer 5.0.

## Methods

The **ComponentInfo** object does not define any methods.

## Properties

The **ComponentInfo** object defines the following properties.

Property	Access type	Description
<b>ComponentCode</b>	Read-only	The component code of the component in question.
<b>Path</b>	Read-only	The path of the component.
<b>State</b>	Read-only	The state of the component.

## Requirements

<b>Version</b>	Windows Installer 5.0 or later.
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IComponentInfo is defined as 000C1099-0000-0000-C000-000000000046

## See Also

[Using the Automation Interface](#)  
[Windows Installer Scripting Examples](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ComponentInfo.ComponentCode Property

The component code of the component in question.

**Windows Installer 4.5 or earlier:** Not supported. This property is available beginning with Windows Installer 5.0.

## Syntax

Script

## Requirements

<b>Version</b>	Windows Installer 5.0 or later.
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IComponentInfo is defined as 000C1099-0000-0000-C000-000000000046

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ComponentInfo.Path Property

The path of the component.

**Windows Installer 4.5 or earlier:** Not supported. This property is available beginning with Windows Installer 5.0.

## Syntax

Script

## Requirements

<b>Version</b>	Windows Installer 5.0 or later.
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IComponentInfo is defined as 000C1099-0000-0000-C000-000000000046

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ComponentInfo.State Property

The state of the component.

**Windows Installer 4.5 or earlier:** Not supported. This property is available beginning with Windows Installer 5.0.

## Syntax

Script

## Requirements

<b>Version</b>	Windows Installer 5.0 or later.
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IComponentInfo is defined as 000C1099-0000-0000-C000-000000000046

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Component Object

The Component object represents a unique instance of a component that is available for enumeration.

**Windows Installer 4.5 or earlier:** Not supported. This object is available beginning with Windows Installer 5.0.

## Methods

The **Component** object does not define any methods.

## Properties

The **Component** object defines the following properties.

Property	Access type	Description
<b>ComponentCode</b>	Read-only	The component code of the component in question.
<b>Context</b>	Read-only	The context that was determined to be applicable to the component in question.
<b>UserSID</b>	Read-only	The user SID for the enumerated component.

## Requirements

<b>Version</b>	Windows Installer 5.0 or later.
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IComponent is defined as 000C1097-0000-0000-C000-000000000046

## See Also

[Using the Automation Interface](#)  
[Windows Installer Scripting Examples](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Component.ComponentCode Property

The component code of the component in question.

**Windows Installer 4.5 or earlier:** Not supported. This property is available beginning with Windows Installer 5.0.

## Syntax

Script

## Requirements

<b>Version</b>	Windows Installer 5.0 or later.
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IComponent is defined as 000C1097-0000-0000-C000-000000000046

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Component.UserID Property

The user SID for the enumerated component.

**Windows Installer 4.5 or earlier:** Not supported. This property is available beginning with Windows Installer 5.0.

## Syntax

Script

## Requirements

<b>Version</b>	Windows Installer 5.0 or later.
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IComponent is defined as 000C1097-0000-0000-C000-000000000046

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Component.Context Property

The context that was determined to be applicable to the component in question.

**Windows Installer 4.5 or earlier:** Not supported. This property is available beginning with Windows Installer 5.0.

## Syntax

Script

## Requirements

<b>Version</b>	Windows Installer 5.0 or later.
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IComponent is defined as 000C1097-0000-0000-C000-000000000046

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Database Object

The **Database** object accesses an installer database.

The **Database** object is released when it is either taken out of scope or when the object variable associated with it is set to null. The **Commit** method must be called before the **Database** object is released to write out all persistent changes. If the **Commit** method is not called, the installer performs an implicit rollback upon object destruction.

The client can use the following procedure for data access.

## ► To Query API Sequencing

1. Obtain a **Database** object by calling the **OpenDatabase** or the **Installer** object.
2. Initiate a query using a SQL string by calling the **OpenView** method of the **Database** object.
3. Set query parameters in a **Record** object and execute the database query by calling the **Execute** method of the **View** object. This produces a result that can be fetched or updated.
4. Call the **Fetch** method of the **View** object repeatedly to return **Record** objects.
5. Update database rows of a **Record** object obtained by the **Fetch** method using the **Modify** method of the **View** object.
6. Release the query and any unfetched records by calling the **Close** method of the **View** object.
7. Persist any database updates by calling the **Commit** method of the **Database** object.

## Methods

The **Database** object defines the following methods.

Method	Description

<b>ApplyTransform</b>	Applies the transform to this database.
<b>Commit</b>	Finalizes the persistent form of the database.
<b>CreateTransformSummaryInfo</b>	Creates and populates the summary information stream of an existing transform file.
<b>EnableUIPreview</b>	Facilitates the authoring of dialog boxes and billboards by providing the support needed to view user interface dialog boxes stored in the installer database.
<b>Export</b>	Copies the structure and data from a specified table to a text archive file.
<b>GenerateTransform</b>	Creates a transform.
<b>Import</b>	Imports a database table from a text archive file.
<b>Merge</b>	Merges the reference database with the base database.
<b>OpenView</b>	Returns a <b>View</b> object representing the query specified by a SQL string.

## Properties

The **Database** object defines the following properties.

<b>Property</b>	<b>Access type</b>	<b>Description</b>
<b>DatabaseState</b>	Read-only	Returns the persistence state of the database.

<b>PrimaryKeys</b>	Read-only	Returns a <b>Record</b> object containing the table name and the column names (comprising the primary keys).
<b>SummaryInformation (Database Object)</b>	Read-only	Returns a <b>SummaryInfo</b> object that can be used to examine, update, and add properties to the summary information stream.
<b>TablePersistent</b>	Read-only	Returns the persistence state of the table.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IDatabase is defined as 000C109D-0000-0000-C000-000000000046

## See Also

[Windows Installer Scripting Examples](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Database.ApplyTransform Method

The **ApplyTransform** method of the **Database** object applies the transform to this database.

## Syntax

```
Script
ApplyTransform(
    storage,
    errorConditions
)
```

## Parameters

### *storage*

Path to the transform file being applied. This parameter is required.

### *errorConditions*

Specifies the error conditions that are to be suppressed. Specify as a combination of the following integer values.

<b>Error condition</b>	<b>Meaning</b>
msiTransformErrorAddExistingRow 0x0001	Adds a row that already exists.
msiTransformErrorDeleteNonExistingRow 0x0002	Deletes a row that does not exist.
msiTransformErrorAddExistingTable 0x0004	Adds a table that already exists.
msiTransformErrorDeleteNonExistingTable 0x0008	Deletes a table that does not exist.
msiTransformErrorUpdateNonExistingRow 0x0010	Updates a row that does not exist.
msiTransformErrorChangeCodePage	Transform and database

0x0020	code pages do not match and neither has a neutral code page.
msiTransformErrorViewTransform 0x0100	Creates the temporary <a href="#">_TransformView</a> table.

## Return Value

This method does not return a value.

## Remarks

The **ApplyTransform** method delays transforming tables until the last possible moment. The steps taken in **ApplyTransform** are to immediately transform the table and column catalogs for the database. The table and column catalogs are updated according to which table is added or deleted and which column is added (no deletion of columns is allowed). If a table is currently loaded in memory and needs to be transformed, it is transformed. Otherwise, the table state is set to that requiring a transform so that when the table is loaded, or when the database is committed, the transform is applied. Transform in this instance means that the actual (row) data of the table is added, deleted, or updated.

If the method fails, you can obtain extended error information by using the [LastErrorRecord](#) method.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll

**IID**

IID\_IDatabase is defined as 000C109D-0000-0000-  
C000-0000000000046

## See Also

[Database](#)

[Database Transforms](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Database.Commit Method

The **Commit** method of the [Database](#) object finalizes the persistent form of the database. All persistent data is written to the writeable database, and no temporary columns or rows are written. This method has no effect on a database opened as read-only. This method can be called multiple times to save the current state of tables loaded into memory. When the database is finally closed, any changes made subsequent to the last **Commit** are rolled back. This method is normally called prior to shutdown when all database changes have been finalized.

## Syntax

```
Script  
Commit()
```

## Parameters

This method has no parameters.

## Return Value

This method does not return a value.

## Remarks

If the method fails, you can obtain extended error information by using the [LastErrorRecord](#) method.

## Requirements

Version
Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000

**DLL**

Msi.dll

**IID**

IID\_IDatabase is defined as 000C109D-0000-0000-C000-000000000046

## See Also

### [Database](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Database.CreateTransformSummaryInfo Method

The **CreateTransformSummaryInfo** method of the **Database** object creates and populates the summary information stream of an existing transform file. This method fills in the properties with the base and reference **ProductCode** and **ProductVersion**.

## Syntax

```
Script
CreateTransformSummaryInfo(
    reference,
    storage,
    errorConditions,
    validation
)
```

## Parameters

*reference*

Required database that does not include the changes.

*storage*

The name of the generated transform file. This is optional.

*errorConditions*

Required error conditions that should be suppressed when the transform is applied. Combine one or more of the following error condition values.

Error condition name	Meaning
msiTransformErrorNone 0	None of the following conditions.
msiTransformErrorAddExistingRow 1	Adds a row that already exists.

msiTransformErrorDeleteNonExistingRow 2	Deletes a row that does not exist.
msiTransformErrorAddExistingTable 4	Adds a table that already exists.
msiTransformErrorDeleteNonExistingTable 8	Deletes a table that does not exist.
msiTransformErrorUpdateNonExistingRow 16	Updates a row that does not exist.
msiTransformErrorChangeCodepage 32	Transform and database code pages do not match and neither code page is neutral.

*validation*

Required when the transform is applied to a database; shows which properties should be validated to verify that this transform can be applied to the database. The properties are all contained in the [Summary Information Stream Property Set](#).

Combine one or more of the following values.

<b>Validation flag</b>	<b>Meaning</b>
msiTransformValidationNone 0	No validation done.
msiTransformValidationLanguage 1	Default language must match base database.
msiTransformValidationProduct 2	Product must match base database.

To validate product version, first choose one or more of these three flags to indicate how much of the version is to be verified.

<b>Validation flag</b>	<b>Meaning</b>
msiTransformValidationMajorVer 8	Checks major version only.
msiTransformValidationMinorVer 16	Checks major and minor version only.
msiTransformValidationUpdateVer 32	Checks major, minor, and update versions.

Then choose one of the following to indicate the required relationship between the product version of the database the transform is being applied to and that of the base database.

<b>Validation flag</b>	<b>Meaning</b>
msiTransformValidationLess 64	Applied version < base version
msiTransformValidationLessOrEqual 128	Applied version <= base version
msiTransformValidationEqual 256	Applied version = base version
msiTransformValidationGreaterOrEqual 512	Applied version >= base version
msiTransformValidationGreater 1024	Applied version > base version

To validate that the transform is being applied to a package having the appropriate **UpgradeCode**, set the following flag.

<b>Validation flag</b>	<b>Meaning</b>
msiTransformValidationUpgradeCode	Validates that the transform is

2048

the appropriate **UpgradeCode**.

## Return Value

This method does not return a value.

## Remarks

To create a summary information stream for a transform, the **ProductCode** and **ProductVersion** properties must be defined in the **Property** tables of both the base and reference databases. If **msiTransformValidationUpgradeCode** is used, the **UpgradeCode** property must be defined in both databases.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IDatabase is defined as 000C109D-0000-0000-C000-000000000046

## See Also

[Database Transforms](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Database.DatabaseState Property

The **DatabaseState** property of the **Database** object is a read-only property.

This property returns the persistence state of the database as one of the following parameters.

Database state	Value	Description
msiDatabaseStateRead	0	Database opens as read-only. Changes to persistent data are not permitted and temporary data is not saved.
msiDatabaseStateWrite	1	Database is fully operational for read and write.

## Syntax

Script

```
Database.DatabaseState
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IDatabase is defined as 000C109D-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Database.EnableUIPreview Method

The **EnableUIPreview** method of the **Database** object facilitates the authoring of dialog boxes and billboards by providing the support needed to view user interface dialog boxes stored in the installer database. The method starts the preview mode by returning a preview **Database** object. The preview mode ends when the preview object is released.

## Syntax

```
Script  
EnableUIPreview()
```

## Parameters

This method has no parameters.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IDatabase is defined as 000C109D-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Database.Export Method

The **Export** method of the **Database** object copies the structure and data from a specified table to a [text archive file](#).

## Syntax

```
Script
Export(
    table,
    path,
    file
)
```

## Parameters

*table*

Required name of the database table. Case-sensitive if using the installer database.

*path*

Required string that is the path to the folder where the text file is placed.

*file*

Required name of the file to be created. This does not include the folder, as that must be set in the path object.

## Return Value

This method does not return a value.

## Remarks

If the method fails, you can obtain extended error information by using the [LastErrorRecord](#) method.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IDatabase is defined as 000C109D-0000-0000-C000-000000000046

## See Also

### [Database](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Database.GenerateTransform Method

The **GenerateTransform** method of the **Database** object creates a *transform* that, when applied to the object database, results in the reference database. The transform is stored in the storage object.

If the transform is to be applied during an installation you must use the **CreateTransformSummaryInfo** method to populate the summary information stream.

## Syntax

```
Script
GenerateTransform(
    reference,
    storage
)
```

## Parameters

*reference*

Required database that does not include the changes.

*storage*

The name of the generated transform file. This is optional.

## Return Value

This method does not return a value.

## Remarks

A transform can add non-primary key columns to the end of a table. A transform cannot be created that adds primary key columns to a table. A transform cannot be created that changes the order, names, or definitions of columns.

This method returns a Boolean value. It returns TRUE if a transform is generated. It returns FALSE if a transform is not generated because there are no differences between the two databases. If the method fails, it generates an error.

If the method fails, you can obtain extended error information by using the [LastErrorRecord](#) method.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IDatabase is defined as 000C109D-0000-0000-C000-000000000046

## See Also

### [Database](#)

### [Database Transforms](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Database.Import Method

The **Import** method of the **Database** object imports a database table from a [text archive files](#), dropping any existing table.

## Syntax

```
Script
Import(
    path,
    file
)
```

## Parameters

*path*

Required folder where the text file is present.

*file*

Required name of the file to be imported. This does not include the folder, as that must be set in the path object. The table name is specified within the file.

## Return Value

This method does not return a value.

## Remarks

If the method fails, you can obtain extended error information by using the [LastErrorRecord](#) method.

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows

<b>Version</b>	Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IDatabase is defined as 000C109D-0000-0000-C000-000000000046

## See Also

[Database MsiDatabaseImport](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Database.Merge Method

The **Merge** method of the [Database](#) object merges the reference database with the base database.

## Syntax

```
Script
Merge(
    reference,
    errorTable
)
```

## Parameters

*reference*

The required [Database](#) object to be merged into the database.

*errorTable*

An optional name of a table to contain the names of the tables containing merge conflicts, the number of conflicting rows within the table, and a reference to the table with the merge conflict.

## Return Value

This method does not return a value.

## Remarks

The [MsiDatabaseMerge](#) function and the **Merge** method of the **Database** object cannot be used to merge a module included within an installation package. They should not be used to merge [Merge Modules](#) into a Windows Installer package. To include a merge module in an installation package, authors of installation packages should follow the guidelines that are described in the [Applying Merge Modules](#) topic.

The **Merge** method does not copy over embedded [cabinet files](#) or [embedded transforms](#) from the reference database into the target database. Embedded data streams that are listed in the [Binary Table](#) or

[Icon Table](#) are copied from the reference database to the target database. Storages embedded in the reference database are not copied to the target database.

If no table is provided, the general error message provides the number of tables containing merge conflicts. Any table can be passed in, but all other columns must be nullable because the operation to update the [Error Table](#) fails if a column is not nullable. A newly created table can be passed in as well because the **Merge** method automatically creates the columns it uses if merge conflicts are found. Two columns are used to present merge conflicts. The first column is the table name and the primary key column. The second column is the number of rows of that table that have merge failures.

If tables of the same name in both databases do not match in the number of primary keys, the column types, the number of columns, or the column names, the **Merge** method fails and posts an error message stating what occurred.

For the Error table to remain, the error handler must commit the database to which the Error table belongs. However, this commit should be done after using the third column to obtain the references to those tables where merge conflicts occurred.

If the method fails, you can obtain extended error information by using the [LastErrorRecord](#) method.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IDatabase is defined as 000C109D-0000-0000-C000-000000000046

## See Also

## Database

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Database.OpenView Method

The **OpenView** method of the **Database** object returns a **View** object that represents the query specified by a SQL string.

## Syntax

```
Script  
OpenView(  
    sql  
)
```

## Parameters

*sql*  
Required SQL query string.

## Return Value

This method does not return a value.

## Remarks

For information about SQL syntax implemented in the installer, see [SQL Syntax](#).

If the method fails, you can obtain extended error information by using the [LastErrorRecord](#) method.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll

**IID**

IID\_IDatabase is defined as 000C109D-0000-0000-  
C000-0000000000046

## See Also

[Database  
SQL Syntax](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Database.PrimaryKeys Property

The **PrimaryKeys** property of the **Database** object returns a **Record** object containing the table name in field 0 and the column names (comprising the primary keys) in succeeding fields corresponding to their column numbers. The field count of the record is the count of primary key columns.

## Syntax

```
Script  
Database.PrimaryKeys
```

## Remarks

The **PrimaryKeys** property cannot be used with the [\\_Tables table](#) or the [\\_Columns table](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IDatabase is defined as 000C109D-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Database.SummaryInformation Property

The **SummaryInformation** property of the **Database** object returns a **SummaryInfo** object that can be used to examine, update, and add properties to the **summary information stream**.

## Syntax

```
Script  
Database.SummaryInformation
```

## Remarks

If a value of *maxProperties* greater than 0 is used to open an existing summary information stream, the **Persist** method must be called before closing the object. Failing to do this will lose the existing stream information.

If the property fails, you can obtain extended error information by using the **LastErrorRecord** method.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IDatabase is defined as 000C109D-0000-0000-C000-000000000046

## See Also

## Database

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Database.TablePersistent Property

The **TablePersistent** property of the **Database** object returns the persistence state of the table as one of the following parameters.

Table state	Value	Description
msiEvaluateConditionFalse	0	Table is temporary.
msiEvaluateConditionTrue	1	Table is persistent.
msiEvaluateConditionNone	2	Table is not in the database.
msiEvaluateConditionError	3	Invalid or missing table name.

## Syntax

[Script](#)

```
Database.TablePersistent
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IDatabase is defined as 000C109D-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# FeatureInfo Object

The **FeatureInfo** object contains information regarding the targeted feature and is created from the **Session** object using the **FeatureInfo** Method.

## Methods

The **FeatureInfo** object does not define any methods.

## Properties

The **FeatureInfo** object defines the following properties.

<b>Property</b>	<b>Access type</b>	<b>Description</b>
<b>Attributes</b>	Read-only	Returns the value for the feature in the Attributes column of the Feature table.
<b>Description</b>	Read-only	Returns the description of the feature.
<b>Title</b>	Read-only	Returns the title of the feature.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IFeatureInfo is defined as 000C109F-0000-0000-C000-000000000046

## See Also

[Windows Installer Scripting Examples](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# FeatureInfo.Attributes Property

The **Attributes** property is a read-write property that returns the value for the feature in the Attributes column of the [Feature table](#).

## Syntax

```
Script  
FeatureInfo.Attributes
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IFeatureInfo is defined as 000C109F-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# FeatureInfo.Description Property

The **Description** property is a read-only property that returns the description of the feature in the Description column of the [Feature table](#).

## Syntax

```
Script  
FeatureInfo.Description
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IFeatureInfo is defined as 000C109F-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# FeatureInfo.Title Property

The **Title** property is a read-only property that returns the title of the feature in the Title column of the [Feature table](#).

## Syntax

Script

```
FeatureInfo.Title
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IFeatureInfo is defined as 000C109F-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer Object

An **Installer** object must be initially created to load the automation support that is required for COM to access the installer functions. This object provides wrappers to create the top-level objects and access their methods.

You can create the **Installer** object from the ProgId "WindowsInstaller.Installer".

## Methods

The **Installer** object defines the following methods.

Method	Description
<b>AddSource</b>	Adds a source to the list of valid network sources in the sourcelist.
<b>AdvertiseProduct</b>	Advertises an installation package.
<b>AdvertiseScript</b>	Advertises an installation package.
<b>ApplyPatch</b>	Invokes an installation and sets the <b>PATCH</b> property to the path of the patch package for each product listed by the patch package as eligible to receive the patch.
<b>ApplyMultiplePatches</b>	Applies one or more patches to products eligible to receive the patch. Sets the <b>PATCH</b> property to the path of the patch packages provided.
<b>ClearSourceList</b>	Removes all network sources from the sourcelist.
<b>CollectUserInfo</b>	Invokes a user interface wizard sequence that collects and stores both user information and the product code.

<b>ConfigureFeature</b>	Configures the installed state of a product feature.
<b>ConfigureProduct</b>	Installs or uninstalls a product.
<b>CreateAdvertiseScript</b>	Generates an advertise script.
<b>CreateRecord</b>	Returns a new <b>Record</b> object with the requested number of fields.
<b>EnableLog</b>	Enables logging of the selected message type for all subsequent installation sessions in the current process space.
<b>ExtractPatchXMLData</b>	Extracts information from a patch as an XML string.
<b>FileHash</b>	Takes the path to a file and returns a 128-bit hash of that file.
<b>FileSignatureInfo</b>	Takes the path to a file and returns a <b>SAFEARRAY</b> of bytes that represents the hash or the encoded certificate.
<b>FileSize</b>	Returns the size of the specified file.
<b>FileVersion</b>	Returns the version string or language string of the specified path.
<b>ForceSourceListResolution</b>	Forces the installer to search the source list for a valid product source the next time a source is required.
<b>InstallProduct</b>	Opens an installer package and initializes an installation session.
<b>LastErrorRecord</b>	Returns a <b>Record</b> object that contains error parameters for the most recent error from the function that produced the error record.

<b>OpenDatabase</b>	Opens an existing database or creates a new one.
<b>OpenPackage</b>	Opens an installer package for use with functions that access the product database and install engine.
<b>OpenProduct</b>	Opens an installer package for an installed product using the product code.
<b>ProvideAssembly</b>	Returns the installed path of an assembly.
<b>ProvideComponent</b>	Returns the full component path and performs any necessary installation.
<b>ProvideQualifiedComponent</b>	Returns the full component path and performs any necessary installation.
<b>RegistryValue</b>	Reads information about a specified registry key of value.
<b>ReinstallFeature</b>	Reinstalls features or corrects problems with installed features.
<b>ReinstallProduct</b>	Reinstalls a product or corrects installation problems in an installed product.
<b>RemovePatches</b>	Removes one or more patches to products eligible to receive the patch.
<b>UseFeature</b>	Increments the usage count for a particular feature and returns the installation state for that feature.

## Properties

The **Installer** object defines the following properties.

---

Property	Access type	Description
<b>ClientsEx</b>	Read-only	Returns a <b>RecordList</b> object that lists products that use a specified installed component.  <b>Windows Installer 4.5 and earlier:</b> Not supported.
<b>ComponentClients</b>	Read-only	Returns a <b>StringList</b> object enumerating the set of clients of a specified component.
<b>ComponentPath</b>	Read-only	Returns the full path to an installed component.
<b>ComponentPathEx</b>	Read-only	Returns a <b>RecordList</b> object that gives the full path of a specified installed component.  <b>Windows Installer 4.5 and earlier:</b> Not supported.
<b>ComponentQualifiers</b>	Read-only	Returns a <b>StringList</b> object enumerating the set of registered qualifiers for the specified component.
<b>Components</b>	Read-only	Returns a <b>StringList</b> object enumerating the set of installed components for all products.
<b>ComponentsEx</b>	Read-only	Returns a <b>RecordList</b> object that lists installed components.  <b>Windows Installer 4.5 and earlier:</b> Not supported.
<b>Environment</b>	Read-	The string value for an environment

	only	variable of the current process.
<b>FeatureParent</b>	Read-only	Specifies the parent feature of a feature.
<b>Features</b>	Read-only	Returns a <b>StringList</b> object enumerating the set of published features for the specified product.
<b>FeatureState</b>	Read-only	Returns the installed state of a feature.
<b>FeatureUsageCount</b>	Read-only	Returns the number of times that the feature has been used.
<b>FeatureUsageDate</b>	Read-only	Returns the date that the specified feature was last used.
<b>FileAttributes</b>	Read-only	Returns a number that represents the combined file attributes for the designated path to a file or folder.
<b>Patches</b>	Read-only	Returns a <b>StringList</b> object that contains all the patches applied to the product.
<b>PatchesEx</b>	Read-only	Enumerates a collection of <b>Patch</b> objects.
<b>PatchFiles</b>	Read-only	Returns a <b>StringList</b> object that contains a list of files that can be updated by the provided list of patches.
<b>PatchInfo</b>	Read-only	Returns information about a patch.
<b>PatchTransforms</b>	Read-only	Returns the semicolon delimited list of transforms that are in the specified patch package and applied to the specified product.
<b>ProductElevated</b>	Read-	Returns True if the product is managed or

	only	False if the product is not managed.
<b>ProductInfo</b>	Read-only	Returns the value of the specified attribute for an installed or published product.
<b>ProductInfoFromScript</b>	Read-only	Returns the value of the specified attribute that is stored in an advertise script.
<b>Products</b>	Read-only	Returns a <b>StringList</b> object enumerating the set of all products installed or advertised for the current user and machine.
<b>ProductsEx</b>	Read-only	Enumerates a collection of <b>Product</b> objects.
<b>ProductState</b>	Read-only	Returns the install state information for a product.
<b>QualifierDescription</b>	Read-only	Returns a text string that describes the qualified component.
<b>RelatedProducts</b>	Read-only	Returns a <b>StringList</b> object enumerating the set of all products installed or advertised for the current user and machine with a specified <b>UpgradeCode</b> property in their property table.
<b>ShortcutTarget</b>	Read-only	Examines a shortcut and returns its product, feature name and component if available.
<b>SummaryInformation</b>	Read-only	Returns a <b>SummaryInfo</b> object that can be used to examine, update and add properties to the summary information stream of a package or transform.
<b>UILevel</b>	Read-	Indicates the type of user interface to be

	only	used when opening and processing subsequent packages within the current process space.
<b>Version</b>	Read-only	Returns the string representation of the current version of Windows Installer.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[Using the Automation Interface](#)  
[Windows Installer Scripting Examples](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.AddSource Method

The **AddSource** method of the **Installer** object adds a source to the list of valid network sources in the sourcelist.

## Syntax

```
Script
AddSource(
    Product,
    User,
    Source
)
```

## Parameters

### *Product*

Specifies the product code.

### *User*

User name for per-user installation; null or empty string for per-machine installation.

### *Source*

Pointer to the string specifying the source.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll

## **IID**

IID\_Installer is defined as 000C1090-0000-0000-C000-000000000046

## **See Also**

### **[MsiSourceListAddSource](#)**

[source resiliency](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer::AdvertiseProduct Method

The **AdvertiseProduct** method of the **Installer** object advertises an installation package.

## Syntax

```
Script
AdvertiseProduct(
    packagePath,
    context,
    transforms,
    language,
    options
)
```

## Parameters

### *packagePath*

The full path to the Windows Installer package (.msi) to be advertised.

### *context*

The context of the advertisement. This parameter can be one of the following values.

Value	Meaning
msiAdvertiseProductMachine 0	Advertises the application for an installation in the per-machine <a href="#">installation context</a> . This makes the package available for installation by all users of the computer.
msiAdvertiseProductUser 1	Advertises the application for an installation in the per-user installation context.

### *transforms*

The list of transforms to apply to the product. Transforms in the list are delimited by semicolons. This parameter is optional.

### *language*

The language of the installation package to use. This parameter is optional.

### *options*

The advertisement options. This parameter is optional. This parameter can be one of the following values.

Value	Meaning
msiAdvertiseDefault 0	Standard advertisement
msiAdvertiseSingleInstance 1	Advertises a new instance of the product. Requires that the first transform in the transform list of the <i>transforms</i> parameter be the instance transform that changes the product code. For more information, see <a href="#">Installing Multiple Instances of Products and Patches</a> .

## Return Value

This method does not return a value.

## Remarks

The **AdvertiseProduct** method uses the **MsiAdvertiseProductEx** function.

## Examples

The following example demonstrates the use of the **AdvertiseProduct** method.

---

```

Dim installer
Set installer = CreateObject("WindowsInstaller.Installer")

'
' Perform machine advertisement of package, use transform
'

Installer.AdvertiseProduct "c:\scratch\simpletst\rtm\simple

'
' Verify advertised product state and registration
'

MsgBox Installer.ProductState("{BAE98781-CF88-4309-8E2D-3D8
MsgBox Installer.ProductInfo("{BAE98781-CF88-4309-8E2D-3D8

'
' Remove Product
'

Installer.InstallProduct "c:\scratch\simpletst\rtm\simple.r

```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 4.5 on Windows Server 2003 and Windows XP
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[Installer](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer::AdvertiseScript Method

The **AdvertiseScript** method of the **Installer** object advertises an installation package.

## Syntax

```
Script
AdvertiseScript(
    scriptPath,
    scriptFlags,
    removeItems
)
```

## Parameters

### *scriptPath*

The full path to the script file generated by the **CreateAdvertiseScript** method.

### *scriptFlags*

The flags that control the advertisement. This parameter can be a combination of the following values.

Value	Meaning
msiAdvertiseScriptCacheInfo 0x001	Include this flag if the icons need to be created or removed.
msiAdvertiseScriptShortcuts 0x004	Include this flag if the shortcuts need to be created or removed.
msiAdvertiseScriptMachineAssign 0x008	Include this flag if the product is to be assigned to a computer.
msiAdvertiseScriptConfigurationRegistration	Include this flag if the

0x020	configuration and management information in the registry data needs to be written or removed.
msiAdvertiseScriptValidateTransformList 0x040	Include this flag to force the validation of the transforms listed in the script against previously registered transforms for this product. Note that transform conflicts are detected using a string comparison that is case insensitive and are evaluated between per-user and per-machine installations across all <a href="#">installation contexts</a> .
msiAdvertiseScriptClassInfoRegistration 0x080	Include this flag if advertisement information in the registry related to COM classes needs to be written or removed.
msiAdvertiseScriptExtensionInfoRegistration 0x100	Include this flag if advertisement information in the registry related to an extension needs to be written or removed.
msiAdvertiseScriptAppInfo 0x180	Include this flag if the advertisement information in the registry needs to be

	written or removed.
msiAdvertiseScriptRegData 0x1A0	Include this flag if the advertisement information in the registry needs to be written or removed.

### *removeItems*

TRUE if the specified items are to be removed instead of being created.

## Return Value

This method does not return a value.

## Remarks

The **AdvertiseScript** method uses the **MsiAdvertiseScript** function. The use of the **AdvertiseScript** method requires that the script be running within a local system process.

## Examples

The following example demonstrates the use of the **AdvertiseScript** method.

```
Dim installer
Set installer = CreateObject("WindowsInstaller.Installer")

' Advertise Simple package using an advertise script
'   created by CreateAdvertiseScript Method
'
'   Flags 424 indicate msiAdvertiseScriptMachineAssign, msiA
Installer.AdvertiseScript "c:\scratch\simpletst\rtm\simple.
' Verify Simple is installed
```

```
MsgBox Installer.ProductState("{BAE98781-CF88-4309-8E2D-3D8
'
' Remove Simple using advertise script
'
Installer.AdvertiseScript "c:\scratch\simpletst\rtm\simple.
' Verify simple is removed
MsgBox Installer.ProductState("{BAE98781-CF88-4309-8E2D-3D8
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 4.5 on Windows Server 2003 and Windows XP
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

### [Installer](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.ApplyPatch Method

For each product listed by the patch package as eligible to receive the patch, the **ApplyPatch** method of the **Installer** object invokes an installation and sets the **PATCH** property to the path of the patch package.

## Syntax

Script

```
ApplyPatch(  
    PatchPackage,  
    InstallPackage,  
    InstallType,  
    CommandLine  
)
```

## Parameters

*PatchPackage*

Specifies a path to the patch package.

*InstallPackage*

If *InstallType* is set to `msiInstallTypeNetworkImage`, *InstallPackage* specifies the path to the product that is to be patched. If *InstallType* is set to `msiInstallTypeDefault` and *InstallPackage* is set to 0, the installer applies the patch to every eligible product listed in the patch package.

If *InstallType* is `msiInstallTypeSingleInstance`, the installer applies the patch to the product specified by *InstallPackage*. In this case, other eligible products listed in the patch package are ignored and the *InstallPackage* parameter contains the null-terminated string representing the product code of the instance to patch. This type of installation requires the Windows Installer version shipped with the Windows Server 2003 or later or Windows Installer XP SP1 or later.

*InstallType*

This parameter specifies the type of installation to patch. The *InstallType* parameter is ignored if *InstallPackage* is omitted.

Value	Meaning
msiInstallTypeNetworkImage	Indicates a administrative installation. In this case, <i>InstallPackage</i> must be set to a package path. A value of 1 for msiInstallTypeNetworkImage specifies a administrative installation.
msiInstallTypeDefault	Searches system for products to patch. In this case, <i>InstallPackage</i> must be an empty string.
msiInstallSingleInstance	Patch the product specified by <i>InstallPackage</i> . <i>InstallPackage</i> is the product code of the instance to patch. This type of installation requires the Windows Installer version shipped with Windows Server 2003 or later or Windows Installer XP SP1 or later. For more information see, <a href="#">Installing Multiple Instances of Products and Patches</a> .

### CommandLine

Specifies property settings being set on the command line. See Remarks section.

### Return Value

This method does not return a value.

### Remarks

Because the list delimiter for transforms, sources, and patches is a semicolon, this character should not be used for file names or paths.

The **REINSTALL** property is required when applying a [small update](#) or [minor upgrade](#) patch. Without this property, the patch is registered on the

system but cannot update files.

**Windows Installer 2.0:** You must set the **REINSTALL** property on the command line when applying a small update or minor upgrade patch. For patches that do not use a [Custom Action Type 51](#) to automatically set the **REINSTALL** and **REINSTALLMODE** properties, the **REINSTALL** property must be explicitly set with the *CommandLine* parameter. Set the **REINSTALL** property to list the features affected by the patch, or use a practical default setting of "REINSTALL=ALL". The default value of the **REINSTALLMODE** property is "omus".

**Windows Installer 3.0 and later:** Beginning with Windows Installer version 3.0, the **REINSTALL** property is configured by the installer and does not need to be set on the command line.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[MsiApplyPatch](#)  
[About Properties](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.ApplyMultiplePatches Method

The **ApplyMultiplePatches** method applies one or more patches to products that are eligible to receive the patch. The method sets the **PATCH** property.

## Syntax

```
Script
ApplyMultiplePatches(
    PatchPackagesList,
    Product,
    szPropertiesList
)
```

## Parameters

### *PatchPackagesList*

A string that contains a semicolon-delimited list of the paths to patch files. For example: ""c:\sus\download\cache\Office\sp1.msp;c:\sus\download\cache\Office\QFE1.msp;c:\sus\download\cache\Office\QFE2.msp"

### *Product*

This parameter provides the **ProductCode** of the product being patched. This parameter is optional. When this parameter is null, the patches are applied to all products that are eligible to receive these patches.

### *szPropertiesList*

A null-terminated string that specifies command-line property settings. This parameter is optional. See [About Properties](#) and [Setting Public Property Values on the Command Line](#). These properties are shared by all target products.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[About Properties](#)

[ProductCode](#)

[PATCH](#)

[Setting Public Property Values on the Command Line](#)

[MsiApplyMultiplePatches](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.ClearSourceList Method

The **ClearSourceList** method of the **Installer** object removes all network sources from the source list.

## Syntax

```
Script
ClearSourceList(
    Product,
    User
)
```

## Parameters

### *Product*

Specifies the product code.

### *User*

User name for per-user installation; null or empty string for per-machine installation.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[MsiSourceListClearAll](#)  
Source Resiliency

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.ClientsEx Property

This property returns a [RecordList](#) object that lists products that use a specified installed component. This property calls [MsiEnumClientsEx](#).

**Windows Installer 4.5 or earlier:** Not supported. This property is available beginning with Windows Installer 5.0.

## Syntax

Script

```
Installer.ClientsEx
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[MsiEnumClientsEx](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.CollectUserInfo Method

The **CollectUserInfo** method of the [Installer](#) object invokes a user interface wizard sequence which collects and stores both user information and the product code.

## Syntax

```
Script  
CollectUserInfo(  
    Product  
)
```

## Parameters

*Product*

Specifies the **product code** of the product.

## Return Value

This method does not return a value.

## Remarks

An application should call the **CollectUserInfo** method the first time it is run. The **CollectUserInfo** method opens the product's installation package and invokes an authored user interface wizard sequence which collects user information. Upon completion of the wizard sequence, the collected user information is registered. The **UILevel** property should be set to `msiUILevelFull` because this API requires an authored user interface.

The **CollectUserInfo** method invokes the [FirstRun Dialog](#).

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows

<b>Version</b>	Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

### [MsiCollectUserInfo](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.ComponentClients Property

The read-only **ComponentClients** property returns a **StringList** object enumerating the set of clients of a specified component.

## Syntax

```
Script  
Installer.ComponentClients
```

## Remarks

To enumerate the component clients, an application may iterate through the **StringList** object using a For Each construct. Because clients are not ordered, any new components has an arbitrary index. This means that the function may return clients in any order.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

### [MsiEnumClients](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.ComponentPath Property

The **ComponentPath** property is a read-only property that returns the full path to an installed component. If the key path for the component is a registry key then the registry key is returned.

## Syntax

Script

```
Installer.ComponentPath
```

## Remarks

If the component is a registry key, the registry roots are represented numerically. For example, a registry path of "HKEY\_CURRENT\_USER\SOFTWARE\Microsoft" would be returned as "01:\SOFTWARE\Microsoft". The registry roots returned are defined as follows:

Root	Returned value
HKEY_CLASSES_ROOT	00
HKEY_CURRENT_USER	01
HKEY_LOCAL_MACHINE	02
HKEY_USERS	03
HKEY_PERFORMANCE_DATA	04

## Requirements

**Version**

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000

**DLL**

Msi.dll

**IID**

IID\_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

### [MsiGetComponentPath](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.ComponentPathEx Property

This property returns a [RecordList](#) object that gives the full path of a specified installed component. This property calls [MsiGetComponentPathEx](#).

**Windows Installer 4.5 or earlier:** Not supported. This property is available beginning with Windows Installer 5.0.

## Syntax

```
Script  
Installer.ComponentPathEx
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_Installer is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[MsiGetComponentPathEx](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.ComponentQualifiers Property

The **ComponentQualifiers** property is a read-only property that returns a **StringList** object enumerating the set of registered qualifiers for the specified component.

## Syntax

```
Script  
Installer.ComponentQualifiers
```

## Remarks

To enumerate qualifiers the application iterates through the **StringList** object using a For Each construct. Because qualifiers are not ordered, any new qualifier has an arbitrary index, meaning the function can return qualifiers in any order.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[MsiEnumComponentQualifiers](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.Components Property

The read-only **Components** property returns a **StringList** object enumerating the set of installed components for all products.

## Syntax

```
Script  
Installer.Components
```

## Remarks

To enumerate the components, an application can iterate through the **StringList** object using a For Each construct. Because components are not ordered, any new components has an arbitrary index. This means that the function can return components in any order.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

### [MsiEnumComponents](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.ComponentsEx Property

This property returns a [RecordList](#) object that lists installed components. This property calls [MsiEnumComponentsEx](#).

**Windows Installer 4.5 or earlier:** Not supported. This property is available beginning with Windows Installer 5.0.

## Syntax

Script

```
Installer.ComponentsEx
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[MsiEnumComponentsEx](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.ConfigureFeature Method

The **ConfigureFeature** method of the **Installer** object configures the installed state of a product feature.

## Syntax

```
Script
ConfigureFeature(
    Product,
    Feature,
    InstallState
)
```

## Parameters

### *Product*

Specifies the product code of the product.

### *Feature*

Specifies the feature ID of the feature to be configured.

### *InstallState*

Specifies the installation state for the feature. This parameter must be one of the following values.

Value	Meaning
msiInstallStateAdvertised	The feature is advertised
msiInstallStateLocal	The feature is installed locally.
msiInstallStateAbsent	The feature is uninstalled.
msiInstallStateSource	The feature is installed to run from source.
msiInstallStateDefault	The feature is installed to its default location.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

### [MsiConfigureFeature](#)

[Installation and Configuration Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.ConfigureProduct Method

The **ConfigureProduct** method of the **Installer** object installs or uninstalls a product.

## Syntax

```
Script
ConfigureProduct(
    Product,
    InstallLevel,
    InstallState
)
```

## Parameters

### *Product*

Specifies the product code of the product.

### *InstallLevel*

Specifies the default installation configuration of the product. The InstallLevel parameter is ignored and all features are installed if the InstallState parameter is set to any other value than msInstallStateDefault.

This parameter must be either 0 (install using authored feature levels), 65535 (install all features), or a value between 0 and 65535 to install a subset of available features.

### *InstallState*

Specifies the installation state for the feature. This parameter must be one of the following values.

Value	Meaning
msiInstallStateAdvertised	The feature is advertised
msiInstallStateLocal	The feature is installed locally.
msiInstallStateAbsent	The feature is uninstalled.

msiInstallStateSource	The feature is installed to run from source.
msiInstallStateDefault	The feature is installed to its default location.

## Return Value

This method does not return a value.

## Remarks

The **ConfigureProduct** method displays the user interface using current settings. User interface settings can be changed by modifying the **UILevel property (Installer object)** before calling the **ConfigureProduct** method.

If the *InstallState* parameter is set to any other value than *msiInstallStateDefault*, the *InstallLevel* parameter is ignored and all features of the product are installed. Use the **ConfigureFeature** method to control the installation of individual features when the *InstallState* parameter is not set to *msiInstallStateDefault*.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

**MsiConfigureProduct**  
Installation and Configuration Functions

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer::CreateAdvertiseScript Method

The **CreateAdvertiseScript** method of the **Installer** object generates an advertise script.

## Syntax

```
Script  
CreateAdvertiseScript(  
    packagePath,  
    scriptFilePath,  
    transforms,  
    language,  
    platform,  
    options  
)
```

## Parameters

### *packagePath*

The full path to the Windows Installer package (.msi) to be advertised.

### *scriptFilePath*

The full path to the script file to be created with the advertised information.

### *transforms*

The list of transforms to apply to the product. Transforms in the list are delimited by semicolons. This parameter is optional.

### *language*

The language of the installation package to use. This parameter is optional.

### *platform*

This parameter specifies for which platform the installer should create the script. This parameter can be one of the following values.

---

Value	Meaning
msiAdvertiseCurrentPlatform 0	Creates a script for the current platform.
msiAdvertiseX86Platform 1	Creates a script for the x86 platform.
msiAdvertiseIA64Platform 2	Creates a script for the Intel Itanium Processor Family (IPF).
msiAdvertiseX64Platform 4	Creates a script for the x64 platform.

### options

Advertisement options. This parameter is optional. This parameter can be one of the following values. This parameter is optional.

Value	Meaning
msiAdvertiseDefault 0	Standard advertisement
msiAdvertiseSingleInstance 1	Advertises a new instance of the product. Requires that the first transform in the transform list of the <i>transforms</i> parameter be the instance transform that changes the product code. For more information, see <a href="#">Installing Multiple Instances of Products and Patches</a> .

### Return Value

This method does not return a value.

### Remarks

The **AdvertiseProduct** method uses the **MsiAdvertiseProductEx** function.

## Examples

The following example demonstrates the use of the **CreateAdvertiseScript** method.

```
Dim installer
Set installer = CreateObject("WindowsInstaller.Installer")

'
' Create an advertise script for Orca
'

Installer.CreateAdvertiseScript "\\products\public\orca\orca"
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 4.5 on Windows Server 2003 and Windows XP
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

### [Installer](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.CreateRecord Method

The **CreateRecord** method of the **Installer** object returns a new **Record** object with the requested number of fields.

## Syntax

```
Script  
CreateRecord(  
    count  
)
```

## Parameters

*count*

Required number of fields, which may be 0. The maximum number of fields in a record is limited to 65535.

## Return Value

This method does not return a value.

## Remarks

Field 0, not one of the fields in *count*, is normally used for record-oriented items such as format strings or execution op-codes.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
	IID_IInstaller is defined as 000C1090-0000-0000-

**IID**

C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.EnableLog Method

The **EnableLog** method of the **Installer** object enables logging of the selected message type for all subsequent installation sessions in the current process space.

## Syntax

```
Script
EnableLog(
    logMode,
    logFile
)
```

## Parameters

### *logMode*

A required string that contains letters representing the message types to log. The string can be a combination of the following values.

Value	Description
I	Information-only messages.
w	Non-fatal warning messages.
e	Error messages that may be fatal errors.
f	List of files in use that need to be replaced.
a	Start of action notification.
r	Action data record containing content specific to action.
u	User request messages.
c	UI initialization parameters.
m	Out-of-memory message.
v	Sends large amounts of information to log file not generally useful to users. May be used for support.
p	Dump property table; "property = value" at engine termination

+	Append to existing log file.
!	Flush each line to the log file.
x	Extra debugging information(creation of handles etc.) This option only available with Windows Server 2003.
o	Out-of-disk-space messages.

### *logFile*

Required string containing the path to the log file to be created. Use an empty string ("") to turn off logging.

## Return Value

This method does not return a value.

## Remarks

The path to the logfile location must already exist when using this method. The Installer does not create the directory structure for the logfile.

The logging options set using **EnableLog** override any existing Windows Installer logging policy settings.

Logging overwrites an existing log file by default. You must use the '+' letter in the logging mode to append to an existing log file.

The '!' option is not recommended because it can significantly slow installation. This option may be useful when debugging an installation.

The following sample script turns on verbose logging for an installation. At the end of the installation, the generated log file will be at c:\temp\install.log.

```
Dim Installer
Set Installer = CreateObject("WindowsInstaller.Installer")
Installer.EnableLog "voicewarmup", "c:\temp\install.log"
Installer.InstallProduct "\\server\share\products\setup.exe"
```



## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[Windows Installer Logging](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.Environment Property

The **Environment** property of the [Installer](#) object is a read-write property that is the string value for an environment variable of the current process.

## Syntax

```
Script  
Installer.Environment
```

## Remarks

Setting an environment variable with the **Environment** property only affects the active session. To make persistent changes to an environment variable, use the [Environment table](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.ExtractPatchXMLData Method

The **ExtractPatchXMLData** method of the **Installer** object extracts information from a patch as an XML string. The information can be used to determine whether the patch applies on a target system. This method calls **MsiExtractPatchXMLData**.

## Syntax

```
Script
ExtractPatchXMLData(
    PatchPath
)
```

## Parameters

### *PatchPath*

Full path to the patch from which the applicability information is to be extracted.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
	IID_IInstaller is defined as 000C1090-0000-0000-

**IID**

C000-000000000046

## See Also

[MsiExtractPatchXMLData](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.FeatureParent Property

The **FeatureParent** property specifies the parent feature of a feature. An empty string for this property indicates the feature has no parent and is a root item. This is a read-only property.

## Syntax

```
Script  
Installer.FeatureParent
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.Features Property

The **Features** property is a read-only property that returns a **StringList** object enumerating the set of published features for the specified product.

## Syntax

```
Script  
Installer.Features
```

## Remarks

To enumerate the features, an application iterates through the **StringList** object using a For Each construct. Because features are not ordered, any new feature has an arbitrary index, meaning the function can return features in any order.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[MsiEnumFeatures](#)  
[System Status Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.FeatureState Property

The read-only **FeatureState** property returns the installed state of a feature.

## Syntax

Script

```
Installer.FeatureState
```

## Remarks

This property returns one of the following values.

Value	Description
msiInstallStateAbsent	The feature is not installed.
msiInstallStateAdvertised	The feature is advertised.
msiInstallStateLocal	The feature is installed to run locally.
msiInstallStateSource	The feature is installed to run from source.
msiInstallStateInvalidArg	An invalid parameter was passed to the function.
msiInstallStateUnknown	The product code or feature ID is unknown.
msiInstallStateBadConfig	The configuration data is corrupt.

The **FeatureState** property does not validate that the feature is accessible.

## Requirements

**Version**

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000

**DLL**

Msi.dll

**IID**

IID\_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

### [MsiQueryFeatureState](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.FeatureUsageCount Property

The **FeatureUsageCount** property is a read-only property that returns the number of times the feature has been used.

## Syntax

```
Script  
Installer.FeatureUsageCount
```

## Remarks

Use of the [UseFeature](#), [ProvideComponent](#) or [ProvideQualifiedComponent](#) methods or their API equivalents on the specified feature increments this property.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[MsiGetFeatureUsage](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.FeatureUsageDate Property

The **FeatureUsageDate** property is a read-only property that returns the date the specified feature was last used.

## Syntax

```
Script  
Installer.FeatureUsageDate
```

## Remarks

Use of the **UseFeature**, **ProvideComponent** or **ProvideQualifiedComponent** methods or their API equivalents on the specified feature sets this property.

The date is in the MS-DOS date format as shown in the following table.

Bits	Contents
0-4	Day of the month (1-31)
5-8	Month (1 = January, 2 = February, and so on)
9-15	Year offset from 1980 (add 1980 to get actual year)

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[MsiGetFeatureUsage](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.FileAttributes Property

The **FileAttributes** property of the **Installer** object returns a number representing the combined file attributes for the designated path to a file or folder.

## Syntax

```
Script  
Installer.FileAttributes
```

## Remarks

The **FileAttributes** property returns the following values.

File attribute	Value	Value
FILE_ATTRIBUTE_READONLY	0x00000001	1
FILE_ATTRIBUTE_HIDDEN	0x00000002	2
FILE_ATTRIBUTE_SYSTEM	0x00000004	4
FILE_ATTRIBUTE_DIRECTORY	0x00000010	16
FILE_ATTRIBUTE_TEMPORARY	0x00000100	256
FILE_ATTRIBUTE_COMPRESSED	0x00000800	2048
FILE_ATTRIBUTE_OFFLINE	0x00001000	4096

Returns -1 if the file or folder does not exist or is not accessible.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000

**DLL**

Msi.dll

**IID**

IID\_IInstaller is defined as 000C1090-0000-0000-  
C000-0000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.FileHash Method

The **FileHash** method of the **Installer Object** takes the path to a file and returns a 128-bit hash of that file. The file hash information is returned as a **Record Object**. The entire 128-bit file hash is returned as four 32-bit **IntegerData** property fields.

The values returned in the **Record Object** correspond to the four fields of the **MSIFILEHASHINFO** structure returned by **MsiGetFileHash**. The numbering of four fields is 1-based in the **MsiFileHash Table**.

- Field 1 corresponds to the HashPart1 column.
- Field 2 corresponds to the HashPart2 column.
- Field 3 corresponds to the HashPart3 column.
- Field 4 corresponds to the HashPart4 column.

## Syntax

```
Script
FileHash(
    FilePath,
    Options
)
```

## Parameters

### *FilePath*

Path to the file that is to be hashed.

### *Options*

Reserved for future use.

The value of this parameter must be 0 (zero).

## Return Value

If successful, this method returns a **Record Object** that contains the hash of the file.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[Default File Versioning](#)  
[Manage File Sizes and Versions](#)  
[MsiFileHash Table](#)  
[MsiGetFileHash](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.FileSignatureInfo Method

The **FileSignatureInfo** method of the **Installer** object takes the path to a file and returns a SAFEARRAY of bytes that represent the hash or the encoded certificate. The values can then be used to populate the **MsiDigitalSignature**, **MsiPatchCertificate**, and **MsiDigitalCertificate** tables. For more information, see the **SAFEARRAY Data Type**.

## Syntax

```
Script
FileSignatureInfo(
    FilePath,
    Options,
    Format
)
```

## Parameters

### *FilePath*

Full path to a file that is digitally signed.

When populating the **MsiDigitalSignature** and **MsiDigitalCertificate** tables, *FilePath* points to a digitally signed cabinet. When populating the **MsiPatchCertificate** and **MsiDigitalCertificate** tables, *FilePath* points to a digitally signed patch.

### *Options*

Special error case flags.

Flag	Meaning
msiSignatureOptionInvalidHashFatal 1	With <i>Options</i> set to <b>msiSignatureOptionInvalidHashFatal</b> , <b>FileSignatureInfo</b> always returns a fatal error for an invalid hash.  If <i>Options</i> is not set to <b>msiSignatureOptionInvalidHashFatal</b> and <i>Format</i> is set to

msiSignatureInfoCertificate, **FileSignatureInfo** does not return an error for an invalid hash.

### Format

The requested signature information.

Flag	Meaning
msiSignatureInfoCertificate 0	Returns a SAFEARRAY of bytes that represent the encoded certificate.
msiSignatureInfoHash 1	Returns a SAFEARRAY of bytes that represent the hash.

## Return Value

If successful, the method returns a SAFEARRAY of bytes that contain either the hash or encoded certificate.

## Remarks

To author a fully verified signed installation by using automation, use the **FileSignatureInfo** method to populate the [MsiDigitalCertificate](#), [MsiPatchCertificate](#), and [MsiDigitalSignature](#) tables. For more information, see [Authoring a Fully Verified Signed Installation Using Automation](#).

## Requirements

Version
Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000

<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[Authoring a Fully Verified Signed Installation Using Automation  
Digital Signatures and Windows Installer  
\*\*MsiGetFileSignatureInformation\*\*](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.FileSize Method

The **FileSize** method of the **Installer** object uses Win32 API calls to return the size of the file specified in *Path*.

## Syntax

```
Script  
FileSize(  
    Path  
)
```

## Parameters

*Path*

Required string containing the path to the file.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.FileVersion Method

The **FileVersion** method of the **Installer** object returns the version string or language string of the path specified in *Path* using the format in which the installer expects to find them in the database. For versions, this is a string in "#.#.#.#" format. For language, this is the decimal language ID.

## Syntax

```
Script
FileVersion(
    Path,
    Language
)
```

## Parameters

### *Path*

Required string containing the path to the file.

### *Language*

Flag for designating whether the returned value is a language ID or version string. TRUE returns the language, FALSE returns the version. This parameter is optional, with a default value of FALSE.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll

**IID**

IID\_Installer is defined as 000C1090-0000-0000-  
C000-0000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.ForceSourceListResolution Method

The **ForceSourceListResolution** method of the **Installer** object forces the installer to search the source list for a valid product source the next time a source is needed, such as when the installer performs an installation or a reinstallation, or when it needs the path for a component set to run from source.

## Syntax

```
Script  
ForceSourceListResolution(  
    Product,  
    User  
)
```

## Parameters

*Product*

Specifies the product code.

*User*

User name for per-user installation; null or empty string for per-machine installation.

## Return Value

This method does not return a value.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000

<b>DLL</b>	Msi.dll
<b>IID</b>	IID_Installer is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[MsiSourceListForceResolution](#)

[Source Resiliency](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.InstallProduct Method

The **InstallProduct** method of the **Installer** object opens an installer package and initializes an install session.

## Syntax

```
Script
InstallProduct(
    packagePath,
    propertyValues
)
```

## Parameters

*packagePath*

Required string containing the path to the install package.

*propertyValues*

Optional string containing property=value pairs separated by white space.

To perform an administrative installation, include ACTION=ADMIN in *propertyValues*. For more information, see the **ACTION** property.

## Return Value

This method does not return a value.

## Remarks

To completely remove a product, set REMOVE=ALL in *propertyValues*. For more information, see **REMOVE** property.

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows

<b>Version</b>	Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.LastErrorRecord Method

The **LastErrorRecord** method of the **Installer** object returns a **Record** object that contains error parameters for the most recent error from the function that produced the error record.

## Syntax

```
Script  
LastErrorRecord( )
```

## Parameters

This method has no parameters.

## Return Value

This method does not return a value.

## Remarks

The **Record** object is reset after the execution of this function of any function that generates an error record.

Only the following designated functions generate an error record:

- **OpenDatabase method (Installer Object)**
- **Commit**
- **OpenView**
- **Import**
- **Export**
- **Merge**
- **GenerateTransform**
- **ApplyTransform**
- **Execute**

- **Modify**
- **SetStream**
- **SummaryInformation**
- **SourcePath**
- **TargetPath**
- **ComponentCurrentState**
- **ComponentRequestState**
- **FeatureCurrentState**
- **FeatureRequestState**
- **FeatureCost**
- **FeatureValidStates**
- **SetInstallLevel**

The following sample in VBScript uses a call to **OpenDatabase** to show how to obtain extended error information from one of the methods or properties that support the **LastErrorRecord** method. The sample constructs an error message when the **OpenDatabase** method fails. The **Err** object is used to determine whether an error was encountered.

```
Const msiOpenDatabaseModeReadOnly      = 0

On Error Resume Next ' defer error handling

Dim installer
Set installer = CreateObject("WindowsInstaller.Installer")

' attempt to open the non-existent MSI database
Dim database
Set database = installer.OpenDatabase("c:\nonexistent.msi",

' test for error
If Err.Number <> 0 Then
    Dim message, errorRec
    message = Err.Source & " " & Hex(Err.Number) & ": "
    If Not installer Is Nothing Then
        ' try to obtain extended error info
```

```
        Set errorRec = installer.LastErrorRecord
        If Not errorRec Is Nothing Then message = m
    End If

    MsgBox message

    ' PLACE ADDITIONAL SCRIPTING CODE HERE TO LOG AND/OR
    ' DETERMINE WHETHER TO CONTINUE PROCESSING ANYTHING
End If
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.OpenPackage Method

The **OpenPackage** method of the **Installer** object opens an installer package for use with functions that access the product database and install engine, returning an **Session** object.

## Syntax

```
Script
OpenPackage(
    packagePath,
    options
)
```

## Parameters

*packagePath*

Required string containing the path name of the package.

*options*

An optional integer value that specifies whether or not **OpenPackage** should ignore the current computer state when creating the Session object. No value or a value of 0 for options defaults to the original behavior. When options is 1, the **OpenPackage** Method ignores the current computer state when opening the package. A value of 1 prevents changes to the current computer state. For more information, see [MsiOpenPackageEx](#).

## Return Value

This method does not return a value.

## Remarks

The **OpenPackage** method can accept the database handle directly instead of the string for the package path.

Note that only one **Session** object can be opened by a single process. **OpenPackage** cannot be used in a custom action because the active

installation is the only session allowed.

A safe **Session** object ignores the current computer state when opening the package and prevents changes to the current computer state. For more information, see **MsiOpenPackageEx**.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.OpenDatabase Method

The **OpenDatabase** method of the **Installer** object opens an existing database or creates a new one, returning a **Database** object. It generates an error if the **Database** object cannot be successfully created and opened.

## Syntax

```
Script
OpenDatabase(
    name,
    openMode
)
```

## Parameters

### *name*

Required string that contains the path name of the database. If an empty string is supplied, a temporary database is created that is not persisted.

### *openMode*

A parameter from the following list or a string that contains the path name of the new output database file that is to be written to upon commit.

Parameter	Meaning
msiOpenDatabaseModeReadOnly 0	Opens a database read-only, no persistent changes.
msiOpenDatabaseModeTransact 1	Opens a database read/write in transaction mode.
msiOpenDatabaseModeDirect 2	Opens a database direct read/write without transaction.
msiOpenDatabaseModeCreate	Creates a new database, transact

3	mode read/write.
msiOpenDatabaseModeCreateDirect 4	Creates a new database, direct mode read/write.
msiOpenDatabaseModeListScript 5	Opens a database to view advertise script files, such as the files generated by the <b>CreateAdvertiseScript</b> method.
msiOpenDatabaseModePatchFile 32	Adds this flag to indicate a patch file.

## Return Value

This method does not return a value.

## Remarks

When a database is opened as the output of another database, the summary information stream of the output database is actually a read-only mirror of the original database and thus cannot be changed. Additionally, it is not persisted with the database. To create or modify the summary information for the output database it must be closed and reopened.

To make and save changes to a database first open the database in transaction (`msiOpenDatabaseModeTransact`), create (`msiOpenDatabaseModeCreate` or `msiOpenDatabaseModeCreateDirect`), or direct (`msiOpenDatabaseModeDirect`) mode. After making the changes, always call the **Commit** method before closing the database handle. The **Commit** method flushes all buffers.

Always call the **Commit** method on a database that has been opened in direct mode (`msiOpenDatabaseModeDirect` or `msiOpenDatabaseModeCreateDirect`) before closing the database. Failure to do this may corrupt the database.

Because the **OpenDatabase** method initiates database access, it cannot be used with a running installation.

If the method fails, you can obtain extended error information by using the [LastErrorRecord](#) method.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

### [Installer](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.OpenProduct Method

The **OpenProduct** method of the **Installer** object opens an installer package for an installed product using the product code and returns a **Session** object.

## Syntax

```
Script  
OpenProduct(  
    productCode  
)
```

## Parameters

*productCode*

Required string containing the unique product code (a **GUID**) or an activation descriptor written by the installer.

## Return Value

This method does not return a value.

## Remarks

Note that only one **Session** object can be opened by a single process. **OpenProduct** cannot be used in a custom action because the active installation is the only session allowed.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
----------------	--

**DLL**

Msi.dll

**IID**

IID\_IInstaller is defined as 000C1090-0000-0000-  
C000-0000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.Patches Property

The read-only **Patches** property of the **Installer** object returns a **StringList** object that contains all the patches applied to the product.

## Syntax

```
Script  
Installer.Patches
```

## Remarks

To enumerate the patches, an application iterates through the **StringList** object using a For Each construct. Because patches are not ordered, any new patch has an arbitrary index. This means that the function can return patches in any order.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

### [MsiEnumPatches](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.PatchesEx Property

The **PatchesEx** property returns a [RecordList](#) object that enumerates the list of patches. This property calls [MsiEnumPatchesEx](#).

## Syntax

Script

```
Installer.PatchesEx
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_Installer is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[Installer](#)

[MsiEnumPatchesEx](#)

[Patch](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.PatchFiles Property

The **PatchFiles** property returns a **StringList** object that contains a list of files that can be updated by the provided list of patches. This property calls **MsiGetPatchFileList**. For more information about using the **PatchFiles** property see [Listing the Files that can be Updated](#).

## Syntax

Script

```
Installer.PatchFiles
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 4.5 on Windows Server 2003 and Windows XP
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

### [Installer Object](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.PatchInfo Property

The read-only **PatchInfo** property of the **Installer** object returns information about a patch.

## Syntax

```
Script  
Installer.PatchInfo
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

### [MsiGetPatchInfo](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.PatchTransforms Property

The read-only **PatchTransforms** property returns the semi-colon delimited list of transforms that are in the specified patch package and applied to the specified product.

## Syntax

```
Script  
Installer.PatchTransforms
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

### [MsiEnumPatches](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.ProductInfo Property

The **ProductInfo** property is a read-only property that returns the value of the specified attribute for an installed or published product.

## Syntax

```
Script  
Installer.ProductInfo
```

## Remarks

The **ProductInfo** property ("LocalPackage") does not necessarily return a path to the cached package. Maintenance mode installations should not be invoked from the LocalPackage. The cached package is for internal uses only.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_Installer is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[MsiGetProductInfo](#)  
[MsiGetUserInfo](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer::ProductInfoFromScript Property

The **ProductInfoFromScript** property of the **Installer** object returns the value of the specified attribute that is stored in an advertise script.

## Syntax

```
Script  
Installer.ProductInfoFromScript
```

## Remarks

The **ProductInfoFromScript** property uses the **MsiGetProductInfoFromScript** function.

## Examples

The following sample script demonstrates the use of the **ProductInfoFromScript** property .

```
Dim installer  
Set installer = CreateObject("WindowsInstaller.Installer")  
  
'  
' Create an advertise script for Orca  
'  
  
installer.CreateAdvertiseScript "\\products\public\orca\orca.advertise" "  
  
' Output ProductName Information From Script  
'  
  
MsgBox installer.ProductInfoFromScript("c:\scratch\orca.advertise")
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 4.5 on Windows Server 2003 and Windows XP
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

### [Installer](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer::ProductElevated Property

The **ProductElevated** property of the [Installer](#) object returns True if the product is managed or False if the product is not managed.

## Syntax

```
Script  
Installer.ProductElevated
```

## Remarks

The **ProductElevated** property uses the [MsilsProductElevated](#) function. The return of the property does not take into account the [AlwaysInstallElevated](#) policy.

## Examples

The following sample script demonstrates the use of the **ProductElevated** property .

```
Dim installer  
Set installer = CreateObject("WindowsInstaller.Installer")  
  
'  
' Install Orca tool per-machine  
'  
installer.InstallProduct "\\products\public\orca\orca.msi",  
  
'  
' Verify Orca is managed  
'  
  
Dim bManaged  
bManaged = installer.ProductElevated("{85F4CBCB-9BBC-4B50-A  
  
If bManaged Then  
    MsgBox "Success - Product Is Managed"  
Else
```

```
MsgBox "Failure - Product Is Not Managed"  
End If
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 4.5 on Windows Server 2003 and Windows XP
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

### [Installer](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.ProductsEx Property

The **ProductsEx** property returns a [RecordList](#) object that enumerates the list of products. This property calls [MsiEnumProductsEx](#).

## Syntax

```
Script  
Installer.ProductsEx
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[Installer](#)  
[MsiEnumProductsEx](#)  
[Product](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.Products Property

The **Products** property is a read-only property that returns a [StringList](#) object enumerating the set of all products installed or advertised for the current user and machine.

## Syntax

```
Script  
Installer.Products
```

## Remarks

To enumerate the products, an application iterates through the **StringList** object using a For Each construct. Because products are not ordered, any new product has an arbitrary index. This means that the function can return products in any order.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[MsiEnumProducts](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.ProductState Property Method

The **ProductState property** is a read-only property that returns the install state information for a product.

## Syntax

```
Script  
ProductState Property(  
    Product  
)
```

## Parameters

*Product*

Specifies the product code of the product.

## Return Value

This method does not return a value.

## Remarks

Returns one of the values shown in the following table.

<b>Installation state</b>	<b>Description</b>
msiInstallStateAbsent	The product is installed for a different user.
msiInstallStateDefault	The product is installed for the current user.
msiInstallStateAdvertised	The product is advertised but not installed.
msiInstallStateInvalidArg	An invalid parameter was passed to the function.
msiInstallStateUnknown	The product is neither advertised nor installed.
msiInstallStateBadConfig	The configuration data is corrupt.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

### [MsiQueryProductState](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer::ProvideAssembly Method

The **ProvideAssembly** method of the **Installer** object returns the installed path of an assembly.

## Syntax

```
Script
retVal = ProvideAssembly(
    assembly,
    appContext,
    installMode,
    assemblyInfo
)
```

## Parameters

*assembly*

The strong name of installed assembly that is to be queried.

*appContext*

Set to null for global assemblies. For private assemblies, set *appContext* to the full path of the application configuration file or to the full path of the executable file of the application to which the assembly has been made private.

*installMode*

Defines the installation mode. This parameter can be one of the following values.

Value	Meaning
msiInstallModeDefault 0	Provide the component and perform any installation necessary to provide the component.
msiInstallModeExisting -1	Provide the component only if the feature exists. This option will verify that the assembly exists.

msiInstallModeNoDetection -2	Provide the component only if the feature exists. This option does not verify that the assembly exists.
msiInstallModeNoSourceResolution -3	Provides the assembly only if the assembly is installed local.
Combination of the flags used by <b>ReinstallFeature</b>	Calls the <b>ReinstallFeature</b> method to reinstall the feature using this parameter for <i>ReinstallMode</i> , and then returns the assembly path.

### *assemblyInfo*

Assembly information and assembly type. Set to one of the following values.

Value	Meaning
msiProvideAssemblyNet 0	A .NET assembly.
msiProvideAssemblyWin32 1	A Win32 side-by-side assembly.

### Return Value

The path to the installed assembly.

### Remarks

The **ProvideAssembly** method uses the **MsiProvideAssembly** function.

### Examples

The following sample script demonstrates the use of the ProvideAssembly method.

```
Dim installer
Set installer = CreateObject("WindowsInstaller.Installer")

'
' ProvideAssembly - .NET global
'
MsgBox Installer.ProvideAssembly("System.Security,Version="

'
' ProvideAssembly - .NET private
'
MsgBox Installer.ProvideAssembly("Sample,Version=""1.0.0.0"

'
' ProvideAssembly - win32 global
'
MsgBox Installer.ProvideAssembly("Microsoft.MSXML2,publicKey
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 4.5 on Windows Server 2003 and Windows XP
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[Installer](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.ProvideComponent Method

The **ProvideComponent** method of the **Installer** object returns the full component path and performs any necessary installation. If necessary, the **ProvideComponent** method of the **Installer** object prompts for the source and increments the usage count for the feature.

## Syntax

```
Script
ProvideComponent(
    Product,
    Feature,
    Component,
    InstallMode
)
```

## Parameters

### *Product*

Specifies the product code of the product.

### *Feature*

Specifies the feature ID of the feature containing the component.

### *Component*

Specifies the component code.

### *InstallMode*

Defines the installation mode. This parameter can be one of the values shown in the following table.

Name	Meaning
msiInstallModeDefault 0	Provides the component path, performing any installation, if necessary.
msiInstallModeExisting -1	Provides the component path only if the feature exists;

	otherwise, returns an empty string. This mode verifies the existence of the component's key file.
msiInstallModeNoDetection -2	Provides the component path only if the feature exists. Otherwise, returns an empty string. This mode checks the component's registration but does not verify the existence of the component's key file.
msiInstallModeNoSourceResolution -3	Provides the component path only if the feature exists with an InstallState parameter of <i>msiInstallStateLocal</i> . This checks the component's registration but does not verify the existence of the component's key file.
combination of the msiReinstallMode flags	Calls <b>ReinstallFeature</b> to reinstall the feature using this parameter for the <i>ReinstallMode</i> parameter, and then provides the component.

## Return Value

This method does not return a value.

## Remarks

The **ProvideComponent** method combines the functionality of **UseFeature**, **ConfigureFeature**, and **ComponentPath**. The **ProvideComponent** method simplifies the calling sequence, but it also increments the usage count and should be used with caution to prevent

inaccurate usage counts. The **ProvideComponent** method also provides less flexibility than a series of individual calls to the methods and properties previously mentioned.

If the application is recovering from an unexpected situation, the application has probably already called **UseFeature** and incremented the usage count. In this case, the application should avoid incrementing the usage count by calling the **ConfigureFeature** method instead of the **ProvideComponent** method.

The `msiInstallModeExisting` option cannot be used in combination with `msiReinstallMode` flags.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

### [MsiProvideComponent](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.ProvideQualifiedComponent Method

The **ProvideQualifiedComponent** method of the [Installer](#) object returns the full component path and performs any necessary installation. If necessary, this method prompts for the source and increments the usage count for the feature.

## Syntax

```
Script
ProvideQualifiedComponent(
    Category,
    Qualifier,
    InstallMode
)
```

## Parameters

### *Category*

Specifies the component ID for the requested component. This may not be the GUID for the component itself but rather a server that provides the correct functionality, as in the ComponentId column of the [PublishComponent table](#).

### *Qualifier*

Specifies a qualifier into a list of advertising components (from [PublishComponent table](#)).

### *InstallMode*

Defines the installation mode. This parameter can be one of the values shown in the following table.

<b>InstallMode</b>	<b>Meaning</b>
msiInstallModeDefault 0	Provides the component, performing any necessary installation.

msiInstallModeExisting -1	Provides the component only if the feature exists; otherwise returns an empty string. This mode verifies the existence of the component's key file.
msiInstallModeNoDetection -2	Provides the component only if the feature exists; otherwise returns an empty string. This mode only checks that the component is registered but does not verify the existence of the component's key file.
msiInstallModeNoSourceResolution -3	Provides the component path only if the feature exists with an InstallState parameter of <i>msiInstallStateLocal</i> . This checks the component's registration but does not verify the existence of the component's key file.
combination of the msiReinstallMode flags	Calls <b>ReinstallFeature</b> to reinstall the feature using this parameter for the <i>ReinstallMode</i> parameter, and then provides the component.

## Return Value

This method does not return a value.

## Requirements

	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows
--	--

<b>Version</b>	Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

### [MsiProvideQualifiedComponent](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.QualifierDescription Property

The read-only **QualifierDescription** property returns a text string describing the qualified component. This localizable string is authored into the AppData column of the [PublishComponent table](#) and can be displayed to the user. The qualifier distinguishes multiple forms of the same component, such as a component that is implemented in multiple languages.

## Syntax

```
Script  
Installer.QualifierDescription
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[MsiEnumComponentQualifiers](#)  
Qualified Components

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.RegistryValue Method

The **RegistryValue** method of the **Installer** object reads information about a specified registry key or value. If the key or value specified does not exist, the method returns an error of 9, "Subscript out of Range."

## Syntax

```
Script
RegistryValue(
    root,
    key,
    value
)
```

## Parameters

*root*

In Windows NT 4.0, the registry root is either a numeric root key or a machine name as a string. Machine names are always strings. In Windows 95, Windows 98, or Windows Me, the registry root is a numeric root key only. You can only access HKLM on a remote machine.

Root	Meaning
HKEY_CLASSES_ROOT	0
HKEY_CURRENT_USER	1
HKEY_LOCAL_MACHINE	2
HKEY_USERS	3
HKEY_PERFORMANCE_DATA	4
HKEY_CURRENT_CONFIG	5
HKEY_DYN_DATA	6

---

*key*

A string containing the complete key path from the root. On Windows 95, Windows 98, or Windows Me, registry key class names are not supported and cannot be used.

*value*

This optional parameter designates which associated value to return for the specified key. The value is one of the values shown in the following table.

<b>Value</b>	<b>Meaning</b>
Missing or blank	Returns a Boolean designating whether the key exists.
String	Returns the data associated with the named value, fails if the value name is non-existent.
Positive integer	Returns the 1-based enumerated value name, it is empty if non-existent. This option uses the <b>RegEnumValue</b> function.
Negative integer	Returns the 1-based enumerated subkey name, this is empty if non-existent. This option uses the <b>RegEnumKey</b> function.
Zero integer	Returns the string class name for the designated key.
Empty string ""	Returns the default value of the registry key.

## **Return Value**

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.ReinstallFeature Method

The **ReinstallFeature** method of the [Installer](#) object reinstalls features or corrects problems with installed features.

## Syntax

```
Script
ReinstallFeature(
    Product,
    Feature,
    ReinstallMode
)
```

## Parameters

### *Product*

Specifies the product code of the product.

### *Feature*

Specifies the feature to be reinstalled. The parent feature or child feature of the specified feature is not reinstalled. To reinstall the parent or child feature, you must call the **ReinstallFeature** method for each separately or use the [ReinstallProduct](#) method.

### *ReinstallMode*

Specifies the type of reinstallation. This parameter can be one or more of the following values.

Value	Meaning
msiReinstallModeFileMissing	Reinstalls only if the file is missing.
msiReinstallModeFileOlderVersion	Reinstalls if the file is missing or is an older version.
msiReinstallModeFileEqualVersion	Reinstalls if the file is missing or is an equal or older version.

msiReinstallModeFileExact	Reinstalls if the file is missing or is not an exact version.
msiReinstallModeFileVerify	Checks sum executables, and reinstalls if they are missing or corrupt.
msiReinstallModeFileReplace	Reinstalls all files regardless of version.
msiReinstallModeUserData	Ensures required per=user registry entries.
msiReinstallModeMachineData	Ensures required per=machine registry entries.
msiReinstallModeShortcut	Validates shortcuts.
msiReinstallModePackage	Uses the recache source to install the package.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

### [MsiReinstallFeature](#)

[Installation and Configuration Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.ReinstallProduct Method

The **ReinstallProduct** method of the **Installer** object reinstalls a product or corrects installation problems in an installed product.

## Syntax

```
Script
ReinstallProduct(
    Product,
    ReinstallMode
)
```

## Parameters

### *Product*

Specifies the product code of the product.

### *ReinstallMode*

Specifies the type of reinstallation. This parameter can be one or more of the following values.

Value	Meaning
msiReinstallModeFileMissing	Reinstalls only if the file is missing.
msiReinstallModeFileOlderVersion	Reinstalls if the file is missing or is an older version.
msiReinstallModeFileEqualVersion	Reinstalls if the file is missing or is an equal or older version.
msiReinstallModeFileExact	Reinstalls if the file is missing or is not an exact version.
msiReinstallModeFileVerify	Checks sum executables, and reinstalls if they are missing or corrupt.

msiReinstallModeFileReplace	Reinstalls all files regardless of version.
msiReinstallModeUserData	Ensures required per=user registry entries.
msiReinstallModeMachineData	Ensures required per=machine registry entries.
msiReinstallModeShortcut	Validates shortcuts.
msiReinstallModePackage	Uses the recache source to install the package.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

### [MsiReinstallProduct](#)

[Installation and Configuration Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.RelatedProducts Property

The read-only **RelatedProducts** property returns a **StringList** object enumerating the set of all products installed or advertised for the current user and machine with a specified **UpgradeCode** property in their [Property table](#).

## Syntax

```
Script  
Installer.RelatedProducts
```

## Remarks

To enumerate the related products, an application iterates through the **StringList** using a For Each construct. Because related products are not ordered, any new related product has an arbitrary index. This means that the function can return related products in any order.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[MsiEnumRelatedProducts](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.RemovePatches Method

The **RemovePatches** method removes one or more patches to products eligible to receive the patch. The **RemovePatches** method calls [MsiRemovePatches](#).

## Syntax

```
Script
RemovePatches(
    PatchList,
    ProductCode,
    UninstallType,
    PropertyList
)
```

## Parameters

### *PatchList*

A string that contains a semicolon delimited list of patches to remove. Each patch can be represented by either the full path to the patch package or by patch GUID. This parameter is required.

### *ProductCode*

A string with the GUID of the product from which the patches are to be removed. This parameter is required.

### *UninstallType*

An integer value that specifies the type of patch removal. This parameter must be `msiInstallTypeSingleInstance`.

### *PropertyList*

A string that specifies the Property=Value pairs to include. This parameter is optional.

## Return Value

This method does not return a value.

## Remarks

See [Uninstalling Patches](#) for an example that demonstrates how an application can remove a patch from all products that are available to the user.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

[ProductCode](#)

[MsiRemovePatches](#)

[Uninstalling Patches](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.ShortcutTarget Property

The **ShortcutTarget** property of the **Installer** object examines a shortcut and returns its product, feature name, and component if available.

## Syntax

```
Script  
Installer.ShortcutTarget
```

## Remarks

ShortcutTarget returns a **Record object** that contains three fields:

- Field 1 is a GUID for the product code of the shortcut, if available. This field can be null.
- Field 2 is the Feature ID of the shortcut, if available. This field can be null.
- Field 3 is a GUID for the component code, if available. This field can be null.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

**MsiGetFeatureState**  
**MsiGetComponentState**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.SummaryInformation Property

The **SummaryInformation** property of the **Installer** object returns a **SummaryInfo** object that can be used to examine, update, and add properties to the summary information stream of a package or transform.

## Syntax

```
Script  
Installer.SummaryInformation
```

## Remarks

If a value of *maxProperties* greater than 0 is used to open an existing summary information stream, the **Persist** method must be called before closing the object. Failing to do this loses the existing stream information.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.UILevel Property

The **UILevel** property of the **Installer** object is a read-write property that indicates the type of user interface to be used when opening and processing subsequent packages within the current process space.

## Syntax

Script

```
Installer.UILevel
```

## Remarks

User interface level	Value	Description
msiUILevelNoChange	0	Does not change UI level.
msiUILevelDefault	1	Uses default UI level.
msiUILevelNone	2	Silent installation.
msiUILevelBasic	3	Simple progress and error handling.
msiUILevelReduced	4	Authored UI and wizard dialog boxes suppressed.
msiUILevelFull	5	Authored UI with wizards, progress, and errors.
msiUILevelHideCancel	32	If combined with the msiUILevelBasic value, the installer shows progress dialog boxes but does not display a <b>Cancel</b> button on the dialog box to prevent users from canceling the installation.
msiUILevelProgressOnly	64	If combined with the msiUILevelBasic value, the installer displays progress dialog boxes but does not display any modal dialog boxes or error dialog boxes.
msiUILevelEndDialog	128	If combined with any above value, the installer displays a modal dialog box at the

	end of a successful installation or if there has been an error. No dialog box is displayed if the user cancels.
--	---

See also, [Determining UI Level from a Custom Action](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.UseFeature Method

The **UseFeature** method of the **Installer** object increments the usage count for a particular feature and returns the installation state for that feature. This method should be used to indicate an application's intent to use a feature.

## Syntax

```
Script
UseFeature(
    Product,
    Feature,
    InstallMode
)
```

## Parameters

### *Product*

Specifies the product code of the product.

### *Feature*

Identifies the feature to be used.

### *InstallMode*

This parameter must be *msiInstallModeNoDetection*.

## Return Value

This method does not return a value.

## Remarks

The **UseFeature** method should only be used on features known to be published. The application should determine the status of the feature by calling either the **FeatureState** property or **Features** property or their API equivalents.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

## See Also

### [MsiUseFeatureEx](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer.Version Property

The **Version** property of the **Installer** object is a read-only property that is the string representation of the current version of Windows Installer. The string is returned in the following form.

*major.minor.build.update*

## Syntax

Script

```
Installer.Version
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IInstaller is defined as 000C1090-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patch Object

The **Patch** object represents a unique instance of a patch that has been registered or applied.

The object can be instantiated with the **Patch** property as "WindowsInstaller.Installer.Patch(*PatchCode*, *ProductCode*, *UserSid*, *Context*)". For a machine context, the *UserSid* parameter must be an empty string. The *ProductCode* can be set to an empty string for patches that are registered only and not yet applied to any product. The *ProductCode* can be set to an empty string when only reading or updating a patch's source list information.

## Methods

The **Patch** object defines the following methods.

Method	Description
<b>SourceListAddSource</b>	Add a network or URL source to the source list.
<b>SourceListAddMediaDisk</b>	Add a disk to the set of registered disks.
<b>SourceListClearSource</b>	Remove a network or URL source from the source list.
<b>SourceListClearMediaDisk</b>	Remove a disk from the set of registered disks from the source list.
<b>SourceListClearAll</b>	Clears the complete source list of the specified type of sources.
<b>SourceListForceResolution</b>	Clears the last used source from the source list. This forces a source list resolution the next time the source is required.

## Properties

The **Patch** object defines the following properties.

<b>Property</b>	<b>Access type</b>	<b>Description</b>
<b>Context</b>	Read-only	Context of this patch instance is an MSIINSTALLCONTEXT value. This is a read-only property.
<b>MediaDisks</b>	Read-only	Enumerates all the media disks for this patch instance. This is a read-only property.
<b>PatchCode</b>	Read-only	Returns the patch code. This is a read-only property.
<b>PatchProperty</b>	Read-only	Gets property information about a specific patch applied to a specific instance of the product. This is a read-only property.
<b>ProductCode</b>	Read-only	Returns the product code. This is a read-only property.
<b>SourceListInfo</b>	Read-only	Gets and sets the source information properties. This is a read or write property.
<b>Sources</b>	Read-only	Enumerates all the sources for this patch instance. This is a read-only property.
<b>State</b>	Read-only	Installation state of the patch. This is a read-only property.
<b>UserSid</b>	Read-only	Returns the User SID, under the account this patch instance is available. This is a read-only property.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IPatch is defined as 000C10A1-0000-0000-C000-000000000046

## See Also

[Windows Installer Scripting Examples](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patch.SourceListAddMediaDisk Method

The **SourceListAddMediaDisk** method adds a disk to the set of registered disks. Accepts *Diskid*, *VolumeLabel* and *DiskPrompt* as parameters. This method calls on [MsiSourceListAddMediaDisk](#).

## Syntax

```
Script
SourceListAddMediaDisk(
    Diskid,
    VolumeLabel,
    DiskPrompt
)
```

## Parameters

### *Diskid*

This parameter provides the ID of the disk being added or updated.

### *VolumeLabel*

This parameter provides the label of the disk being added or updated. An update, overwrites the existing volume label in the registry. To change the disk prompt only, get the existing registered volume label and provide it along with the new disk prompt. A NULL or empty string registers an empty string (0 bytes in length) as the volume label.

### *DiskPrompt*

This parameter provides the disk prompt of the disk being added or updated. An update overwrites the existing disk prompt in the registry. To change the volume label only, get the existing disk prompt from the registry and provide it with the new volume label. A NULL or empty string registers an empty string (0 bytes in length) as the disk prompt.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IPatch is defined as 000C10A1-0000-0000-C000-000000000046

## See Also

### [Patch](#)

### [MsiSourceListAddMediaDisk](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patch.SourceListAddSource Method

The **SourceListAddSource** method adds a network or URL source. Accepts *SourcePath*, *Type* and *Index* as parameters. This method calls [MsiSourceListAddSourceEx](#).

## Syntax

```
Script
SourceListAddSource(
    Type,
    SourcePath,
    Index
)
```

## Parameters

### *Type*

Type of source to be added: MSISOURCETYPE\_NETWORK or MSISOURCETYPE\_URL.

### *SourcePath*

Path to the source to be added.

### *Index*

If **SourceListAddSource** is called with a new source and *Index* set to 0, the installer adds the source to the end of the source list.

If this function is called with a source already existing in the source list and *Index* is set to 0, the installer retains the source's existing index.

If the function is called with an existing source in the source list and *Index* is set to a non-zero value, the source is removed from its current location in the list and inserted at the position specified by *Index*, before any source that already exists at that position.

If the function is called with a new source and *Index* is set to a non-zero value, the source is inserted at the position specified by *Index*, before any source that already exists at that position. The index value for all sources in the list after the index specified by *Index* are

updated to ensure unique index values and the preexisting order is guaranteed to remain unchanged.

If *Index* is greater than the number of sources in the list, the source is placed at the end of the list with an index value one larger than any existing source.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IPatch is defined as 000C10A1-0000-0000-C000-000000000046

## See Also

**Patch**  
**MsiSourceListAddSourceEx**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patch.SourceListClearMediaDisk Method

The **SourceListClearMediaDisk** method of the **Patch** object removes a specified disk from the set of registered disks for a patch. Accepts *Diskid* as a parameter. This method calls **MsiSourceListClearMediaDisk**.

## Syntax

```
Script
SourceListClearMediaDisk(
    Diskid
)
```

## Parameters

*Diskid*

This parameter provides the ID of the disk to remove.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IPatch is defined as 000C10A1-0000-0000-C000-000000000046

## See Also

[Patch](#)

[MsiSourceListClearMediaDisk](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patch.SourceListClearSource Method

The **SourceListClearSource** method removes a network or URL source. This method calls [MsiSourceListClearSource](#).

## Syntax

```
Script  
SourceListClearSource(  
    Type,  
    SourcePath  
)
```

## Parameters

### *Type*

Type of source to be removed.

MSISOURCETYPE\_NETWORK  
MSISOURCETYPE\_URL

### *SourcePath*

Path to the source to be removed.

## Return Value

This method does not return a value.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000

**DLL**

Msi.dll

**IID**

IID\_IPatch is defined as 000C10A1-0000-0000-  
C000-0000000000046

## See Also

[Patch](#)

[MsiSourceListClearSource](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patch.SourceListClearAll Method

The **SourceListClearAll** method of the **Patch** object clears the complete source list of all sources of the specified type for a patch. Accepts *Type* as a parameter. This method calls **MsiSourceListClearAllEx**.

## Syntax

```
Script  
SourceListClearAll(  
    Type  
)
```

## Parameters

*Type*

The type of source type, such as network, URL or media.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IPatch is defined as 000C10A1-0000-0000-C000-000000000046

## See Also

**Patch**  
**MsiSourceListClearAllEx**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patch.SourceListForceResolution Method

The **SourceListForceResolution** method clears the last used source property. This forces the installer to search the source list for a valid patch source the next time the patch source is required. For example, the installer requires the patch source to perform an installation or reinstallation when the local cache copy of the patch is missing. This method calls **MsiSourceListForceResolution**.

## Syntax

```
Script  
SourceListForceResolution( )
```

## Parameters

This method has no parameters.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IPatch is defined as 000C10A1-0000-0000-C000-000000000046

## See Also

[Patch  
MsiSourceListForceResolution](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patch.Context Property

The **Context** property returns the context of this patch. This is a read-only property.

## Syntax

```
Script  
Patch.Context
```

## Remarks

This property returns one of the following values.

Context	Value	Meaning
MSIINSTALLCONTEXT_USERMANAGED	1	Patch under managed context.
MSIINSTALLCONTEXT_USER	2	Patch under unmanaged context.
MSIINSTALLCONTEXT_MACHINE	4	Patch under machine context.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IPatch is defined as 000C10A1-0000-0000-C000-000000000046

## See Also

### [Patch Object](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patch.MediaDisks Property

The **MediaDisks** property enumerates all the media disks for this product instance. This property calls the [MsiSourceListEnumMediaDisks](#). Returns the disk information as array of [Record](#) objects. This is a read-only property.

## Syntax

Script

```
Patch.MediaDisks
```

## Remarks

In each record the first field contains the disk Id, the second field contains the volume label and the third field contains the disk prompt registered for the disk.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IPatch is defined as 000C10A1-0000-0000-C000-000000000046

## See Also

[Patch](#)  
[MsiSourceListEnumMediaDisks](#)  
[Record](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patch.PatchCode Property

The **PatchCode** property returns the patch code GUID of the patch. This is a read-only property.

## Syntax

Script

```
Patch.PatchCode
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IPatch is defined as 000C10A1-0000-0000-C000-000000000046

## See Also

### [Patch](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patch.PatchProperty Method

The **PatchProperty** property gets information about a specific patch applied to a specific instance of the product. This property calls [MsiGetPatchInfoEx](#). This is a read-only property.

## Syntax

```
Script  
PatchProperty(  
    szProperty  
)
```

## Parameters

*szProperty*

The *szProperty* parameter can be one of the following values.

Name	Meaning
LocalPackage	Get the cached patch file used by the product.
Transforms	Get the set of patch transforms applied to the product by the last patch installation. This value may not be available for per-user-unmanaged applications if the user is not logged in to the computer.
InstallDate	Get the date when the patch was applied to the product.
Uninstallable	Returns "1" if the patch is marked as possible to uninstall from the product. In this case, the installer can still block the uninstallation if this patch is required by another patch that cannot be uninstalled.
State	Returns "1" if this patch is currently applied to the product. Returns "2" if this patch has been superseded by another patch. Returns "4" if this patch has been made obsolete by another patch. These values correspond to the constants used by the <i>dwFilter</i> parameter of <a href="#">MsiEnumPatchesEx</a> .

DisplayName	Get the registered display name for the patch. For patches that do not include the DisplayName property in the <a href="#">MsiPatchMetadata</a> table, the returned display name is an empty string ("").
MoreInfoURL	Get the registered support information URL for the patch. For patches that do not include the MoreInfoURL property in the MsiPatchMetadata table, the returned support information URL is an empty string ("").

## Return Value

This method does not return a value.

## Remarks

This method can return `ERROR_UNKNOWN_PATCH`, if the [Patch](#) object is initialized with an empty string for *ProductCode*.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IPatch is defined as 000C10A1-0000-0000-C000-000000000046

## See Also

[Patch](#)

[MsiEnumPatchesEx](#)

## **MsiGetPatchInfoEx**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patch.ProductCode Property

The **ProductCode** property of the [Patch](#) object returns the **ProductCode** GUID of the product. This is a read-only property.

## Syntax

```
Script  
Patch.ProductCode
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IPatch is defined as 000C10A1-0000-0000-C000-000000000046

## See Also

### [Patch](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patch.SourceListInfo Property

The **SourceListInfo** property of the **Patch** object gets and sets the source information properties for a patch. This property call **MsiSourceListGetInfo** or **MsiSourceListSetInfo**. This is a read or write property.

## Syntax

Script

```
Patch.SourceListInfo
```

## Remarks

Not all properties that can be retrieved can be set. The *szProperty* parameter can be one of the following values.

Property	Can set?	Meaning
MediaPackagePath	Y	The path relative to the root of the installation media.
DiskPrompt	Y	The prompt template used when prompting the user for installation media.
LastUsedSource	Y	The most recently used source location for the patch. When you set this property, prefix the source location with "n;" for a network source or "u;" for URL type. For example, use "n;\\scratch\\scratch\\test" or "u;http://microsoft.com/Patches/Office/SP1".
LastUsedType	N	"n" if the last-used source is a network location. "u" if the last used source was a URL location. "m" if the last used source was media. Empty string ("" ) if there is no last used source.
PackageName	Y	The name of the Windows Installer package or patch package on the source.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IPatch is defined as 000C10A1-0000-0000-C000-000000000046

## See Also

### [Patch](#)

### [MsiSourceListGetInfo](#)

### [MsiSourceListSetInfo](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patch.Sources Property

The **Sources** property enumerates all the sources for the patch instance. This property calls [MsiSourceListEnumSources](#) and returns an array of strings, and accepts the source type as argument. This is a read-only property.

## Syntax

```
Script  
Patch.Sources
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IPatch is defined as 000C10A1-0000-0000-C000-000000000046

## See Also

[Patch](#)  
[MsiSourceListEnumSources](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patch.State Property

The **State** property returns the installation state of this instance of the patch. This is a read-only property.

## Syntax

```
Script  
Patch.State
```

## Remarks

This property returns one of the following values.

Installation State	Meaning
MSIPATCHSTATE_APPLIED	Patch is applied to this product instance.
MSIPATCHSTATE_SUPERSEDED	Patch is applied to this product instance but is superseded.
MSIPATCHSTATE_OBSOLETED	Patch is applied in this product instance but obsolete.

This method can return `ERROR_UNKNOWN_PATCH`, if the **Patch** object is initialized with an empty string for *ProductCode*.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll

**IID**

IID\_IPatch is defined as 000C10A1-0000-0000-C000-000000000046

**See Also**

**Patch**  
**MsiGetPatchInfoEx**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patch.UserSid Property

The **UserSid** property returns the user security identifier (SID) under which this patch instance is available. This is a read-only property.

## Syntax

```
Script  
Patch.UserSid
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IPatch is defined as 000C10A1-0000-0000-C000-000000000046

## See Also

### [Patch](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Product Object

The **Product** object represents a unique instance of a product that is either advertised, installed or unknown.

The object can be instantiated with the **Product** property as "WindowsInstaller.Installer.Product(*ProductCode*, *UserSid*, *Context*)". *UserSid* must be NULL for per-machine context. *UserSid* can be null to specified current user, when context is not per-machine. *ProductCode* and *Context* parameters are required.

## Methods

The **Product** object defines the following methods.

Method	Description
<b>SourceListAddSource</b>	Add a network or URL source to the source list.
<b>SourceListAddMediaDisk</b>	Add a disk to the set of registered disks.
<b>SourceListClearSource</b>	Remove a network or URL source from the source list.
<b>SourceListClearMediaDisk</b>	Remove a disk from the set of registered disks from the source list.
<b>SourceListClearAll</b>	Clears the complete source list of the specified type of sources.
<b>SourceListForceResolution</b>	Clears the last used source. This forces a source list resolution the next time the source is required.

## Properties

The **Product** object defines the following properties.

<b>Property</b>	<b>Access type</b>	<b>Description</b>
<b>ComponentState</b>	Read-only	The state of a specified component for this product instance. This is a read-only property.
<b>Context</b>	Read-only	Context of this product instance as an MSIINSTALLCONTEXT value. This is a read-only property.
<b>FeatureState</b>	Read-only	The state of a specified feature for this product instance. This is a read-only property.
<b>InstallProperty</b>	Read-only	The value of a specified property. This is a read-only property.
<b>MediaDisks</b>	Read-only	Enumerates all the media disks for this product instance. This is a read-only property.
<b>ProductCode</b>	Read-only	Returns the product code. This is a read-only property.
<b>SourceListInfo</b>	Read-only	Get and set the source information properties. This is a read or write property.
<b>Sources</b>	Read-only	Enumerates all the sources for this product instance. This is a read-only property.
<b>State</b>	Read-only	Installation state of the product. This is a read-only property.
<b>UserSid</b>	Read-only	Returns the User SID, under which account this product instance is available. This is a read-only property.

## Requirements



<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IProduct is defined as 000C10A0-0000-0000-C000-000000000046

## See Also

[Windows Installer Scripting Examples](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Product.SourceListAddMediaDisk Method

The **SourceListAddMediaDisk** method adds a disk to the set of registered disks. Accepts *Diskid*, *VolumeLabel* and *DiskPrompt* as parameters. This method calls on [MsiSourceListAddMediaDisk](#).

## Syntax

```
Script
SourceListAddMediaDisk(
    Diskid,
    VolumeLabel,
    DiskPrompt
)
```

## Parameters

### *Diskid*

This parameter provides the ID of the disk being added or updated.

### *VolumeLabel*

This parameter provides the label of the disk being added or updated. An update overwrites the existing volume label in the registry. To change the disk prompt only, get the existing registered volume label and provide it along with the new disk prompt. An empty string for this parameter registers an empty string (0 bytes in length) as the volume label.

### *DiskPrompt*

This parameter provides the disk prompt of the disk being added or updated. An update overwrites the existing disk prompt in the registry. To change the volume label only, get the existing disk prompt from the registry and provide it with the new volume label. An empty string for this parameter registers an empty string (0 bytes in length) as the disk prompt.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IProduct is defined as 000C10A0-0000-0000-C000-000000000046

## See Also

### Product

### [MsiSourceListAddMediaDisk](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Product.SourceListAddSource Method

The **SourceListAddSource** method adds a network or URL source. Accepts *SourcePath*, *Type*, and *Index* as parameters. This method calls [MsiSourceListAddSourceEx](#).

## Syntax

```
Script
SourceListAddSource(
    Type,
    SourcePath,
    Index
)
```

## Parameters

### *Type*

Type of source to be added: MSISOURCETYPE\_NETWORK or MSISOURCETYPE\_URL.

### *SourcePath*

Path to the source to be added.

### *Index*

If **SourceListAddSource** is called with a new source and *Index* is set to 0, the installer adds the source to the end of the source list.

If this function is called with a source already existing in the source list and *Index* is set to 0, the installer retains the source's existing index.

If the function is called with an existing source in the source list and *Index* is set to a non-zero value, the source is removed from its current location in the list and inserted at the position specified by *Index*, before any source that already exists at that position.

If the function is called with a new source and *Index* is set to a non-zero value, the source is inserted at the position specified by *Index*,

before any source that already exists at that position. The index value for all sources in the list after the index specified by *Index* are updated to ensure unique index values and the pre-existing order is guaranteed to remain unchanged.

If *Index* is greater than the number of sources in the list, the source is placed at the end of the list with an index value one larger than any existing source.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IProduct is defined as 000C10A0-0000-0000-C000-000000000046

## See Also

### Product

### [MsiSourceListAddSourceEx](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Product.SourceListClearSource Method

The **SourceListClearSource** method removes a network or URL source. Accepts *Type*, the source type, and *SourcePath*, the source path, as parameters to be removed. This method calls the [MsiSourceListClearSource](#).

## Syntax

```
Script
SourceListClearSource(
    Type,
    SourcePath
)
```

## Parameters

*Type*

Type of source to be removed: MSISOURCETYPE\_NETWORK or MSISOURCETYPE\_URL.

*SourcePath*

Path to the source to be removed.

## Return Value

This method does not return a value.

## Requirements

**Version**

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000

<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IProduct is defined as 000C10A0-0000-0000-C000-000000000046

## See Also

[Product](#)

[MsiSourceListClearSource](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Product.SourceListClearAll Method

The **SourceListClearAll** method the **Product** object removes all sources for the product. The type of source to remove can be specified.

## Syntax

```
Script
SourceListClearAll(
    Type
)
```

## Parameters

*Type*

Type of source to be removed: MSISOURCETYPE\_MEDIA, MSISOURCETYPE\_NETWORK or MSISOURCETYPE\_URL.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IProduct is defined as 000C10A0-0000-0000-C000-000000000046

## See Also

**Product**  
**MsiSourceListClearSource**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Product.SourceListClearMediaDisk Method

The **SourceListClearMediaDisk** method of the **Product** object removes a specified disk from the set of registered disks for a product. Accepts *Diskid* as a parameter. This method calls [MsiSourceListClearMediaDisk](#).

## Syntax

```
Script
SourceListClearMediaDisk(
    Diskid
)
```

## Parameters

*Diskid*

This parameter provides the ID of the disk to remove.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IProduct is defined as 000C10A0-0000-0000-C000-000000000046

## See Also

### Product

[MsiSourceListClearMediaDisk](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Product.SourceListForceResolution Method

The **SourceListForceResolution** method clears the LastUsedSource property. This forces the installer to search the source list for a valid product source the next time a source is required, such as when the installer performs an installation or a reinstallation, or when the path is required for a component set to run from the source.

This method calls [MsiSourceListForceResolution](#).

## Syntax

```
Script  
SourceListForceResolution( )
```

## Parameters

This method has no parameters.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IProduct is defined as 000C10A0-0000-0000-C000-000000000046

## See Also

### Product

[MsiSourceListForceResolution](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Product.SourceListInfo Property

The **SourceListInfo** property of the **Product** object gets and sets the source information properties for a product. This property calls **MsiSourceListGetInfo** or **MsiSourceListSetInfo**. This is a read or write property.

## Syntax

Script

```
Product.SourceListInfo
```

## Remarks

Not all properties that can be retrieved can be set. The *szProperty* parameter can be one of the following values.

Property	Can set?	Meaning
MediaPackagePath	Y	The path relative to the root of the installation media.
DiskPrompt	Y	The prompt template used when prompting the user for installation media.
LastUsedSource	Y	The most recently used source location for the product. When you set this property, prefix the source location with "n;" for a network source or "u;" for URL type. For example, "n;\scratch\scratch\MySource" and "u;http://MyServer/MyFolder/MySource".
LastUsedType	N	"n" if the last-used source is a network location. "u" if the last used source was a URL location. "m" if the last used source was media. Empty string ("" ) if there is no last used source.
PackageName	Y	The name of the Windows Installer package or patch package on the source.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IProduct is defined as 000C10A0-0000-0000-C000-000000000046

## See Also

### Product

[MsiSourceListGetInfo](#)

[MsiSourceListSetInfo](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Product.ComponentState Method

The **ComponentState** property is the installation state of the component for the instance of this product. This is a read-only property.

This property calls [MsiQueryComponentState](#), with the ProductCode, UserSid, and Context of the object. The component Id GUID is provided as a parameter.

## Syntax

```
Script  
ComponentState(  
    ID  
)
```

## Parameters

*ID*

Component code GUID of the component as found in the ComponentID column of the [Component table](#).

## Return Value

This method does not return a value.

## Remarks

If the call succeeds, the property contains the value as a **DWORD**.

State	Meaning
INSTALLSTATE_LOCAL	The component is installed locally.
INSTALLSTATE_SOURCE	The component is installed to run from the source.

If the call fails, the property contains an error code from

## MsiQueryComponentState.

Error	Meaning
ERROR_ACCESS_DENIED	The calling process must have administrative privileges to get information for a user other than the current user.
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_SUCCESS	The function completed successfully.
ERROR_UNKNOWN_COMPONENT	The component ID does not identify a known component.
ERROR_UNKNOWN_PRODUCT	The product code does not identify a known product.
ERROR_FUNCTION_FAILED	An unexpected internal failure.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IProduct is defined as 000C10A0-0000-0000-C000-000000000046

## See Also

**Product**  
**MsiQueryComponentState**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Product.ProductCode Property

The **ProductCode** property of the **Product** object returns the **ProductCode** GUID of the product. This is a read-only property.

## Syntax

Script

```
Product.ProductCode
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IProduct is defined as 000C10A0-0000-0000-C000-000000000046

## See Also

### [Product](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Product.InstallProperty Method

The **InstallProperty** property is the value of the property for the instance of this product.

This property calls the [MsiGetProductInfoEx](#) function, with the *ProductCode*, *UserSid* and *Context* of the **Product** object and the requested property as a parameter.

## Syntax

```
Script
InstallProperty(
    property
)
```

## Parameters

*property*

Specifies the property to be retrieved. The properties in the following list can only be retrieved from applications that are already installed. Note that [required](#) properties are guaranteed to be available, but other properties are available only if that property has been set. See the indicated links to the installer [properties](#) for information about how each property is set.

Installed properties	Meaning
INSTALLPROPERTY_PRODUCTSTATE	State of the prod in string form as Advertised and " installed.
INSTALLPROPERTY_HELPLINK	Support link. Fo information, see <a href="#">ARPHELPLIN</a> .
INSTALLPROPERTY_HELPTELEPHONE	Support telephor information, see

	<a href="#">ARPHELPTEL</a> property.
INSTALLPROPERTY_INSTALLDATE	Installation date.
INSTALLPROPERTY_INSTALLEDPRODUCTNAME	Installed product more information: <a href="#">ProductName</a> p
INSTALLPROPERTY_INSTALLLOCATION	Installation location more information: <a href="#">ARPINSTALLL</a> property.
INSTALLPROPERTY_INSTALLSOURCE	Installation source information, see <a href="#">SourceDir</a> prop
INSTALLPROPERTY_LOCALPACKAGE	Local cached pa
INSTALLPROPERTY_PUBLISHER	Publisher. For m information, see <a href="#">Manufacturer</a> p
INSTALLPROPERTY_URLINFOABOUT	URL information information, see <a href="#">ARPURLINFO</a> property.
INSTALLPROPERTY_URLUPDATEINFO	URL update info more information: <a href="#">ARPURLUPDA</a> property.
INSTALLPROPERTY_VERSIONMINOR	Minor product v derived from the <a href="#">ProductVersion</a>
INSTALLPROPERTY_VERSIONMAJOR	Major product v derived from the

	<b>ProductVersion</b>
INSTALLPROPERTY_VERSIONSTRING	Product version. information, see <b>ProductVersion</b>

To retrieve the product ID, registered owner, or registered company from applications that are already installed, set *property* to one of the following text string values.

<b>Value</b>	<b>Description</b>
ProductID	The product identifier. For more information, see the <b>ProductID</b> property.
RegCompany	The company registered to use this product.
RegOwner	The owner registered to use this product.

To retrieve the instance type of the product, set *property* to the following value. This property is available for advertised or installed products.

<b>Value</b>	<b>Description</b>
InstanceType	A missing value or a value of 0 indicates a normal product installation. A value of 1 indicates a product installed using a multiple instance transform and the <b>MSINERINSTANCE</b> property. Available with the installer running Windows Server 2003 or Windows XP with SP1. For more information, see <a href="#">Installing Multiple Instances of Products and Patches</a> .

The properties in the following list can also be retrieved from applications that are advertised. These properties cannot be retrieved for product instances that are installed under a per-user-unmanaged context for user accounts other than current user account.

---

Advertised properties	Description
INSTALLPROPERTY_TRANSFORMS	Transforms.
INSTALLPROPERTY_LANGUAGE	Product language.
INSTALLPROPERTY_PRODUCTNAME	Human readable–product name. For more information, see the <a href="#">ProductName</a> property.
INSTALLPROPERTY_ASSIGNMENTTYPE	Equals zero (0) if the product is advertised installed per-user. Equals one (1) if the product is advertised installed per-computer for all users.
INSTALLPROPERTY_PACKAGECODE	Identifier of the package this product was installed from. For details, see <a href="#">Package Codes</a> .
INSTALLPROPERTY_VERSION	Product version derived from the <a href="#">ProductVersion</a> property.
INSTALLPROPERTY_PRODUCTICON	Primary icon for the package. For more information, see the <a href="#">ARPPRODUCTICON</a> property.
INSTALLPROPERTY_PACKAGENAME	Name of the original installation package.
INSTALLPROPERTY_AUTHORIZED_LUA_APP	A value of 1 indicates product that can be serviced by non-

administrators using [User Account Control \(UAC\) Patching](#). A missing value or a value of 0 indicates least-privilege patching is not enabled. Available with Windows Installer 3.0 and later.

## Return Value

This method does not return a value.

## Remarks

If the call succeeds, the property contains the value as a string.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IProduct is defined as 000C10A0-0000-0000-C000-000000000046

## See Also

### [Product](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Product.Context Property

The **Context** property returns the context of this product. This is a read-only property.

## Syntax

Script

```
Product.Context
```

## Remarks

This property can return one of the following values.

Context	Value	Meaning
MSIINSTALLCONTEXT_USERMANAGED	1	Products under managed context.
MSIINSTALLCONTEXT_USER	2	Products under unmanaged context.
MSIINSTALLCONTEXT_MACHINE	4	Products under machine context.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IProduct is defined as 000C10A0-0000-0000-C000-000000000046

## See Also

### Product

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Product.FeatureState Method

The **FeatureState** property is the installation state of the feature for the instance of this product. This is a read-only property.

This property calls **MsiQueryFeatureStateEx**, with the *ProductCode*, *UserSid* and *Context* of the object. The feature Id is provided as a parameter.

## Syntax

```
Script
FeatureState(
    FeatureId
)
```

## Parameters

*FeatureId*

Feature Id appearing in the Feature column of the [Feature Table](#).

## Return Value

This method does not return a value.

## Remarks

If the call succeeds, the property contains the value as a **DWORD**.

State	Meaning
INSTALLSTATE_ADVERTISED	This feature is advertised.
INSTALLSTATE_LOCAL	The feature is installed locally.
INSTALLSTATE_SOURCE	The feature is installed to run from source.

If the call fails, the property contains an error code from

## MsiQueryFeatureStateEx.

Error	Meaning
ERROR_ACCESS_DENIED	The calling process must have administrative privileges to get information for a product installed for a user other than the current user.
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_SUCCESS	The function completed successfully.
ERROR_UNKNOWN_FEATURE	The feature ID does not identify a known feature.
ERROR_UNKNOWN_PRODUCT	The product code does not identify a known product.
ERROR_FUNCTION_FAILED	An unexpected internal failure.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IPProduct is defined as 000C10A0-0000-0000-C000-000000000046

## See Also

### Product

## **MsiQueryFeatureStateEx**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Product.MediaDisks Property

The **MediaDisks** property enumerates all the media disks for this product instance. This property calls the [MsiSourceListEnumMediaDisks](#). Returns the disk information as an array of [Record](#) objects. This is a read-only property.

## Syntax

Script

```
Product.MediaDisks
```

## Remarks

In each record, the first field contains the disk Id, the second field contains the volume label and the third field contains the disk prompt registered for the disk.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IProduct is defined as 000C10A0-0000-0000-C000-000000000046

## See Also

[Product](#)  
[MsiSourceListEnumMediaDisks](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Product.Sources Property

The **Sources** property enumerates all the sources for the product instance. This property calls [MsiSourceListEnumSources](#) and returns the array of strings, and accepts the source type as argument. The source type can be MSISOURCETYPE\_NETWORK or MSISOURCETYPE\_URL. This is a read-only property.

## Syntax

Script

```
Product.Sources
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IProduct is defined as 000C10A0-0000-0000-C000-000000000046

## See Also

### Product

### [MsiSourceListEnumSources](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Product.State Property

The **State** property returns the installation state of this instance of the product. This is a read-only property.

## Syntax

```
Script  
Product.State
```

## Remarks

This property returns one of the following values.

Installation State	Meaning
INSTALLSTATE_DEFAULT	Instance of product installed locally.
INSTALLSTATE_ADVERTISED	Instance of product advertised.
INSTALLSTATE_UNKNOWN	Instance of product unknown.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IProduct is defined as 000C10A0-0000-0000-C000-000000000046

## See Also

## Product

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Product.UserSid Property

The **UserSid** property returns the user security identifier (SID) under which this product instance is available. This is a read-only property.

## Syntax

Script

```
Product.UserSid
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IProduct is defined as 000C10A0-0000-0000-C000-000000000046

## See Also

### [Product](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Record Object

The **Record** object is a container for holding and transferring a variable number of values. Fields within the record are numerically indexed and can contain strings, integers, objects, and null values. Fields beyond the allocated record size are treated as having permanently null values. Field number 0 is reserved.

## Methods

The **Record** object defines the following methods.

Method	Description
<b>ClearData</b>	Clears the data in all fields, setting them to null.
<b>FormatText</b>	Formats fields according to the template in field 0.
<b>ReadStream</b>	Reads a specified number of bytes from a record field holding stream data.
<b>SetStream</b>	Copies the content of the specified file into the designated record field as stream data.

## Properties

The **Record** object defines the following properties.

Property	Access type	Description
<b>DataSize</b>	Read-only	Returns the size of the data for the designated field.
<b>FieldCount</b>	Read-only	Returns the number of fields in the record.

<b>IntegerData</b>	Read-only	Transfers 32-bit integer data in to or out of a specified field within the record.
<b>IsNull</b>	Read-only	Returns True if the indicated field is null and False if the field contains data.
<b>StringData</b>	Read-only	Transfers string data in to or out of a specified field within the record.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IRecord is defined as 000C1093-0000-0000-C000-000000000046

## See Also

### [CreateRecord Method](#)

### [Windows Installer Scripting Examples](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Record.ClearData Method

The **ClearData** method of the **Record** object clears the data in all fields, setting them to Null. Any objects stored in the fields are released.

## Syntax

```
Script  
ClearData()
```

## Parameters

This method has no parameters.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IRecord is defined as 000C1093-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Record.DataSize Property

The **DataSize** property of the **Record** object is a read-only property that returns the size of the data for the designated field. If the data is a stream, the stream length in bytes is returned. If the data is a string, the string length without Null is returned. If the data is an integer, the value 4 is returned (indicating the size of the integer). If the data is Null, 0 is returned.

## Syntax

[Script](#)

```
Record.DataSize
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IRecord is defined as 000C1093-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Record.FieldCount Property

The **FieldCount** property of the **Record** object is a read-only property that returns the number of fields in the record. Read access to fields beyond this count returns Null values. Write access fails.

## Syntax

[Script](#)

```
Record.FieldCount
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IRecord is defined as 000C1093-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Record.FormatText Method

The **FormatText** method of the **Record** object formats fields according to the template in field 0.

## Syntax

```
Script  
FormatText()
```

## Parameters

This method has no parameters.

## Return Value

This method does not return a value.

## Remarks

The **FormatText** method follows the functionality of the **MsiFormatRecord** function if **MsiFormatRecord** was passed a null installer handle as its first parameter. As a result, only the record field parameters are processed and properties are not available for substitution.

For example, a string such as "format this field: [1], format this property: [property]" is resolved to "format this field: value from field 1, format this property: [property]."

Parameters that are to be **formatted** are enclosed in square brackets [...]. The square brackets can be iterated because the substitutions are resolved from the inside out.

If a part of the string is enclosed in curly braces { } and contains no square brackets, it is left unchanged, including the curly braces.

Note in the case of **deferred execution custom actions**, **FormatText** only supports a limited set of properties: the CustomActionData and

ProductCode properties. For more information, see [Obtaining Context Information for Deferred Execution Custom Actions](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IRecord is defined as 000C1093-0000-0000-C000-000000000046

## See Also

[MsiFormatRecord](#)  
[Formatted](#)  
[Column Data Types](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Record.IntegerData Property

This is the **IntegerData** property of the **Record** object. This read-write property transfers 32-bit integer data in to or out of a specified field within the record. If a field value cannot be converted to an integer, `msiDatabaseNullInteger` is returned.

## Syntax

Script

```
Record.IntegerData
```

## Remarks

To set a record integer field to null, use `msiDatabaseNullInteger`. The returned value of a nonexistent field is `msiDatabaseNullInteger`. Attempting to store a value in a nonexistent field causes an error.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IRecord is defined as 000C1093-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Record.IsNull Property

The **IsNull** property of the **Record** object is a read-only property that returns True if the indicated field is Null and False if the field contains data.

## Syntax

```
Script  
Record.IsNull
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IRecord is defined as 000C1093-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Record.ReadStream Method

The **ReadStream** method of the **Record** object reads a specified number of bytes from a record field that contains stream data.

## Syntax

```
Script  
ReadStream(  
    field,  
    length,  
    format  
)
```

## Parameters

*field*

The required field number of the value within the record, 1-based.

*length*

The required number of bytes to read from the stream.

*format*

Required interpretation and return of the data bytes.

Parameter name	Meaning
msiReadStreamInteger 0	As a long integer the length must be 1 to 4.
msiReadStreamBytes 1	The data as a <b>BSTR</b> —one byte per character.
msiReadStreamAnsi 2	The ANSI bytes translated to a Unicode <b>BSTR</b> .
msiReadStreamDirect 3	The byte pairs that are returned directly as a <b>BSTR</b> .

## Return Value

This method returns a string that contains the requested number of bytes read from a record field.

## Remarks

The returned value of a nonexistent field is an Empty value. If the stream has fewer bytes than the count requested, the returned string is shortened appropriately.

For an example of this method, see [Copy ANSI File Into a Database Field](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IRecord is defined as 000C1093-0000-0000-C000-000000000046

## See Also

### [Record](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Record.SetStream Method

The **SetStream** method of the [Record](#) object copies the content of the specified file into the designated record field as stream data. Stream data cannot be inserted into temporary fields.

## Syntax

```
Script
SetStream(
    field,
    filePath
)
```

## Parameters

*field*

Required field number of the value within the record, 1-based.

*filePath*

The location of the file to copy. No translation of any type is performed.

## Return Value

This method does not return a value.

## Remarks

If the method fails, you can obtain extended error information by using the [LastErrorRecord](#) method.

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows

<b>Version</b>	Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IRecord is defined as 000C1093-0000-0000-C000-000000000046

## See Also

### [Record](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Record.StringData Property

The **StringData** property of the **Record** object is a read-write property that transfers string data in to or out of a specified field within the record. If an integer or object has been stored, its string value is returned.

## Syntax

```
Script  
Record.StringData
```

## Remarks

The returned value of a nonexistent field is an empty string. To set a record string field to null, use either an empty variant or an empty string. Attempting to store a value in a nonexistent field causes an error.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IRecord is defined as 000C1093-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RecordList Object

The **RecordList** object is a collection of **Record** objects. You must verify that the **RecordList** object exists and is not empty before referencing its properties.

## Methods

The **RecordList** object does not define any methods.

## Properties

The **RecordList** object defines the following properties.

<b>Property</b>	<b>Access type</b>	<b>Description</b>
<b>Count</b>	Read-only	Returns the number of items in the <b>RecordList</b> object.
<b>Item</b>	Read-only	Returns a record in a <b>RecordList</b> object collection.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IRecordList is defined as 000C1096-0000-0000-C000-000000000046

## See Also

## **Record**

### Windows Installer Scripting Examples

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RecordList.Count Property

The **Count** property is a read-only property that returns the number of items in the **RecordList** object.

## Syntax

```
Script  
RecordList.Count
```

## Remarks

The client must verify that the **RecordList** object exists and is not empty before referencing the Count property.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IRecordList is defined as 000C1096-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RecordList.Item Property

The **Item** property is a read-only property that returns a record in a **RecordList** Object collection.

## Syntax

```
Script  
RecordList.Item
```

## Remarks

The client must verify that the **RecordList** object exists and is not empty before referencing the Item property.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IRecordList is defined as 000C1096-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session Object

The **Session** object controls the installation process. It opens the Installer database, which contains the installation tables and data. This object is associated with a standard set of action functions, each performing particular operations on data from one or more tables. Additional custom actions may be added for particular product installations. The basic engine function is a sequencer that fetches sequential records from a designated sequence table, evaluates any specified condition expression, and executes the designated action. Actions not recognized by the engine are deferred to the UI handler object for processing, usually dialog box sequences.

Note that only one **Session** object can be opened by a single process.

## Methods

The **Session** object defines the following methods.

Method	Description
<b>DoAction</b>	Executes the specified action.
<b>EvaluateCondition</b>	Evaluates a logical expression containing symbols and values and returns an integer of the enumeration <code>msiEvaluateConditionErrorEnum</code> .
<b>FeatureInfo</b>	Returns a <b>FeatureInfo</b> object containing descriptive information for the specified feature.
<b>FormatRecord</b>	Returns a formatted string from template and record data.
<b>Message</b>	Performs any enabled logging operations and defers execution to the UI handler object associated with the engine.
<b>Sequence</b>	Opens a query on the specified table, ordering

	the actions by the numbers in the Sequence column. For each row fetched, the <b>DoAction</b> method is called, provided that any supplied condition expression does not evaluate to False.
<b>SetInstallLevel</b>	Sets the install level for the current installation to a specified value and recalculates the Select and Installed states for all features.

## Properties

The **Session** object defines the following properties.

<b>Property</b>	<b>Access type</b>	<b>Description</b>
<b>ComponentCosts</b>	Read-only	Returns a <b>RecordList</b> object enumerating the disk space per drive required to install a component.
<b>ComponentCurrentState</b>	Read-only	Returns the current installed state of the designated component.
<b>ComponentRequestState</b>	Read-only	Obtains or requests a change in the Action state of a row in the Component table.
<b>Database</b>	Read-only	Returns the database for the current installation session.
<b>FeatureCost</b>	Read-only	Returns the total amount of disk space (in units of 512 bytes) required by the specified feature and its parent features (up to the root of the Feature table).
<b>FeatureCurrentState</b>	Read-only	Returns the current installed state of the designated feature.

<b>FeatureRequestState</b>	Read-only	Obtains or requests a change in the Select state of a feature's record and subrecords.
<b>FeatureValidStates</b>	Read-only	Returns an integer representing bit flags with each relevant bit representing a valid installation state for the specified feature.
<b>Installer</b>	Read-only	Returns the active installer object.
<b>Language (Session Object)</b>	Read-only	Represents the numeric language identifier used by the current installation session.
<b>Mode</b>	Read-only	This property is a value representing the designated mode flag for the current installation session.
<b>ProductProperty</b>	Read-only	Represents the string value of a named installer property.
<b>Property (Session Object)</b>	Read-only	Retrieves product properties from the product database.
<b>SourcePath</b>	Read-only	Provides the full path to the designated folder on the source media or server image.
<b>TargetPath</b>	Read-only	Provides the full path to the designated folder on the installation target drive.
<b>VerifyDiskSpace</b>	Read-only	Returns true if enough disk space exists, and false if the disk is full.

## Requirements

---

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISession is defined as 000C109E-0000-0000-C000-000000000046

## See Also

[Windows Installer Scripting Examples](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session.ComponentCosts Property

The ComponentCosts property of the [Session](#) object returns a [RecordList](#) object enumerating the disk space per drive required to install a component. This information is used by the user interface to display the disk space required for all drives. The returned disk space costs are in multiples of 512 bytes.

The ComponentCosts property should only be used after the installer has completed [file costing](#) and after the [CostFinalize action](#).

## Syntax

[Script](#)

```
Session.ComponentCosts
```

## Remarks

To obtain the total cost, add the costs for all components plus the installer engine cost (Component = "").

ComponentCosts returns a [RecordList object](#). Each record in the returned RecordList object has the following fields:

Field	Description
1	Volume/Drive name
2	Final disk space cost in multiples of 512 bytes.
3	Temporary disk space cost in multiples of 512 bytes.

## Requirements

[Version](#)

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000

<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISession is defined as 000C109E-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session.ComponentCurrentState Property

The **ComponentCurrentState** property of the [Session](#) object is a read-only property that returns the current installed state of the designated component. For state values, see the [ComponentRequestState](#) property.

## Syntax

*Script*

```
Session.ComponentCurrentState
```

## Remarks

If the property fails, you can obtain extended error information by using the [LastErrorRecord](#) method.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISession is defined as 000C109E-0000-0000-C000-000000000046

## See Also

[Session](#)  
[ComponentRequestState](#) property

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session.ComponentRequestState Property

The **ComponentRequestState** property of the [Session](#) object obtains or requests a change in the Action state of a row in the [Component table](#).

## Syntax

*Script*

```
Session.ComponentRequestState
```

## Remarks

Selection state	Value	Description
Null	Null	Requests that no action be taken for this item.
msiInstallStateAbsent	2	Item is to be removed.
msiInstallStateLocal	3	Item is to be installed locally.
msiInstallStateSource	4	Item is to be installed and run from the source media.
msiInstallStateDefault	5	If installed, the item is to be reinstalled in the same state.

If the property fails, you can obtain extended error information by using the [LastErrorRecord](#) method.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000

**DLL**

Msi.dll

**IID**

IID\_ISession is defined as 000C109E-0000-0000-C000-000000000046

## See Also

### [Session](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session.Database Property

The **Database** property of the **Session** object is a read-only property that returns the database for the current install session as a **Database** object.

## Syntax

```
Script  
Session.Database
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISession is defined as 000C109E-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session.DoAction Method

The **DoAction** method of the [Session](#) object executes the action function corresponding to the name supplied. If a Null action name is supplied, the engine uses the uppercase value of the ACTION property as the action to perform. If no property value is defined, the default action is performed, currently defined as INSTALL. This method returns an integer enumeration.

## Syntax

```
Script  
DoAction(  
    action  
)
```

## Parameters

*action*

Required string name of the action to execute. Case-sensitive.

## Return Value

This method does not return a value.

## Remarks

Actions that update the system, such as the [InstallFiles](#) and [WriteRegistryValues](#) actions, cannot be run by calling the **DoAction** method. The exception to this rule is if the **DoAction** method is called from a custom action that is scheduled in the [InstallExecuteSequence table](#) between the [InstallInitialize](#) and [InstallFinalize](#) actions. Actions that do not update the system, such as [AppSearch](#) or [CostInitialize](#), can be called.

## Requirements

---

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISession is defined as 000C109E-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session.EvaluateCondition Method

The **EvaluateCondition** method of the **Session** object evaluates a logical expression that contains symbols and values. This method uses the **MsiEvaluateCondition** function.

## Syntax

```
Script  
EvaluateCondition(  
    condition  
)
```

## Parameters

*condition*

Required string that contains the logical expression. For more information, see [Conditional Statement Syntax](#).

## Return Value

This method returns an integer that indicates the evaluation of the condition.

Constant	Value	Description
msiEvaluateConditionFalse	0	The condition evaluates to false.
msiEvaluateConditionTrue	1	The condition evaluates to true.
msiEvaluateConditionNone	2	A conditional expression is not provided.
msiEvaluateConditionError	3	The condition contains a syntax error.

## Remarks

Conditional expressions can be used to compare feature and component

states. The following table shows the feature and component states that the EvaluateCondition method uses.

State	Value	Description
Null	Null	No action taken on feature or component.
msiInstallStateAbsent	2	Feature or component is not present.
msiInstallStateLocal	3	Feature or component is installed on the local computer.
msiInstallStateSource	4	Feature or component is installed to run from source.

**Note** The states are not set until the [SetInstallLevel](#) method is called, either directly or by the [CostFinalize Action](#). Therefore, state checking is only useful in conditional expression in an action sequence table.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISession is defined as 000C109E-0000-0000-C000-000000000046

## See Also

[Conditional Statement Syntax](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session.FeatureCost Property

The **FeatureCost** property of the **Session** object returns the total amount of disk space (in units of 512 bytes) required by the specified feature and its parent features (up to the root of the Feature table). For each feature, the total cost is made up of the disk costs attributed to every component linked to the feature.

## Syntax

```
Script  
Session.FeatureCost
```

## Remarks

If the property fails, you can obtain extended error information by using the **LastErrorRecord** method.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISession is defined as 000C109E-0000-0000-C000-000000000046

## See Also

### [Session](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session.FeatureCurrentState Property

The **FeatureCurrentState** property of the [Session](#) object is a read-only property that returns the current installed state of the designated feature. For state values, see the [FeatureRequestState](#) property.

## Syntax

Script

```
Session.FeatureCurrentState
```

## Remarks

If the property fails, you can obtain extended error information by using the [LastErrorRecord](#) method.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISession is defined as 000C109E-0000-0000-C000-000000000046

## See Also

[Session](#)  
[FeatureRequestState](#) property

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session.FeatureInfo Method

The **FeatureInfo** method of the **Session** object returns a **FeatureInfo** object containing descriptive information for the specified feature.

## Syntax

```
Script  
FeatureInfo(  
    Feature  
)
```

## Parameters

*Feature*  
Identifies the feature of interest.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISession is defined as 000C109E-0000-0000-C000-000000000046

## See Also

[MsiGetFeatureInfo](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session.FeatureRequestState Property

The **FeatureRequestState** property is a read-write property of the **Session** object. It can be used to obtain the current action state of a feature or to request a change in the action of a feature and its subfeatures. The feature must be named in the **Feature** table. If a change to the action state of a feature is requested, the action state of all components of the changed feature may be changed as well. The action state of components shared by more than one feature will be changed as appropriate based on the new feature action state (see Remarks). The **FeatureRequestState** property can also be used to configure all features at once by specifying the keyword ALL instead of a specific feature name.

## Syntax

```
Script  
Session.FeatureRequestState
```

## Remarks

The **SetInstallLevel** method must be called before calling **FeatureRequestState**.

If the current state of the feature is being requested, the **FeatureRequestState** property returns one of the following values.

Selection state name	Meaning
msiInstallStateUnknown = -1	No action is taken for the feature. This is equivalent to null.
msiInstallStateAdvertised =1	Feature is to be advertised.
msiInstallStateAbsent = 2	Feature is to be removed.
msiInstallStateLocal = 3	Feature is to be installed as run-local.

msiInstallStateSource = 4	Feature is to be installed as run-from-source.
msiInstallStateDefault = 5	Feature is to be reinstalled with the feature's current action state.

For example, the following obtains the current state of "MyFeature" from a VBScript custom action.

```
Dim iRequestState
iRequestState = Session.FeatureRequestState("MyFeature")
```

If the state of the feature is being configured, the **FeatureRequestState** property should be set to one of the following values.

Selection state name	Meaning
msiInstallStateAdvertised = 1	Advertise the feature.
msiInstallStateAbsent = 2	Remove the feature.
msiInstallStateLocal = 3	Install the feature as run-local.
msiInstallStateSource = 4	Install the feature as run-from-source.

For example, the following requests from a VBScript custom action that "MyFeature" be installed to run locally on the computer.

```
Session.FeatureRequestState("MyFeature")=3
```

When **FeatureRequestState** is called, the installer attempts to set the action state of each component tied to the specified feature to the specified state, as best as possible. However, there are common situations when the request cannot be fully honored. For example, if a feature is tied to two components, Component A and Component B, through the [FeatureComponents](#) table and Component A has the **msidbComponentAttributesLocalOnly** attribute and component B has the **msidbComponentAttributesSourceOnly** attribute. In this case, if **FeatureRequestState** is called with a requested state of either

msiInstallStateLocal or msiInstallStateSource, the request cannot be honored for both components. In this case, both components can be turned on with Component A set to msiInstallStateLocal and Component B set to msiInstallStateSource.

If more than one feature is linked to a single component, the final action state of that component is determined as follows: If at least one feature calls for the component to be installed locally, it is installed msiInstallStateLocal. If at least one feature calls for the component to be run from the source media, it is installed msiInstallStateSource. If at least one feature calls for the removal of the component, the action state is msiInstallStateAbsent. For more information on how features are selected for installation, see the Feature table.

If the property fails, you can obtain extended error information by using the [LastErrorRecord](#) method.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISession is defined as 000C109E-0000-0000-C000-000000000046

## See Also

[Session Feature](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session.FeatureValidStates Property

The **FeatureValidStates** property of the **Session** object returns an integer representing bit flags with each relevant bit representing a valid installation state for the specified feature.

## Syntax

```
Script  
Session.FeatureValidStates
```

## Remarks

The return value is composed of bit flags as follows. Bit 0: if set, Local is a valid state. Bit 1: if set, Source is a valid state.

The **FeatureValidStates** property only succeeds after the installer has called the **CostInitialize** and **CostFinalize** actions.

**FeatureValidStates** determines state validity by querying all components that are linked to the specified feature without taking into account the current installed state of any component.

The possible valid states for a feature are determined as follows:

- If the feature does not contain components, both `INSTALLSTATE_LOCAL` and `INSTALLSTATE_SOURCE` are valid states for the feature.
- If at least one component of the feature has an attribute of `msidbComponentAttributesLocalOnly` or `msidbComponentAttributesOptional`, `INSTALLSTATE_LOCAL` is a valid state for the feature.
- If at least one component of the feature has an attribute of `msidbComponentAttributesSourceOnly` or `msidbComponentAttributesOptional`, `INSTALLSTATE_SOURCE` is a valid state for the feature.

- If a file of a component belonging to the feature is patched or from a compressed source, then `INSTALLSTATE_SOURCE` is not included as a valid state for the feature.
- `INSTALLSTATE_ADVERTISE` is not a valid state if the feature disallows advertisement (`msidbFeatureAttributesDisallowAdvertise`) or the feature requires platform support for advertisement (`msidbFeatureAttributesNoUnsupportedAdvertise`) and the platform does not support it.
- `INSTALLSTATE_ABSENT` is a valid state for the feature if its attributes do not include `msidbFeatureAttributesUIDisallowAbsent`.
- Valid states for child features marked to follow the parent feature (`msidbFeatureAttributesFollowParent`) are based upon the parent feature's action or installed state.

If the property fails, you can obtain extended error information by using the [LastErrorRecord](#) method.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISession is defined as 000C109E-0000-0000-C000-000000000046

## See Also

### [Session](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session.FormatRecord Method

The **FormatRecord** method of the **Session** object returns a formatted string from a template and record data.

## Syntax

```
Script  
FormatRecord(  
    record  
)
```

## Parameters

*record*

Required **Record** object containing a template and data to be formatted. The template string must be set in field 0 followed by any referenced data parameters.

## Return Value

This method does not return a value.

## Remarks

The **FormatRecord** method uses the following format process.

Parameters to be **formatted** are enclosed in square brackets [..]. The square brackets can be iterated because the substitutions are resolved from inside out.

If a part of the string is enclosed in curly braces { } and contains no square brackets, the part is left unchanged, including the curly braces.

If a part of the string is enclosed in curly braces and contains one or more property names, and if all the properties are found, the text (with the resolved substitutions) is displayed without the curly braces. If any of the properties are not found, all the text in the curly braces and the braces themselves are removed.

## ▶ To format strings using the FormatRecord method

1. The numeric parameters are substituted by replacing the marker with the value of the corresponding record field, with missing or Null values producing no text.
2. The string that results is processed by replacing the non-record parameters with the corresponding values, as noted in the following descriptions.
  - If a substring of the form "[propertyname]" is encountered, it is replaced by the value of the property.
  - If a substring of the form "[%environmentvariable]" is found, the value of the environment variable is substituted.
  - If a substring of the form [#filekey] is found, it is replaced by the full path of the file, with the value *filekey* used as a key into the [File table](#). The value of [#filekey] remains blank and is not replaced by a path until the installer runs the [CostInitialize action](#), [FileCost action](#), and [CostFinalize action](#). The value of [#filekey] depends upon the installation state of the component to which the file belongs. If the component is being run from source, the value is the path to the source location of the file. If the component is being run locally, the value is the path to the target location of the file after installation. If the component is absent, the path is blank. For more information about checking the installation state of components, see [Checking the Installation of Features, Components, Files](#).
  - If a substring of the form [\$componentkey] is found, it is replaced by the install directory of the component, with the value *componentkey* used as a key into the [Component table](#). The value of [\$componentkey] remains blank and is not replaced by a directory until the installer runs the

[CostInitialize action](#), [FileCost action](#), and [CostFinalize action](#). The value of [*\$componentkey*] depends upon the installation state of the component. If the component is being run from source, the value is the source directory of the file. If the component is being run locally, the value is the target directory after installation. If the component is absent, the value is left blank. For more information about checking the installation state of components, see [Checking the Installation of Features, Components, Files](#).

- If a substring of the form "[c]" is found, it is replaced by the character without any further processing. Only the first character after the backslash is kept; everything else is removed.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISession is defined as 000C109E-0000-0000-C000-000000000046

## See Also

[Formatted Column Data Types](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session.Installer Property

The read-only **Installer** property of the **Session** object returns the active **Installer** object.

## Syntax

```
Script  
Session.Installer
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISession is defined as 000C109E-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session.Language Property

The **Language** property of the **Session** object is a read-only property that represents the numeric language ID used by the current install session.

## Syntax

```
Script  
Session.Language
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISession is defined as 000C109E-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session.Message Method

The **Message** method of the **Session** object performs any enabled logging operations and defers execution to the UI handler object associated with the engine. Logging may be selectively enabled for the various message types. See the **EnableLog** method.

If record field 0 contains a formatting string, it is used to format the data in the other fields. Else if the message is an error, warning, or user message, an attempt is made to find a message template in the Error table for the current database using the error number found in field 1 of the record for message types and return values.

## Syntax

```
Script
Message(
    kind,
    record
)
```

## Parameters

### *kind*

The *kind* parameter is required to be one of the following values. To display a message box with push buttons and icons, calculate the value of *kind* by adding the standard message box styles used by MessageBox and MessageBoxEx to `msiMessageTypeError`, `msiMessageTypeWarning`, or `msiMessageTypeUser`. For more information see the Remarks section below.

Constant	Meaning
<code>msiMessageTypeFatalExit</code> &H00000000	Premature termination, possibly fatal out of memory.
<code>msiMessageTypeError</code> &H01000000	Formatted error message, [1] is message number in <a href="#">Error table</a> .

msiMessageTypeWarning &H02000000	Formatted warning message, [1] is message number in <a href="#">Error table</a> .
msiMessageTypeUser &H03000000	User request message, [1] is message number in <a href="#">Error table</a> .
msiMessageTypeInfo &H04000000	Informative message for log, not to be displayed.
msiMessageTypeFilesInUse &H05000000	List of files in use that need to be replaced.
msiMessageTypeResolveSource &H06000000	Request to determine a valid source location.
msiMessageTypeOutOfDiskSpace &H07000000	Insufficient disk space message.
msiMessageTypeActionStart &H08000000	Start of action, [1] action name, [2] description, [3] template for ACTIONDATA messages.
msiMessageTypeActionData &H09000000	Action data. Record fields correspond to the template of ACTIONSTART message.
msiMessageTypeProgress &H0A000000	Progress bar information. See the description of record fields below.
msiMessageTypeCommonData &H0B000000	To enable the Cancel button set [1] to 2 and [2] to 1.  To disable the Cancel button set [1] to 2 and [2] to 0

*record*

Required **Record** object containing a message-specific field.

## Return Value

Constant	Value
msiMessageStatusError	-1
msiMessageStatusNone	0
msiMessageStatusOk	1
msiMessageStatusCancel	2
msiMessageStatusAbort	3
msiMessageStatusRetry	4
msiMessageStatusIgnore	5
msiMessageStatusYes	6
msiMessageStatusNo	7

## Remarks

### Message Record Fields

The following describes the record field definitions when `msiMessageTypeProgress` is passed as the message type.

Field 1 specifies the type of the progress message. The meaning of the other fields depend upon the value in this field. The possible values that can be set into Field 1 are as follows.

Message name	Value	Field 1 description
MasterReset	0	Resets progress bar and sets the expected total number of ticks in the bar.
ActionInfo	1	Provides information related to progress messages to be sent by the current action.

ProgressReport	2	Increments the progress bar.
ProgressAddition	3	Enables an action (such as CustomAction) to add ticks to the expected total number of progress of the progress bar.

The meaning of Field 2 depends upon the value in Field 1 as follows.

<b>Field 1</b>		
<b>value</b>	<b>Field 2 description</b>	
0	Expected total number of ticks in the progress bar.	
1	Number of ticks the progress bar moves for each ActionData message. This field is ignored if Field 3 is 0.	
2	Number of ticks the progress bar has moved.	
3	Number of ticks to add to total expected progress.	

The meaning of Field 3 depends upon the value in Field 1 as follows.

<b>Field 1</b>		
<b>value</b>	<b>Field 3 value</b>	<b>Field 3 description</b>
0	0	Forward progress bar (left to right)
	1	Backward progress bar (right to left)
1	0	The current action will send explicit ProgressReport messages.
	1	Increment the progress bar by the number of ticks specified in Field 2 each time an ActionData message is sent.
2	Unused	
3	Unused	

The meaning of Field 4 depends upon the value in Field 1 as follows.



Field 1 value	Field 4 value	Field 4 description
0	0	Execution is in progress. In this case, the UI could calculate and display the time remaining.
	1	Creating the execution script. In this case, the UI could display a message to please wait while the installer finishes preparing the installation.
1	Unused	
2	Unused	
3	Unused	

## Displaying Message Boxes

To display a message box with push buttons and icons, calculate the value of *kind* by adding the standard message box styles used by [MessageBox](#) and [MessageBoxEx](#) to `msiMessageTypeError`, `msiMessageTypeWarning`, or `msiTypeUser`. The available push button options for VBScript are `vbOKOnly` (`MB_OK`), `vbOKCancel` (`MB_OKCANCEL`), `vbAbortRetryIgnore` (`MB_ABORTRETRYIGNORE`), `vbYesNoCancel` (`MB_YESNOCANCEL`), `vbYesNo` (`MB_YESNO`), and `vbRetryCancel` (`MB_RETRYCANCEL`). The available icon options for VBScript are `vbCritical` (`MB_ICONERROR`), `vbQuestion` (`MB_ICONQUESTION`), `vbExclamation` (`MB_ICONWARNING`), and `vbInformation` (`MB_ICONINFORMATION`).

For example, the following call sends a `msiMessageTypeError` message with the `vbExclamation` icon and `vbYesNo` buttons.

```
Session.Message &H01000034, record
```

If a custom action calls the **Message** method, the custom action should be capable of handling a cancellation by the user and should return `msiDoActionStatusUserExit`.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISession is defined as 000C109E-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session.Mode Property

This is the **Mode** property of the [Session](#) object. This property is a value representing the designated mode flag for the current install session. Most of the mode flags are read-only externally, but a few specified flags may be set as well.

The [MsiGetMode](#) function returns a Boolean TRUE or FALSE, indicating whether the specific property passed into the function is currently set (TRUE) or not set (FALSE).

Note that not all the run mode values of *flag* are available when calling the **Mode** property from a deferred custom action. For more information, see [Obtaining Context Information for Deferred Execution Custom Actions](#).

## Syntax

```
Script  
Session.Mode
```

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session.ProductProperty Property

The **ProductProperty** Property of the **Session** object is a read-only property that retrieves product properties from the product database.

## Syntax

Script

```
Session.ProductProperty
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISession is defined as 000C109E-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session.Property Property

The **Property** property of the **Session** object is a read-write property that represents the string value of a named installer property, as maintained by the **Session** object in the in-memory Property table, or, if it is prefixed with a percent sign (%), the value of a system environment variable for the current process. Either string or integer values may be supplied. A non-existent property or environment variable is equivalent to its value being Null.

## Syntax

Script

```
Session.Property
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISession is defined as 000C109E-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session.Sequence Method

The **Sequence** method of the **Session** object opens a query on the specified table, ordering the actions by the numbers in the Sequence column. For each row fetched, the **DoAction** method is called, provided that any supplied condition expression does not evaluate to False. Returns an enumeration `msiDoActionStatusEnum`, as described in the **DoAction** method.

## Syntax

```
Script
Sequence(
    table
)
```

## Parameters

*table*

Required string name of the table to use for sequencing.

## Return Value

This method does not return a value.

## Remarks

This method is normally called internally by top-level actions.

An action sequence containing actions that update the system, such as the **InstallFiles** and **WriteRegistryValues** actions, cannot be run by calling the **Sequence** method. The exception to this rule is if the **Sequence** method is called from a custom action that is scheduled in the **InstallExecuteSequence table** between the **InstallInitialize** and **InstallFinalize** actions. Actions that do not update the system, such as **AppSearch** or **CostInitialize**, can be called.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISession is defined as 000C109E-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session.SetInstallLevel Method

The **SetInstallLevel** method of the **Session** object sets the install level for the current installation to a specified value and recalculates the Select and Installed states for all features in the Feature table. It also sets the Action state of each component in the **Component table** based on the new level.

## Syntax

```
Script
SetInstallLevel(
    installLevel
)
```

## Parameters

*installLevel*

Required requested new install level.

## Return Value

This method does not return a value.

## Remarks

The **CostInitialize action** must be executed prior to calling **SetInstallLevel**.

If 0 is passed for the *installLevel* parameter, the current install level is not changed, but all features are still updated based on the current install level. For example, this functionality could be used by the Handler module to reset all selections back to their initial default states at any point in the UI selection process.

If the method fails, you can obtain extended error information by using the **LastErrorRecord** method.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISession is defined as 000C109E-0000-0000-C000-000000000046

## See Also

### [Session](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session.SourcePath Property

The **SourcePath** property of the **Session** object is a read-only property that provides the full path to the designated folder on the source media or server image.

## Syntax

Script

```
Session.SourcePath
```

## Remarks

If the property fails, you can obtain extended error information by using the **LastErrorRecord** method.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISession is defined as 000C109E-0000-0000-C000-000000000046

## See Also

### [Session](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session.TargetPath Property

The **TargetPath** property of the **Session** object is a read-write property that provides the full path to the designated folder on the installation target drive.

## Syntax

```
Script  
Session.TargetPath
```

## Remarks

Do not attempt to configure the target path if the components using those paths are already installed for the current user or for a different user. Check the **ProductState** property to determine if the product that contains the component is installed.

If the property fails, you can obtain extended error information by using the **LastErrorRecord** method.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISession is defined as 000C109E-0000-0000-C000-000000000046

## See Also

[Session](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Session.VerifyDiskSpace Property

The **VerifyDiskSpace** property is a read-only property. It returns true if enough disk space exists and false if the disk is full. Because this property uses information gathered by the costing actions, **VerifyDiskSpace** should only be called after the [CostInitialize action](#), [FileCost action](#), and [CostFinalize action](#).

## Syntax

Script

```
Session.VerifyDiskSpace
```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISession is defined as 000C109E-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# StringList Object

The **StringList** object is a collection of strings. The client must verify that the **StringList** object exists and is not empty before referencing its properties.

## Methods

The **StringList** object does not define any methods.

## Properties

The **StringList** object defines the following properties.

Property	Access type	Description
<b>Count</b>	Read-only	Returns the number of items in the <b>StringList</b> object.
<b>Item</b>	Read-only	Returns a string in the <b>StringList</b> object collection.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IStringList is defined as 000C1095-0000-0000-C000-000000000046

## See Also

## Windows Installer Scripting Examples

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# StringList.Count Property

The **Count** property is a read-only property that returns the number of items in the **StringList** object.

## Syntax

```
Script  
StringList.Count
```

## Remarks

The client must verify that the **StringList** object exists and is not empty before referencing the **Count** property.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IStringList is defined as 000C1095-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# StringList.Item Property

The **Item** property is a read-only property that returns a string in the **StringList Object** collection.

## Syntax

```
Script  
StringList.Item
```

## Remarks

The client must verify that the **StringList Object** exists and is not empty before referencing the **Item** property.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IStringList is defined as 000C1095-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SummaryInfo Object

The **SummaryInfo** object is used to read, create, and update document properties from the [summary information stream](#) of a storage object.

## Methods

The **SummaryInfo** object defines the following method.

Method	Description
<b>Persist</b>	Formats and writes the previously stored properties into the standard summary information stream.

## Properties

The **SummaryInfo** object defines the following properties.

Property	Access type	Description
<b>PropertyProperty (SummaryInfo Object)</b>	Read-only	Gets or sets the value for the specified property in the summary information stream.
<b>PropertyCount</b>	Read-only	Indicates the current number of property values in the summary information object.

## Requirements

	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or
--	--

<b>Version</b>	Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISummaryInfo is defined as 000C109B-0000-0000-C000-000000000046

## See Also

[SummaryInformation Property \(Database Object\)](#)

[SummaryInformation Property \(Installer Object\)](#)

[Windows Installer Scripting Examples](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SummaryInfo.Persist Method

The **Persist** method of the **SummaryInfo** object formats and writes the previously stored properties into the standard SummaryInformation stream. It generates an error if the stream cannot be successfully written. This method may only be called once after all the property values have been set. Properties may still be read after the stream is written.

## Syntax

```
Script  
Persist()
```

## Parameters

This method has no parameters.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISummaryInfo is defined as 000C109B-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SummaryInfo.Property Property

The **Property** property of the **SummaryInfo** object sets or gets the value for the specified property in the summary information stream. The properties are read when the **SummaryInfo** object is created, but they are not written until the **Persist** method is called. Setting a property to Empty causes its removal; likewise, requesting a nonexistent property returns an Empty value. Otherwise values may be transferred as strings, integers, or date (date time) types.

## Syntax

Script

```
SummaryInfo.Property
```

## Remarks

### Standard Summary Property IDs

(not an enumeration)

Parameter name	Value	Description
PID_DICTIONARY	0	Special format, not support by SummaryInfo object
PID_CODEPAGE	1	VT_I2
PID_TITLE	2	VT_LPSTR
PID_SUBJECT	3	VT_LPSTR
PID_AUTHOR	4	VT_LPSTR
PID_KEYWORDS	5	VT_LPSTR
PID_COMMENTS	6	VT_LPSTR
PID_TEMPLATE	7	VT_LPSTR
PID_LASTAUTHOR	8	VT_LPSTR
PID_REVNUMBER	9	VT_LPSTR

PID_EDITTIME	10	VT_FILETIME
PID_LASTPRINTED	11	VT_FILETIME
PID_CREATE_DTM	12	VT_FILETIME
PID_LASTSAVE_DTM	13	VT_FILETIME
PID_PAGECOUNT	14	VT_I4
PID_WORDCOUNT	15	VT_I4
PID_CHARCOUNT	16	VT_I4
PID_THUMBNAIL	17	VT_CF (not supported)
PID_APPNAME	18	VT_LPSTR
PID_SECURITY	19	VT_I4

## Property Data Types

(not an enumeration)

Parameter name	Value	Description
VT_I2	2	16-bit integer
VT_I4	3	32-bit integer
VT_LPSTR	30	String
VT_FILETIME	64	Date time (FILETIME, converted to Variant time)
VT_CF	71	Clipboard format + data, not handled by <a href="#">SummaryInfo</a> object

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows

<b>Version</b>	Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISummaryInfo is defined as 000C109B-0000-0000-C000-000000000046

## See Also

[MsiSummaryInfo SetProperty](#)

[MsiSummaryInfo GetProperty](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SummaryInfo.PropertyCount

## Property

The **PropertyCount** property of the [SummaryInfo](#) object is a read-only property that indicates the current number of property values in the summary information object. It takes into account properties that have been added, deleted, or replaced.

### Syntax

```
Script  
SummaryInfo.PropertyCount
```

### Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_ISummaryInfo is defined as 000C109B-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UIPreview Object

The **UIPreview** object is used to view user interface dialog boxes and billboards during authoring. This object is created by the **EnableUIPreview** method of the **Database** object.

## Methods

The **UIPreview** object defines the following methods.

Method	Description
<b>ViewBillboard</b>	Displays an authored billboard using the host control in the currently displayed dialog box.
<b>ViewDialog</b>	Displays an authored UI dialog box stored in the current database.

## Properties

The **UIPreview** object defines the following property.

Property	Access type	Description
<b>Property</b>	Read-only	Represents the string value of a named installer property or, if prefixed with a percent sign (%), the value of a system environment variable for the current process.

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or

<b>Version</b>	Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IUIPreview is defined as 000C109A-0000-0000-C000-000000000046

## See Also

[Windows Installer Scripting Examples](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UIPreview.Property Property

The **Property** property of the **UIPreview** object is a read-write property that represents the string value of a named installer property, or, if it is prefixed with a percent sign (%), the value of a system environment variable for the current process. This is exposed to allow proper display of dialog boxes that are dependent upon property values. Either string or integer values may be supplied. A nonexistent property or environment variable is equivalent to its value being null.

## Syntax

Script

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_UIPreview is defined as 000C109A-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UIPreview.ViewBillboard Method

The **ViewBillboard** method of the **UIPreview** object displays an authored billboard using the host control in the currently displayed dialog box.

## Syntax

```
Script  
ViewBillboard(  
    control,  
    billboard  
)
```

## Parameters

### *control*

Required name of the control hosting the billboard, case-sensitive, along with the dialog box, and the primary keys of the Control database table.

### *billboard*

Required name of the billboard to display using the specified control and current dialog box, and the primary key of the Billboard database table.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll

**IID**

IID\_IUIPreview is defined as 000C109A-0000-0000-  
C000-0000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UIPreview.ViewDialog Method

The **ViewDialog** method of the **UIPreview** object displays an authored user interface dialog box stored in the current database.

## Syntax

```
Script  
ViewDialog(  
    dialog  
)
```

## Parameters

*dialog*

Required name of the dialog box, case-sensitive, and the primary key of the Dialog database table.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_UIPreview is defined as 000C109A-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# View Object

The **View** object represents a result set obtained when processing a query using the **OpenView** method of the **Database** object. Before any data can be transferred, the query must be executed using the **Execute** method, passing to it all replaceable parameters designated within the SQL query string. The query may be executed again, with different parameters if needed, but only after freeing the result set either by fetching all the records or by calling the **Close** method.

## Methods

The **View** object defines the following methods.

Method	Description
<b>Close</b>	Terminates query execution and releases database resources.
<b>Execute</b>	Uses the question mark token to represent parameters in a SQL query. The values of these parameters are passed in as the corresponding fields of a parameter record.
<b>Fetch</b>	Returns a <b>Record</b> object containing the requested column data if more rows are available in the result set, otherwise, it returns null.
<b>GetError</b>	Returns the Validation error and column name for which the error occurred.
<b>Modify</b>	Modifies a database row with a modified <b>Record</b> object obtained by the <b>Fetch</b> method.

## Properties

The **View** object defines the following property.

<b>Property</b>	<b>Access type</b>	<b>Description</b>
<b>ColumnInfo</b>	Read-only	Returns a <b>Record</b> object containing the requested information about each column in the result set.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IView is defined as 000C109C-0000-0000-C000-000000000046

## See Also

[Windows Installer Scripting Examples](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# View.Close Method

The **Close** method of the **View** object terminates query execution and releases database resources.

## Syntax

```
Script  
Close()
```

## Parameters

This method has no parameters.

## Return Value

This method does not return a value.

## Remarks

This method must be called before the **Execute** method is called again on the **View** object unless all rows of the result set have been obtained with the **Fetch** method. It is called internally when the view is destroyed.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IView is defined as 000C109C-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# View.ColumnInfo Property

The **ColumnInfo** property of the **View** object returns a **Record** object containing the requested information about each column in the result set. Either the column names or the column definitions may be requested. Constants supplied in the Select list do not have names.

## Syntax

Script

```
View.ColumnInfo
```

## Remarks

The column descriptions returned by the **ColumnInfo** property are in the format described in **Column Definition Format**. Each column is described by a string in the corresponding record field. The definition string consists of a single letter representing the data type followed by the width of the column (in characters when applicable, or else bytes). A width of zero designates an unbounded width (long text fields and streams). An uppercase letter indicates that Null values are allowed in the column.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IView is defined as 000C109C-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# View.Execute Method

The **Execute** method of the **View** object uses the question mark token to represent parameters in an SQL statement. For more information, see [SQL Syntax](#). The values of these parameters are passed in as the corresponding fields of a parameter record.

## Syntax

```
Script  
Execute(  
    record  
)
```

## Parameters

*record*

Optional **Record** objects that contains the values that replace the parameter tokens (?) in the SQL query.

## Return Value

This method does not return a value.

## Remarks

This method must be called before any calls to the **Fetch** method.

If the SQL query specifies values with parameter markers (?), a record must be supplied that contains all of the replacement values, which must be in the same order and of the same data type as the parameter markers. When this method is used with INSERT and UPDATE queries, the question mark tokens must precede all the non-parameterized values.

For example, these queries are valid:

```
UPDATE {table-list} SET {column}= ? , {column}= {constant}
```

```
INSERT INTO {table} ({column-list}) VALUES (?, {constant-list})
```

However these queries are invalid:

```
UPDATE {table-list} SET {column}= {constant}, {column}=?
```

```
INSERT INTO {table} ({column-list}) VALUES ({constant-list}, ? )
```

If the method fails, you can obtain extended error information by using the [LastErrorRecord](#) method.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IView is defined as 000C109C-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# View.Fetch Method

The **Fetch** method of the **View** object retrieves the next row of column data if more rows are available in the result set, otherwise it is Null. Call the **Execute** method before the **Fetch** method.

## Syntax

```
Script  
Fetch()
```

## Parameters

This method has no parameters.

## Return Value

This method does not return a value.

## Remarks

The same **Record** object should be used to retrieve the data in multiple rows, or else the object should be released by going out of scope. The record can be tested for the end of the result set using the syntax "If FetchRecord Is Nothing".

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IView is defined as 000C109C-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# View.GetError Method

The **GetError** method of the **View** object returns the Validation error and the column name for which the error occurred. In automation, the returned value is a string of the form `##ColumnName`, where the `##` represents a 2-digit error code. It returns the first error in the view's error array; subsequent calls return the next value in the error array. Once a value of '00' is returned, no more errors exist, and subsequent calls merely loop through the array again.

## Syntax

```
Script  
GetError()
```

## Parameters

This method has no parameters.

## Return Value

This method does not return a value.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IView is defined as 000C109C-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# View.Modify Method

The **Modify** method of the **View** object modifies a database row with a modified **Record** object obtained by the **Fetch** method.

## Syntax

```
Script
Modify(
    action,
    record
)
```

## Parameters

### *action*

Required action to be performed on the database row. This action is one of those shown in the following table.

Action name	Meaning
msiViewModifySeek -1	Refreshes the information in the supplied record without changing the position in the result set and without affecting subsequent fetch operations. The record may then be used for subsequent Update, Delete, and Refresh. All primary key columns of the table must be in the query and the record must have at least as many fields as the query. Seek cannot be used with multitable queries. See the remarks. This mode cannot be used with a view containing joins.
msiViewModifyRefresh 0	Refreshes the information in the record. Must first call the <b>Fetch</b> method with the same record. Fails

	for a deleted row. Works with both read-write and read-only records.
msiViewModifyInsert 1	Inserts a record. Fails if a row with the same primary keys exists. Fails with a read-only database. This mode cannot be used with a view containing joins.
msiViewModifyUpdate 2	Updates an existing record. Non-primary keys only. Must first call the <b>Fetch</b> method with the same record. Fails with a deleted record. Works only with read-write records.
msiViewModifyAssign 3	Writes current data in the cursor to a table row. Updates record if the primary keys match an existing row and inserts if they do not match. Fails with a read-only database. This mode cannot be used with a view containing joins.
msiViewModifyReplace 4	Updates or deletes and inserts a record into a table. Must first call the <b>Fetch</b> method with the same record. Updates record if the primary keys are unchanged. Deletes old row and inserts new if primary keys have changed. Fails with a read-only database. This mode cannot be used with a view containing joins.
msiViewModifyMerge 5	Inserts or validates a record in a table. Inserts if primary keys do not match any row and validates if there is a match. Fails if the record does not match the data in the table. Fails if there is a record with a duplicate

	key that is not identical. Works only with read-write records. This mode cannot be used with a view containing joins.
msiViewModifyDelete 6	Removes a row from the table. Must first call the <b>Fetch</b> method with the same record. Fails if the row has been deleted. Works only with read-write records. This mode cannot be used with a view containing joins.
msiViewModifyInsertTemporary 7	Inserts a temporary record. The information is not persistent. Fails if a row with the same primary key exists. Works only with read-write records. This mode cannot be used with a view containing joins.
msiViewModifyValidate 8	Validates a record. Does not validate across joins. Must first call the <b>Fetch</b> method with the same record. Obtain validation errors with <b>GetError</b> method. Works with read-write and read-only records. This mode cannot be used with a view containing joins.
msiViewModifyValidateNew 9	Validates a new record. Does not validate across joins. Checks for duplicate keys. Obtains validation errors by calling <b>GetError</b> method. Requires calling <b>MsiDatabase.OpenView</b> method with a modify value. Works with read-write and read-only records. This mode cannot be used with a view containing joins.
msiViewModifyValidateField	Validates fields of a fetched or new

10	record. Can validate one or more fields of an incomplete record. Obtains validation errors by calling <b>GetError</b> method. Works with read-write and read-only records. This mode cannot be used with a view containing joins.
msiViewModifyValidateDelete 11	Validates a record that will be deleted later. Must first call the <b>Fetch</b> method with the same record. Fails if another row refers to the primary keys of this row. Validation does not check for the existence of the primary keys of this row in properties or strings. Does not check if a column is a foreign key to multiple tables. Obtain validation errors by calling the <b>GetError</b> method. Works with read-write and read-only records. This mode cannot be used with a view containing joins.

*record*

Required. **Record** object obtained by the **Fetch** method with modified field data.

## Return Value

This method does not return a value.

## Remarks

This method must be called after the **Execute** method.

To execute any SQL statement, a view must be created. However, a view that does not create a result set, such as CREATE TABLE or INSERT INTO, cannot be used with the **Modify** method to update tables though

the view.

The `msiViewModifyValidate`, `msiViewModifyValidateNew`, `msiViewModifyValidateField`, and `msiViewModifyValidateDelete` values of the **Modify** method do not perform actual updates; they ensure that the data in the record is valid. Use of these actions requires that the database contain a [\\_Validation table](#).

You cannot fetch a record containing binary data from one database and then use that record to insert the data into a completely different database. To move binary data from one database to another, you should export the data to a file and then import it into the new database using the **SetStream** method of the **Record** object. This ensures that each database has its own copy of the binary data.

**Note** Custom actions can only add, modify, or remove temporary rows, columns, or tables from a database. Custom actions cannot modify persistent data in a database, such as data that is a part of the database stored on disk. For more information, see [Accessing the Current Installer Session from Inside a Custom Action](#).

If the method fails, you can obtain extended error information by using the **LastErrorRecord** method.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>DLL</b>	Msi.dll
<b>IID</b>	IID_IView is defined as 000C109C-0000-0000-C000-000000000046

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer Functions

The following sections describe function calls to Windows Installer API.

[About Installer Functions](#)

[Using Installer Functions](#)

[Installer Function Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# About Installer Functions

Windows Installer allows system administrators to install, update, and remove components. For more information about adding installer functionality, see [Using Installer Functions](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Installer Functions

To add Windows Installer functionality to your application, use the installer service functions. Because the installer is a setup utility and component management system, an application that uses the installer must incorporate the installer service functions as part of normal operation.

The following topics and procedures explain how to implement installer functionality in your application:

- [Initializing an Application](#)
- [Getting Application Information](#)
- [Requesting a Feature](#)
- [Installing an Application](#)
- [Reinstalling a Feature or Application](#)
- [Removing an Advertised Application](#)
- [Working with Features and Components](#)
- [Installing a Missing Component](#)
- [Inventory products and patches](#)
- [Managing Installation Sources](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Initializing an Application

To enable installer functionality, an application must call a number of functions when it is initializing. For more information, see [Installation Mechanism](#). The following steps describe how to use the installer to initialize an application:

## ► To initialize an application

1. Call the [MsiGetProductCode](#) function so the application can identify itself to the installer.  
The product code is a required parameter for many installer functions.
2. Call the [MsiGetUserInfo](#) function to collect user information the first time the application starts.  
If the call to [MsiGetUserInfo](#) fails, call the [MsiCollectUserInfo](#) function to collect user information.
3. Display a default user interface, if necessary, by calling the [MsiSetInternalUI](#) function.  
To author your own user interface, register it with the installer by calling the [MsiSetExternalUI](#) function.
4. Call the [MsiEnableLog](#) function to set the logging level.
5. Present the user with available features by enumerating the features of your application. You can enumerate features in the following ways:
  - Query the installer feature-by-feature. For example, before the application draws a button or a menu item, the application calls the [MsiQueryFeatureState](#) function so the installer can check that the feature is available.
  - Enumerate all of the available features at once by calling the [MsiEnumFeatures](#) function. To use this function, the

application must call **MsiEnumFeatures** repeatedly while incrementing an index.

6. Get detailed information about the current installation by calling the following enumeration functions repeatedly, incrementing an index variable for each call:
  - Call the **MsiEnumProducts** function to enumerate products registered with the installer.
  - Call the **MsiEnumComponents** function to enumerate components.
  - Call the **MsiEnumComponentQualifiers** function to enumerate component qualifiers.
  - Call the **MsiEnumClients** function to enumerate the products for a particular component.

If the return value on an enumeration function is `ERROR_SUCCESS`, there are still more items to be enumerated and the function should be called again with an incremented index variable. If the return value is `ERROR_NO_MORE_ITEMS`, then all of the items have been enumerated, and the function should not be called again.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Getting Application Information

The product database contains information about a product. For more information on obtaining product information with enumeration functions, see [Initializing an Application](#).

## ► To get product information

1. Verify that a product is installed by calling the **MsiQueryProductState** function.
2. Open the database and obtain a handle to it by calling the **MsiOpenProduct** function.  
If the database is contained in an installation package, call the **MsiOpenPackage** function.
3. Use the open handle to obtain product properties with the **MsiGetProductProperty** function, and to obtain descriptive feature information with the **MsiGetFeatureInfo** function.  
If you want to obtain product information using the product code, rather than using the open database handle, call the **MsiGetProductInfo** function instead of **MsiGetProductProperty**.
4. Close an open installation handle by calling the **MsiCloseHandle** function.  
The **MsiCloseAllHandles** function is a diagnostic function and should not be used to close handles you know to be open. It is acceptable to call the **MsiCloseAllHandles** function when the application closes to ensure that all handles have been closed.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

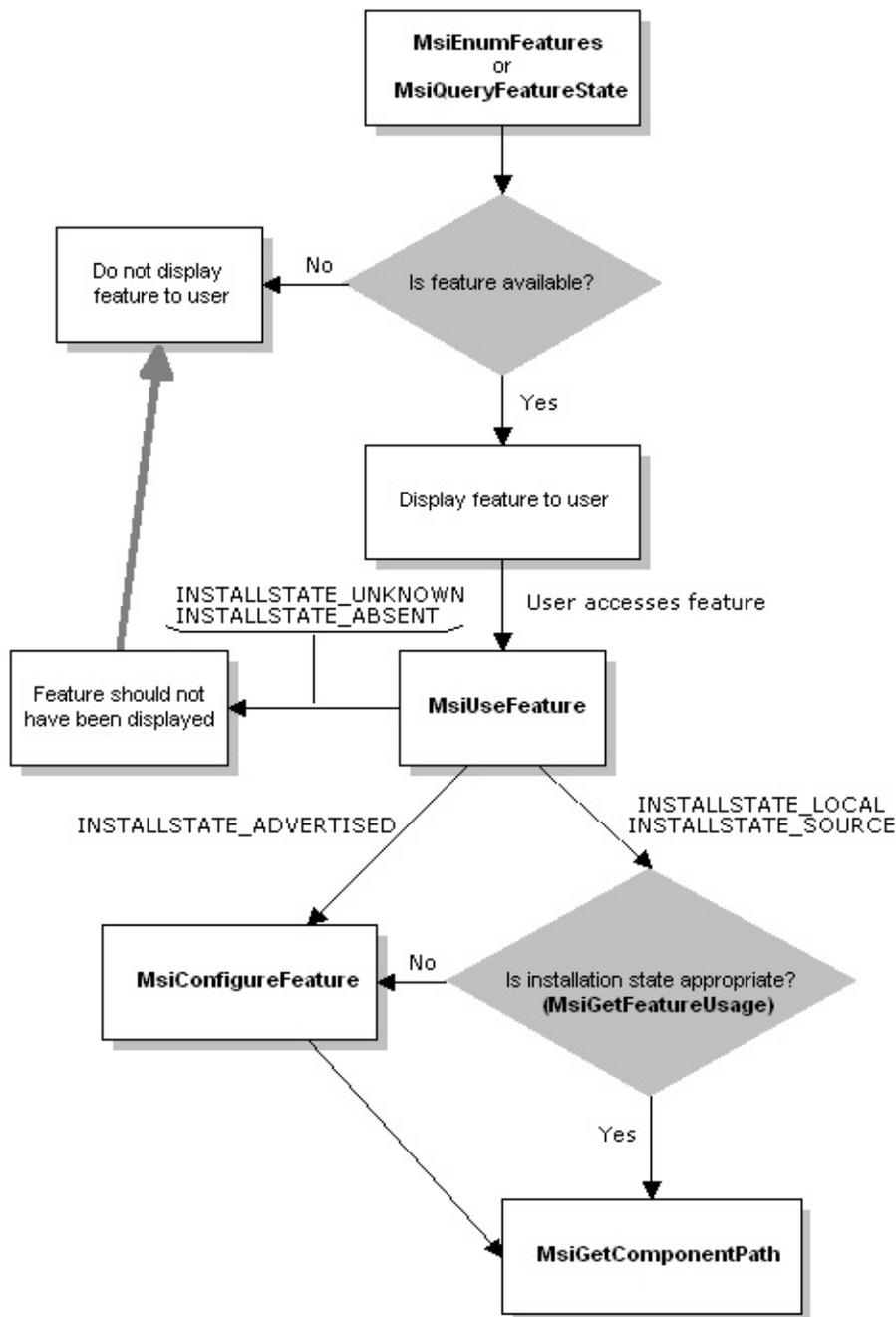
# Requesting a Feature

There are several functions an application must call to request features. Before requesting a feature, the application must ensure that the feature is installed. If the application calls **MsiUseFeature** before the application accesses a feature, the application can use the information returned to maintain usage metrics.

## ► To request a feature

1. Call the **MsiEnumFeatures** or the **MsiQueryFeatureState** function if you want to determine the availability of a feature without incrementing the usage count.
2. Indicate your application's intent to use a feature by calling the **MsiUseFeature** function.
3. Determine file locations by calling the **MsiGetComponentPath** function.
4. Configure the feature by calling the **MsiConfigureFeature** function.
5. Obtain usage metrics that your application can use by calling the **MsiGetFeatureUsage** function.

The following diagram illustrates the feature request model.



[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installing an Application

You can install entire products with Windows Installer functions. The following steps describe how to install a product.

## ► To install a product

1. Run a product through its installation or uninstallation sequence by calling the **MsiInstallProduct** function.
2. Specify the installation level by calling the **MsiConfigureProduct** function.

You can use the parameters of the **MsiConfigureProduct** function to set the installation state (for example, installed locally; installed to run from the source, etc.) and level, such as the range of product features to be included.

For more information about installing an application, see [Resiliency](#) and [Installation Mechanism](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Reinstalling a Feature or Application

Windows Installer can repair, replace, and verify files contained in an application. A partial or complete application reinstallation might be required if any files or registry entries associated with any feature are missing or corrupt.

When a feature or application is reinstalled, all the services, environment variables, and custom actions belonging to the feature or application are reinstalled as well. Note that this means that any changes made to environment variables between the original installation and the reinstallation are lost.

The following list contains methods of reinstalling a feature or product. The first two methods have been automated by the installer:

- Repair, replace, or verify files by calling the **MsiReinstallFeature** function.
- Reinstall the entire product by calling the **MsiReinstallProduct** function.
- Reinstall, replace, or verify files with an installer UI control button through the **Reinstall ControlEvent**.
- Reinstall, replace, or verify files from a command line by setting the **REINSTALL** property and the **REINSTALLMODE** property.

For more information on reinstalling a feature or application, see [Resiliency](#).

## ▶ To reinstall a product using the installer

- Call **MsiReinstallProduct**.

## ▶ To reinstall a feature using the installer

- Call **MsiReinstallFeature**.

## ▶ To reinstall a product or feature with an installer user interface

1. Add a button to the specified dialog box by adding an entry to the [Control table](#).
2. Add a [ReinstallMode ControlEvent](#) to the ControlEvent table, with the Dialog\_ and Control\_ fields referencing the button control created in step 1. In the Argument field, enter a string containing the letters corresponding to the reinstall modes you want (the acceptable values for this field are identical to those accepted for the **REINSTALLMODE** property). The value in the Ordering column for this event should be 1.
3. Add a [Reinstall ControlEvent](#) event to the [ControlEvent table](#), again referencing the same button control. The Argument field for this event will normally be ALL, to force reinstallation of all features, but you can place the name of a specific feature here. The value in the Ordering column for this event should be 2.
4. Add one more event tied to the same button control, to actually initiate the reinstallation. This can be an EndDialog event (with an argument of Return). More typically, however, a NewDialog event would be used here to jump to an **Are you sure you want to reinstall?** confirmation dialog box. The value in the Ordering column for this event should be 3.

If desired, several **REINSTALL** buttons can be created for a single dialog box, allowing the user to select the type of reinstallation performed. In this case, each button is authored as outlined in the preceding procedure, with a different [ReinstallMode ControlEvent](#) parameter for each button.

Once a particular product has been installed (with some or all of the product's features), a reinstallation can be performed at the command line:

▶ **To reinstall a product or feature from a command line**

1. From the command prompt, specify the **REINSTALL** property.

2. From the command prompt, specify the **REINSTALLMODE** property.

Specifying these properties allows the user to reinstall any or all of the product's features. The type of reinstallation can also be specified. For example, you can specify that only those files that are completely missing should be reinstalled, or that only corrupt files (for example, any executable file whose checksum does not match the actual file contents) be replaced.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Removing an Advertised Application

To remove an application that has been installed in the advertised state, simply uninstall it using [MsiInstallProduct](#) or [MsiConfigureProduct](#). You can also remove an advertised application from the command line. See [Command Line Options](#).

Note that you may be unable to remove an advertised application if you have authored the package in such a way that running an installation is conditional upon the statement **NOT Installed**. An advertised application is not installed on the user's computer and the installer does not set the [Installed](#) Property. The package must be authored so that advertised applications can be uninstalled.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Working with Features and Components

There are several functions that change the installation of product [components and features](#). The following describes how to change features and components.

## ► To change the installation of features and components

1. Set the installation level for a component or feature by calling the [MsiSetInstallLevel](#) function.  
Each feature in a package is assigned an installation level in the [Feature table](#). If the installation level of a feature is lower than the level set by **MsiSetInstallLevel**, the feature is selected for installation. After **MsiSetInstallLevel** is called, you can explicitly change whether a feature is installed.
2. Determine which states are available for a specified feature by calling the [MsiGetFeatureValidStates](#) function.
3. Obtain the disk space requirements for a specified feature and its child features by calling the [MsiGetFeatureCost](#) function.
4. Obtain the current state of a feature or component by calling the [MsiGetFeatureState](#) function or the [MsiGetComponentState](#) function.
5. Change the state of the feature or component with the [MsiSetFeatureState](#) function or the [MsiSetComponentState](#) function.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installing a Missing Component

You can use the Windows Installer to detect missing components or files, and then reinstall features that contain the missing components. Because the Installer installs features and not components, it must first resolve to which component a missing file belongs, and then install the feature that contains the component. If more than one feature is linked to the component, the Installer installs the feature that requires the least disk space.

If you call **MsiGetComponentPath**, you can verify that the key file of a component is present. However, it is still possible that other files belonging to the component are missing. In that scenario, call **MsiInstallMissingFile**. The Installer then resolves to which component the file belongs, and installs the feature that is linked to the component that requires the least disk space.

If the **MsiGetComponentPath** function unexpectedly fails, you must install any missing components.

The following procedure shows you how to install missing components.

## ► To detect and install a missing component

1. Call **MsiGetComponentPath** to verify that the key file of a component is present. However, even if the key file of the component is present, it is still possible that other files belonging to the component are missing.
2. Call the **MsiInstallMissingComponent** function if the feature associated with the component is unknown.
3. Call the **MsiConfigureFeature** or **MsiProvideComponent** function if the feature associated with the component is known.
4. Call **MsiInstallMissingFile** if an application is unable to open a file.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Windows Installer to Inventory Products and Patches

Users and applications with administrative privileges can use Windows Installer functions to inventory the Windows Installer applications, features, components and patches installed on the system.

Beginning with Windows Installer 3.0, users and applications that have administrator privileges can enumerate the Windows Installer applications, features, components and patches installed on the system by all users. Administrators and applications can obtain information on a product or patch for a particular user or all users in the system. Applications can obtain the feature state or component state for a particular user.

The inventory functions available beginning with Windows Installer 3.0 can limit the scope of items to be found by installation context and user context. There are three possible installation contexts: per-user, per-machine, and per-user managed. The user context can be a particular user or all users in the system.

The versions of the Windows Installer inventory functions earlier than Windows Installer 3.0 can only enumerate items installed on the system in the machine context or in the per-user context of the current user. This limitation prevents a complete inventory of all Windows Installer products and patches installed in the system by users other than the current user.

- [Enumerating Products](#)
- [Enumerating Patches](#)
- [Obtaining Product Information](#)
- [Obtaining Patch Information](#)
- [Obtaining Component State Information](#)
- [Obtaining Feature State Information](#)

## Enumerating Products

Use the [MsiEnumProductsEx](#) function to enumerate Windows Installer

applications that are installed in the system. This function can find all the per-machine installations and per-user installations of applications (managed and unmanaged) for the current user and other users in the system. Use the *dwContext* parameter to specify the installation context to be found. You can specify any one or any combination of the possible installation contexts. Use the *szUserSid* parameter to specify the user context of applications to be found.

## Enumerating Patches

Use the [MsiEnumPatchesEx](#) function to find patches applied for an application. This function can find patches applied for a particular application or for all applications in the system. This function can find patches applied to all the per-machine installations and per-user installations of applications (managed and unmanaged) for the current user and other users in the system.

You can use the installation context and user context to restrict the patch enumeration to a particular context or across all contexts. Use the *dwContext* parameter to specify the installation context to be found. You can specify any one or any combination of the possible installation contexts. Use the *szUserSid* parameter to specify the user context of applications to be found.

### ▶ To enumerate patches applied for all products advertised or installed by all users in the system

- Call the **MsiEnumPatchesEx** function.
  - Use NULL for the value of the *szProductCode* parameter.
  - Use "s-1-1-0" for the value of the *szUserSid* parameter.
  - Use "MSIINSTALLCONTEXT\_ALL" for the value of the *dwContext* parameter.

### ▶ To enumerate patches applied for all products advertised or installed by all users in the system

1. Call the **MsiEnumProductsEx** function.

- Use NULL for the value of the *szProductCode* parameter.
- Use "s-1-1-0" for the value of the *szUserSid* parameter.
- Use "MSIINSTALLCONTEXT\_ALL" for the value of the *dwContext* parameter.

The function provides a product code, user context, and installation context for each application found.

2. For each application enumerated in step 1, call **MsiEnumPatchesEx** to enumerate the patches. Use the product codes, user contexts and installation contexts obtained from **MsiEnumProductsEx** for the values of *szProductCode*, *szUserSid*, and *dwContext*, and each **MsiEnumProductsEx** function call.

## Obtaining Product Information

Use the **MsiGetProductInfoEx** function to get information about applications that are advertised or installed on the system, and the properties that can be retrieved. This function can get information for an instance of an application installed under a user account other than the current user, but cannot query an instance of a product that is advertised under a per-user unmanaged context for a user account other than the current user.

You can specify the installation context and user context to restrict information for applications installed in a particular context. Use the *dwContext* parameter to specify the installation context to be found. You can specify only one of the possible installation contexts. Use the *szUserSid* parameter to specify the user context of applications to be found.

## Obtaining Patch Information

An application can call the **MsiGetPatchInfoEx** function to query for information about the application of a patch to a specified instance of a product. Properties such as **LocalPackage**, **Transforms**, and **State** can

be retrieved using this function. Not all property values are guaranteed to be available for per-user unmanaged applications if the user is not currently logged in to the machine. You can specify only one of the possible installation contexts.

You can specify the installation context and user context to restrict information to patches applied to applications installed in a particular context. Use the *dwContext* parameter to specify the installation context to be found. You can specify only one of the possible installation contexts. Use the *szUserSid* parameter to specify the user context of applications to be found.

## Obtaining Component State Information

Applications can call the [MsiQueryComponentState](#) function to get the installed state for a component. This function determines if the component is installed locally or installed to run from the source. The function can query for a component of an instance of an application that is installed under user accounts other than the current user, provided that the product is not advertised under the per-user unmanaged context for a user account other than the current user.

You can specify an installation context and user context to get the state of components for applications installed in a particular context. Use the *dwContext* parameter to specify the installation context to be found. You can specify only one of the possible installation contexts. Use the *szUserSid* parameter to specify the user context of applications to be found.

## Obtaining Feature State Information

Applications can call the [MsiQueryFeatureStateEx](#) function to get the installed state for a product feature. This function determines if the feature is advertised, installed locally, or installed to run from the source. The function can be used to query any feature of an instance of an application installed under the machine account or any context under the current user account or the per-user managed context under any user account other than the current user. This function cannot query for an application installed under the per-user unmanaged context for a user account other than the current user. You can specify only one of the

possible installation contexts.

You can specify an installation context and user context to get the state of features for applications installed in a particular context. Use the *dwContext* parameter to specify the installation context to be found. You can specify only one of the possible installation contexts. Use the *szUserSid* parameter to specify the user context of applications to be found.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Managing Installation Sources

Users and applications with administrative privileges can retrieve and modify network, URL, and media source list information for Windows Installer applications and patches on the system.

**Windows Installer 2.0:** Not supported. Administrators cannot read, reorder, or replace entries in the source list and cannot modify or retrieve source list properties. It is possible to manage Network sources, but not URL or Media sources. Administrators can only manage source lists for per-machine applications or applications installed as per-user for the current user. This prevents administrators using versions earlier than Windows Installer version 3.0 from managing source list information for all users in the system.

**Windows Installer 3.0 and later:** Users and applications that have administrator privileges can retrieve and modify source list information for Windows Installer applications and patches installed on the system for all users. The source list functions can be used to manage source lists and source list properties for network, URL and media sources. The installer can reorder source lists from an external process.

Users and applications that have administrative privileges can read and modify the following types of source list information:

- Source lists for applications and patches installed for all users on the system.
- Source lists for patch sources that exist apart from the application sources.
- Source lists for URL and media sources that exist apart from network sources.
- Source list properties such as **MEDIAPACKAGEPATH**, **DiskPrompt**, **LastUsedSource**, **LastUsedType**, and **PackageName**.

The source lists functions can limit the scope of the source lists found by specifying the installation context and user context. There are three possible installation contexts: per-user (unmanaged), per-machine, and per-user managed. The user context can be a particular user or all users on the system.

Non-administrators cannot modify the source list of an instance of an application or patch that exists under another user's per-user (managed or unmanaged) context. Non-administrators can modify the source lists of an instance of an application or patch installed under the following contexts:

- Their own per-user(unmanaged) context.
- The machine context, but only if the [DisableBrowse](#), [AllowLockdownBrowse](#), and [AlwaysInstallElevated](#) policies allows them to browse for an application or patch source.
- Their own per-user managed context, but only if the [DisableBrowse](#), [AllowLockdownBrowse](#), and [AlwaysInstallElevated](#) policies allows them to browse for an application or patch source.

Administrators can modify any source list that a non-administrator can modify. In addition, administrators and applications that have administrative privileges can modify the source lists of an application or patch installed under the following contexts:

- Per-machine context.
- Their own per-user (unmanaged) or their own per-user managed context.
- Another user's per-user managed context.

**Note** Users and applications that have administrative privileges cannot modify the source list of an instance of an application or patch installed in the per-user (unmanaged) context of another user.

## Managing Network and URL sources for Products and Patches

Use the [MsiSourceListAddSourceEx](#) function to add or reorder the source list of network and URL sources for a patch or application in a particular context. Use the *dwContext* parameter to specify the installation context. Use the *szUserSid* parameter to specify the user context.

Use the [MsiSourceListAddSourceEx](#) function to create a source list for a patch that has not yet been applied to any application in the specified context. This can be useful when registering a patch to have elevated privileges. For more information about registering elevated privileges for a patch, see [Patching Per-User Managed Applications](#).

Use the [MsiSourceListClearSource](#) function to remove an existing source for an application or patch in a specified context. Removing the current source for an application or patch forces the installer to search the source list for a source the next time a source is required.

Use the [MsiSourceListEnumSources](#) function to enumerate sources in the source list of a specified patch or application.

## Managing Media sources for Products and Patches

Use the [MsiSourceListAddMediaDisk](#) function to add or update the disk information of the media source of a registered application or patch. Each entry is uniquely identified by a disk ID. If the disk already exists, it is updated with the new volume label and disk prompt values. If the disk does not exist, a new disk entry is created with the new values.

Use the [MsiSourceListClearMediaDisk](#) function to remove an existing registered disk under the media source for an application or patch in a specific context.

Use the [MsiSourceListEnumMediaDisks](#) function to enumerate a list of disks registered under the media source for an application or patch.

## Retrieval and modification of source list information

Use the [MsiSourceListGetInfo](#) and [MsiSourceListSetInfo](#) functions to retrieve or modify information about the source list for an application or patch in a specific context. Use the *dwContext* parameter to specify the installation context. Use the *szUserSid* parameter to specify the user

context.

Source list properties such as **MEDIAPACKAGEPATH**, **DiskPrompt**, **LastUsedSource**, **LastUsedType**, and **PackageName** can be accessed.

**Note** The **LastUsedType** source list property can only be read. It cannot be set directly using the **MsiSourceListSetInfo** function.

## Clearing the complete source list or forcing a source resolution

Use the **MsiSourceListClearAllEx** function to remove all the existing sources of a given source type for the specified application or patch instance. The patch registration is also removed if the patch is not installed by any application in the same context. Use the *dwContext* parameter to specify the installation context. Use the *szUserSid* parameter to specify the user context.

Use the **MsiSourceListForceResolutionEx** to clear the last used source entry for an application or patch in the specified context. This function removes the registration of the property called **LastUsedSource**. This function does not affect the registered source list. Clearing the **LastUsedSource** registration forces the installer to do a source resolution against the registered sources the next time it requires the source.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer Function Reference

To enable Windows Installer in your application, you must use the installer functions. The tables in this topic identify the functions by category.

## User Interface and Logging Functions

Name	Description
<a href="#">MsiSetInternalUI</a>	Enables the internal user interface of the installer.
<a href="#">MsiSetExternalUI</a>	Enables an external user-interface handler that receives messages in a string format.
<a href="#">MsiSetExternalUIRecord</a>	Enables an external user-interface handler that receives messages in a record format.
<a href="#">MsiEnableLog</a>	Sets the log mode for all installations in the calling process.

## Handle Management Functions

Name	Description
<a href="#">MsiCloseHandle</a>	Closes an open installation handle.
<a href="#">MsiCloseAllHandles</a>	Closes all open installation handles. Do not use for cleanup.

## Installation and Configuration Functions

Name	Description
<a href="#">MsiAdvertiseProduct</a>	Advertises a product.
<a href="#">MsiAdvertiseProductEx</a>	Advertises a product.

<b>MsiAdvertiseScript</b>	Copies an advertise script file into specified locations.
<b>MsiInstallProduct</b>	Installs or removes an application or application suite.
<b>MsiConfigureProduct</b>	Installs or removes an application or application suite.
<b>MsiConfigureProductEx</b>	Installs or removes an application or application suite. A product command-line can be specified.
<b>MsiReinstallProduct</b>	Reinstalls or repairs an installation.
<b>MsiConfigureFeature</b>	Configures the installed state of a feature.
<b>MsiReinstallFeature</b>	Validates or repairs features.
<b>MsiInstallMissingComponent</b>	Installs missing components.
<b>MsiInstallMissingFile</b>	Installs missing files.
<b>MsiNotifySidChange</b>	Notifies and updates the Windows Installer internal information with changes to user SIDs. Available beginning with Windows Installer 3.1.
<b>MsiProcessAdvertiseScript</b>	Processes an advertise script file into specified locations.
<b>MsiSourceListAddSource</b>	Adds or reorders the sources of a patch or product in a specified context.
<b>MsiSourceListAddSourceEx</b>	Adds or reorders the sources of a patch or product in a specified context. Creates a source list for a patch that does not exist in a specified context. Available in Windows Installer 3.0.
<b>MsiSourceListClearSource</b>	Removes an existing source for a product or patch in a specified context. Available in Windows Installer 3.0.

<b>MsiSourceListClearAll</b>	Removes all the existing sources of a specific source type for a specified product instance.
<b>MsiSourceListClearAllEx</b>	Removes all the existing sources of a specific source type for a specified product instance. Available in Windows Installer 3.0.
<b>MsiSourceListForceResolution</b>	Removes the registration of the current source of the product or patch, which is registered as the property "LastUsedSource". This function does not affect the registered source list.
<b>MsiSourceListForceResolutionEx</b>	Removes the registration of the current source of the product or patch, which is registered as the property "LastUsedSource". This function does not affect the registered source list. Available in Windows Installer 3.0.
<b>MsiSourceListGetInfo</b>	Retrieves information about the source list for a product or patch in a specific context.
<b>MsiSourceListSetInfo</b>	Sets the most recently used source for a product or patch in a specified context. Available in Windows Installer 3.0.
<b>MsiSourceListEnumMediaDisks</b>	Enumerates the list of disks registered for the media source for a patch or product. Available in Windows Installer 3.0.
<b>MsiSourceListAddMediaDisk</b>	Adds or updates a disk of the media source of a registered product or patch. Available in Windows Installer 3.0.
<b>MsiSourceListClearMediaDisk</b>	Removes an existing registered disk under the media source for a product or patch in a specific context. Available in Windows Installer 3.0.

<b>MsiSourceListEnumSources</b>	Enumerates the sources in the source list of a specified patch or product. Available in Windows Installer 3.0.
---------------------------------	--

## Component-Specific Functions

<b>Name</b>	<b>Description</b>
<b>MsiProvideAssembly</b>	Installs and returns the full component path for an assembly.
<b>MsiProvideComponent</b>	Installs and returns the full component path of a component.
<b>MsiProvideQualifiedComponent</b>	Installs and returns the full component path of a qualified component.
<b>MsiProvideQualifiedComponentEx</b>	Installs and returns the full component path of a qualified component that is published by a product.
<b>MsiGetComponentPath</b>	Returns the full path or registry key to an installed component.
<b>MsiGetComponentPathEx</b>	Returns the full path or registry key to an installed component across user accounts and installation context.  <b>Windows Installer 4.5 and earlier:</b> Not supported.
<b>MsiLocateComponent</b>	Returns the full path to an installed component without a product code.
<b>MsiQueryComponentState</b>	Returns the installed state for a component. Can query components of an instance of a product installed

under user accounts other than the current user. Available in Windows Installer 3.0 or later.

## Application-Only Functions

Name	Description
<b>MsiCollectUserInfo</b>	Stores user information from an installation wizard.
<b>MsiUseFeature</b>	Increments usage count for a feature and indicates installation state.
<b>MsiUseFeatureEx</b>	Increments usage count for a feature and indicates installation state.
<b>MsiGetProductCode</b>	Returns product code using the component code.

## System Status Functions

Name	Description
<b>MsiEnumProducts</b>	Enumerates advertised products.
<b>MsiEnumProductsEx</b>	Enumerates through all the instances of advertised or installed products in a specified context. Available in Windows Installer 3.0 or later.
<b>MsiEnumRelatedProducts</b>	Enumerates currently installed products having a specified upgrade code.
<b>MsiEnumFeatures</b>	Enumerates published features.
<b>MsiEnumComponents</b>	Enumerates the installed components.
<b>MsiEnumComponentsEx</b>	Enumerates the installed components across user accounts and installation context.

	<b>Windows Installer 4.5 and earlier:</b> Not supported.
<b>MsiEnumClients</b>	Enumerates the clients of an installed component.
<b>MsiEnumClientsEx</b>	Enumerates the clients of an installed component across user accounts and installation context.  <b>Windows Installer 4.5 and earlier:</b> Not supported.
<b>MsiEnumComponentQualifiers</b>	Enumerates the advertised qualifiers for a component.
<b>MsiQueryFeatureState</b>	Returns the installed state of a feature.
<b>MsiQueryFeatureStateEx</b>	Returns the installed state for a product feature. Can query features of an instance of a product installed under user accounts other than the current user. Available in Windows Installer 3.0 or later.
<b>MsiQueryProductState</b>	Returns the installed state for an application or application suite.
<b>MsiGetFeatureUsage</b>	Returns usage metrics for a feature.
<b>MsiGetProductInfo</b>	Returns product information for published and installed products.
<b>MsiGetProductInfoEx</b>	Returns product information for advertised and installed products. Can retrieve information on an instance of a product installed under a user account other than the current user. Available in Windows Installer 3.0 or later.
<b>MsiGetUserInfo</b>	Returns registered user information for an installed product.

## Product Query Functions

Name	Description
<b>MsiOpenProduct</b>	Opens a product to use with the functions that access the database.
<b>MsiOpenPackage</b>	Opens a package to use with the functions that access the database.
<b>MsiOpenPackageEx</b>	Opens a package to use with the functions that access the database.
<b>MsiIsProductElevated</b>	Checks whether the product is installed with elevated privileges.
<b>MsiGetProductInfoFromScript</b>	Returns product information for an installer script file.
<b>MsiGetProductProperty</b>	Retrieves properties in the product database.
<b>MsiGetShortcutTarget</b>	Examines a shortcut and returns its product, feature name, and component if available.
<b>MsiGetFeatureInfo</b>	Returns descriptive information for a feature.
<b>MsiVerifyPackage</b>	Verifies that a specified file is an installation package.

## Patching Functions

Name	Description
<b>MsiApplyPatch</b>	Invokes an installation and applies a patch package.
<b>MsiEnumPatches</b>	Returns the GUID for each patch that is applied to a product, and a list of transforms from each patch that apply to

	the product.
<b>MsiGetPatchInfo</b>	Returns information about a patch.
<b>MsiRemovePatches</b>	Uninstalls a patch from a product. Available in Windows Installer 3.0.
<b>MsiDeterminePatchSequence</b>	Determines the best application sequence for a set of patches and product. Available in Windows Installer 3.0.
<b>MsiApplyMultiplePatches</b>	Applies one or more patches to products. Available in Windows Installer 3.0.
<b>MsiEnumPatchesEx</b>	Enumerates all patches applied for a product in a particular context or across all contexts. Available in Windows Installer 3.0.
<b>MsiGetPatchFileList</b>	When provided a list of .msp files this function retrieves the list of files that can be updated by the patches for the target. Available in Windows Installer 4.0.
<b>MsiGetPatchInfoEx</b>	Queries for information about the application of a specified patch to a specified product. Available in Windows Installer 3.0.
<b>MsiExtractPatchXMLData</b>	Extracts information from a patch. Available in Windows Installer 3.0.
<b>MsiDetermineApplicablePatches</b>	Determines the best set of patches required to update a product or set of products. Available in Windows Installer 3.0.

## File Query Functions

---

Name	Description
<b>MsiGetFileHash</b>	Takes the path to a file and returns a 128-bit hash of that file.
<b>MsiGetFileSignatureInformation</b>	Takes the path to a file that has been digitally signed and returns the file's signer certificate and hash.
<b>MsiGetFileVersion</b>	Returns the version string and language string.

## Transaction Management Functions

Name	Description
<b>MsiBeginTransaction</b>	Starts transaction processing of a multiple-package installation and returns an identifier for the transaction. This function is available beginning with Windows Installer 4.5.
<b>MsiJoinTransaction</b>	Requests that the Windows Installer make the current process the owner of the transaction installing a multi-package installation. This function is available beginning with Windows Installer 4.5.
<b>MsiEndTransaction</b>	Commits or rolls back all the installations belonging to the transaction. This function is available beginning with Windows Installer 4.5.

## Database Functions

In addition to the Windows Installer functions identified in the previous tables, you can manipulate information in the installation database by using the database access functions that are described in the [Database Functions](#) section.

## Installer Structures

In addition, some information in the installation database is handled using the structures described in the [Installer Structures](#) section.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiAdvertiseProduct Function

The **MsiAdvertiseProduct** function generates an advertise script or advertises a product to the computer. The **MsiAdvertiseProduct** function enables the installer to write to a script the registry and shortcut information used to assign or publish a product. The script can be written to be consistent with a specified platform by using [MsiAdvertiseProductEx](#).

## Syntax

```
C++UINT MsiAdvertiseProduct(  
    __in LPCTSTR szPackagePath,  
    __in LPCTSTR szScriptfilePath,  
    __in LPCTSTR szTransforms,  
    __in LANGID lgidLanguage  
);
```

## Parameters

*szPackagePath* [in]

The full path to the package of the product being advertised.

*szScriptfilePath* [in]

The full path to script file that will be created with the advertise information. To advertise the product locally to the computer, set `ADVERTISEFLAGS_MACHINEASSIGN` or `ADVERTISEFLAGS_USERASSIGN`.

Flag	Meaning
<code>ADVERTISEFLAGS_MACHINEASSIGN</code> 0	Set to advertise a per-machine installation of the product available to all users.
<code>ADVERTISEFLAGS_USERASSIGN</code> 1	Set to advertise a per-user installation of the product available to a particular

user.
-------

### *szTransforms* [in]

A semicolon-delimited list of transforms to be applied. The list of transforms can be prefixed with the @ or | character to specify the secure caching of transforms. The @ prefix specifies secure-at-source transforms and the | prefix indicates secure full path transforms. For more information, see [Secured Transforms](#). This parameter may be null.

### *lgidLanguage* [in]

The installation language to use if the source supports multiple languages.

## Return Value

Value	Meaning
ERROR_SUCCESS	The function completed successfully.
An error relating to an action	See <a href="#">Error Codes</a> .
<a href="#">Initialization Error</a>	An initialization error occurred.
ERROR_CALL_NOT_IMPLEMENTED	This error is returned if an attempt is made to generate an advertise script on any platform other than Windows 2000 or Windows XP. Advertisement to the local computer is supported on all platforms.

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2
---

<b>Version</b>	or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiAdvertiseProductW</b> (Unicode) and <b>MsiAdvertiseProductA</b> (ANSI)

## See Also

[Multiple-Package Installations](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiAdvertiseProductEx Function

The **MsiAdvertiseProductEx** function generates an advertise script or advertises a product to the computer. This function enables Windows Installer to write to a script the registry and shortcut information used to assign or publish a product. The script can be written to be consistent with a specified platform by using **MsiAdvertiseProductEx**. The **MsiAdvertiseProductEx** function provides the same functionality as [MsiAdvertiseProduct](#).

## Syntax

```
C++UINT MsiAdvertiseProductEx(  
    __in LPCTSTR szPackagePath,  
    __in LPCTSTR szScriptfilePath,  
    __in LPCTSTR szTransforms,  
    __in LANGID lgidLanguage,  
    __in DWORD dwPlatform,  
    __in DWORD dwOptions  
);
```

## Parameters

*szPackagePath* [in]

The full path to the package of the product being advertised.

*szScriptfilePath* [in]

The full path to the script file to be created with the advertised information. To advertise the product locally to the computer, set **ADVERTISEFLAGS\_MACHINEASSIGN** or **ADVERTISEFLAGS\_USERASSIGN**.

Flag	Meaning
<b>ADVERTISEFLAGS_MACHINEASSIGN</b> 0	Set to advertise a per-computer installation of the product available to all users.

ADVERTISEFLAGS_USERASSIGN 1	Set to advertise a per-user installation of the product available to a particular user.
--------------------------------	---

*szTransforms* [in]

A semicolon–delimited list of transforms to be applied. The list of transforms can be prefixed with the @ or | character to specify the secure caching of transforms. The @ prefix specifies secure-at-source transforms and the | prefix indicates secure full path–transforms. For more information, see [Secured Transforms](#). This parameter may be null.

*lgidLanguage* [in]

The language to use if the source supports multiple languages.

*dwPlatform* [in]

Bit flags that control for which platform the installer should create the script. This parameter is ignored if *szScriptfilePath* is null. If *dwPlatform* is zero (0), then the script is created based on the current platform. This is the same functionality as [MsiAdvertiseProduct](#). If *dwPlatform* is 1 or 2, the installer creates script for the specified platform.

Flag	Meaning
none 0	Creates a script for the current platform.
MSIARCHITECTUREFLAGS_X86 1	Creates a script for the x86 platform.
MSIARCHITECTUREFLAGS_IA64 2	Creates a script for the Intel Itanium Processor Family (IPF).
MSIARCHITECTUREFLAGS_AMD64 4	Creates a script for the x64 platform.

### *dwOptions* [in]

Bit flags that specify extra advertisement options. Nonzero value is only available in Windows Installer versions shipped with Windows Server 2003 and Windows XP with SP1 and later.

Flag	Meaning
MSIADVERTISEOPTIONS_INSTANCE 1	Multiple instances through product code changing transform support flag. Advertises a new instance of the product. Requires that the <i>szTransforms</i> parameter includes the instance transform that changes the product code. For more information, see <a href="#">Installing Multiple Instances of Products and Patches</a> .

### Return Value

Value	Meaning
ERROR_SUCCESS	The function completes successfully.
An error that relates to an action	For more information, see <a href="#">Error Codes</a> .
<a href="#">Initialization Error</a>	An initialization error has occurred.
ERROR_CALL_NOT_IMPLEMENTED	This error is returned if an attempt is made to generate an advertise script on any platform other than Windows 2000 or Windows XP. Advertisement to

the local computer is supported on all platforms.

## Remarks

Multiple instances through product code–changing transforms is only available for Windows Installer versions shipping with Windows Server 2003 and Windows XP with SP1 and later.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiAdvertiseProductExW</b> (Unicode) and <b>MsiAdvertiseProductExA</b> (ANSI)

## See Also

[Multiple-Package Installations](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiAdvertiseScript Function

The **MsiAdvertiseScript** function copies an advertised script file into the specified locations.

## Syntax

```
C++UINT MsiAdvertiseScript(  
    __in LPCTSTR szScriptFile,  
    __in DWORD dwFlags,  
    __in PHKEY phRegData,  
    __in BOOL fRemoveItems  
);
```

## Parameters

*szScriptFile* [in]

The full path to a script file generated by **MsiAdvertiseProduct** or **MsiAdvertiseProductEx**.

*dwFlags* [in]

The following bit flags from SCRIPTFLAGS control advertisement. The value of *dwFlags* can be a combination of the following values.

Flag	Meaning
SCRIPTFLAGS_CACHEINFO 0x001	Include this flag if or removed.
SCRIPTFLAGS_SHORTCUTS 0x004	Include this flag if created or removed
SCRIPTFLAGS_MACHINEASSIGN 0x008	Include this flag if a computer.
SCRIPTFLAGS_REGDATA_CNFGINFO 0x020	Include this flag if management information needs to be written
SCRIPTFLAGS_VALIDATE_TRANSFORMS_LIST	Include this flag to

0x040	transforms listed in registered transform transform conflicts comparison that is evaluated between installations across transforms in the sc transforms registered function returns ERROR_INSTALL
SCRIPTFLAGS_REGDATA_CLASSINFO 0x080	Include this flag if the registry related written or removed
SCRIPTFLAGS_REGDATA_EXTENSIONINFO 0x100	Include this flag if the registry related written or removed
SCRIPTFLAGS_REGDATA_APPINFO 0x180	Include this flag if information in the removed.
SCRIPTFLAGS_REGDATA 0x1A0	Include this flag if information in the removed.

### *phRegData* [in]

A registry key under which temporary information about registry data is to be written. If this parameter is null, the registry data is placed under the appropriate key, based on whether the advertisement is per-user or per-machine. If this parameter is non-null, the script will write the registry data under the specified registry key rather than the normal location. In this case, the application will not get advertised to the user.

Note that this registry key cannot be used when generating an advertisement of a product for a user or a computer because the

provider of the registry key generally deletes the key. The registry key is located outside of the normal registry locations for shell, class, and .msi configuration information and it is not under **HKEY\_CLASSES\_ROOT**. This registry key is only intended for getting temporary information about registry data in a script.

*fRemoveItems* [in]

TRUE if specified items are to be removed instead of being created.

## Return Value

Value	Meaning
ERROR_SUCCESS	The function completed successfully.
ERROR_ACCESS_DENIED	The calling process was not running under the LocalSystem account.
An error relating to an action	See <a href="#">Error Codes</a> .
<a href="#">Initialization Error</a>	An error relating to initialization occurred.
ERROR_CALL_NOT_IMPLEMENTED	This function is only available on Windows 2000 and Windows XP.

## Remarks

The process calling this function must be running under the LocalSystem account. To advertise an application for per-user installation to a targeted user, the thread that calls this function must impersonate the targeted user. If the thread calling this function is not impersonating a targeted user, the application is advertised to all users for installation with elevated privileges.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiAdvertiseScriptW</b> (Unicode) and <b>MsiAdvertiseScriptA</b> (ANSI)

## See Also

[Installation Context](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiApplyPatch Function

For each product listed by the patch package as eligible to receive the patch, the **MsiApplyPatch** function invokes an installation and sets the **PATCH** property to the path of the patch package.

## Syntax

```
C++UINT MsiApplyPatch(  
    __in LPCTSTR szPatchPackage,  
    __in LPCTSTR szInstallPackage,  
    __in INSTALLTYPE eInstallType,  
    __in LPCTSTR szCommandLine  
);
```

## Parameters

*szPatchPackage* [in]

A null-terminated string specifying the full path to the patch package.

*szInstallPackage* [in]

If *eInstallType* is set to `INSTALLTYPE_NETWORK_IMAGE`, this parameter is a null-terminated string that specifies a path to the product that is to be patched. The installer applies the patch to every eligible product listed in the patch package if *szInstallPackage* is set to null and *eInstallType* is set to `INSTALLTYPE_DEFAULT`.

If *eInstallType* is `INSTALLTYPE_SINGLE_INSTANCE`, the installer applies the patch to the product specified by *szInstallPackage*. In this case, other eligible products listed in the patch package are ignored and the *szInstallPackage* parameter contains the null-terminated string representing the product code of the instance to patch. This type of installation requires the installer running Windows Server 2003 or Windows XP.

*eInstallType* [in]

This parameter specifies the type of installation to patch.

Type of installation	Meaning

INSTALLTYPE_NETWORK_IMAGE	Specifies an administrative install. In this case, <i>szInstallPackage</i> must be set to a package path. A value of <code>INSTALLTYPE_NETWORK_IMAGE</code> sets this for an administrative installation.
INSTALLTYPE_DEFAULT	Searches system for products to patch. In this case, <i>szInstallPackage</i> must be set to 0.
INSTALLTYPE_SINGLE_INSTANCE	Patch the product specified by <i>szInstallPackage</i> . <i>szInstallPackage</i> must be set to the product code of the instance to patch. This type of installation requires the installer running Windows Server 2003 or Windows XP with Service Pack 2. For more information see, <a href="#">Installing Multiple Instances of Products and Patches</a> .

*szCommandLine* [in]

A null-terminated string that specifies command line property settings. See About [Properties](#) and [Setting Public Property Values on the Command Line](#). See the Remarks section.

## Return Value

Value	Meaning
ERROR_SUCCESS	The function completed successfully.
ERROR_PATCH_PACKAGE_OPEN_FAILED	Patch package could not be opened.
ERROR_PATCH_PACKAGE_INVALID	The patch package is invalid.

ERROR_PATCH_PACKAGE_UNSUPPORTED	The patch package is unsupported.
An error relating to an action	See <a href="#">Error Codes</a> .
<a href="#">Initialization Error</a>	An initialization error occurred.

## Remarks

Because the list delimiter for transforms, sources, and patches is a semicolon, this character should not be used for file names or paths.

## Note

You must set the **REINSTALL** property on the command line when applying a [small update](#) or [minor upgrade](#) patch. Without this property, the patch is registered on the system but cannot update files. For patches that do not use a [Custom Action Type 51](#) to automatically set the **REINSTALL** and **REINSTALLMODE** properties, the **REINSTALL** property must be explicitly set with the *szCommandLine* parameter. Set the **REINSTALL** property to list the features affected by the patch, or use a practical default setting of "REINSTALL=ALL". The default value of the **REINSTALLMODE** property is "omus". Beginning with Windows Installer version 3.0, the **REINSTALL** property is configured by the installer and does not need to be set on the command line.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiApplyPatchW</b> (Unicode) and <b>MsiApplyPatchA</b> (ANSI)

## See Also

[Error Codes](#)

[Initialization Error](#)

[Multiple-Package Installations](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiApplyMultiplePatches Function

The **MsiApplyMultiplePatches** function applies one or more patches to products eligible to receive the patches. The **MsiApplyMultiplePatches** function sets the **PATCH** property with a list of patches delimited by semicolons and invokes the patching of the target products. Other properties can be set using a properties list.

## Syntax

```
C++UINT MsiApplyMultiplePatches(  
    __in      LPCTSTR szPatchPackages,  
    __in_opt  LPCTSTR szProductCode,  
    __in_opt  LPCTSTR szPropertiesList  
);
```

## Parameters

### *szPatchPackages* [in]

A semicolon-delimited list of the paths to patch files as a single string. For example: `""c:\sus\download\cache\Office\sp1.msp;  
c:\sus\download\cache\Office\QFE1.msp;  
c:\sus\download\cache\Office\QFEn.msp""`

### *szProductCode* [in, optional]

This parameter is the **ProductCode** GUID of the product to be patched. The user or application calling **MsiApplyMultiplePatches** must have privileges to apply patches. When this parameter is null, the patches are applied to all eligible products. When this parameter is non-null, the patches are applied only to the specified product.

### *szPropertiesList* [in, optional]

A null-terminated string that specifies command-line property settings used during the patching of products. If there are no command-line property settings, pass in a NULL pointer. An empty string is an invalid parameter. These properties are shared by all target products. For more information, see [About Properties](#) and [Setting Public Property Values on the Command Line](#).

**Note** The properties list should not contain the **PATCH** property. If the **PATCH** property is set in the command line the value is ignored and is overwritten with the patches being applied.

## Return Value

The **MsiApplyMultiplePatches** function returns the following values.

Value	Meaning
ERROR_INVALID_PARAMETER	Some arguments passed in are incorrect or contradicting.
ERROR_SUCCESS	The function completed and all products are successfully patched. ERROR_SUCCESS is returned only if all the products eligible for the patches are patched successfully. If none of the new patches are applicable, <b>MsiApplyMultiplePatches</b> returns ERROR_SUCCESS and product state remains unchanged.
ERROR_SUCCESS_REBOOT_INITIATED	The restart initiated by the last transaction terminated this call to <b>MsiApplyMultiplePatches</b> . All the target products may not have been patched.
ERROR_SUCCESS_REBOOT_REQUIRED	The restart required by the last transaction terminated this call to <b>MsiApplyMultiplePatches</b> . All target products may not

	have been patched.
ERROR_PATCH_PACKAGE_OPEN_FAILED	One of the patch packages provide could not be opened.
ERROR_PATCH_PACKAGE_INVALID	One of the patch packages provide is not a valid one.
ERROR_PATCH_PACKAGE_UNSUPPORTED	One of the patch packages is unsupported.
Any error in Winerror.h	Implies possible partial completion or that one or more transactions failed.

## Remarks

**Windows Installer 2.0:** Not supported. This function is available beginning with Windows Installer 3.0.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

**Unicode and ANSI  
names**

**MsiApplyMultiplePatchesW** (Unicode) and  
**MsiApplyMultiplePatchesA** (ANSI)

## See Also

[About Properties](#)

[Setting Public Property Values on the Command Line](#)

**PATCH**

**ProductCode**

[Multiple-Package Installations](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiBeginTransaction Function

The **MsiBeginTransaction** function starts *transaction processing* of a multiple-package installation and returns an identifier for the transaction. The **MsiEndTransaction** function ends the transaction.

**Windows Installer 4.0 and earlier:** Not supported. This function is available beginning with Windows Installer 4.5.

## Syntax

```
C++UINT WINAPI MsiBeginTransaction(  
    __in    LPCWSTR szTransactionName,  
    __in    DWORD dwTransactionAttributes,  
    __out   MSIHANDLE *hTransactionID,  
    __out   HANDLE *phChangeOfOwnerEvent  
);
```

## Parameters

*szTransactionName* [in]

Name of the multiple-package installation.

*dwTransactionAttributes* [in]

Attributes of the multiple-package installation.

Value	Meaning
0	When 0 or no value is set it Windows Installer closes the UI from the previous installation.
MSITRANSACTION_CHAIN_EMBEDDEDUI	Set this attribute to request that the Windows Installer not shutdown the embedded UI until

the transaction is complete.

*hTransactionID* [out]

Transaction ID is a **MSIHANDLE** value that identifies the transaction. Only one process can own a transaction at a time.

*phChangeOfOwnerEvent* [out]

This parameter returns a handle to an event that is set when the **MsiJoinTransaction** function changes the owner of the transaction to a new owner. The current owner can use this to determine when ownership of the transaction has changed. Leaving a transaction without an owner will roll back the transaction.

## Return Value

The **MsiBeginTransaction** function returns the following values.

Value	Meaning
ERROR_INSTALL_SERVICE_FAILURE	The installation service could not be accessed. This function requires the Windows Installer service.
ERROR_INSTALL_ALREADY_RUNNING	Only one transaction can be open on a system at a time. The function returns this error if called while another transaction is running.
ERROR_INVALID_PARAMETER	An invalid parameter is passed to the function.
ERROR_ROLLBACK_DISABLED	<b>Rollback Installations</b> have been disabled by the <b>DISABLEROLLBACK</b> property or <b>DisableRollback</b> policy.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.5 on Windows Vista, Windows XP, Windows Server 2003, and Windows Server 2008. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiBeginTransactionW</b> (Unicode) and <b>MsiBeginTransactionA</b> (ANSI)

## See Also

### [Multiple Package Installations](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiCloseAllHandles Function

The **MsiCloseAllHandles** function closes all open installation handles allocated by the current thread. This is a diagnostic function and should not be used for cleanup.

## Syntax

```
C++UINT MsiCloseAllHandles(void);
```

## Parameters

This function has no parameters.

## Return Value

This function returns 0 if all handles are closed. Otherwise, the function returns the number of handles open prior to its call.

## Remarks

**MsiCloseAllHandles** only closes handles allocated by the calling thread, and does not affect handles allocated by other threads, such as the install handle passed to custom actions.

The **MsiOpenPackage** function opens a handle to a package and the **MsiOpenProduct** function opens a handle to a product. These functions are for use with functions that access the product database.

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for

<b>Version</b>	information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[Handle Management Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiCloseHandle Function

The **MsiCloseHandle** function closes an open installation handle.

## Syntax

```
C++UINT MsiCloseHandle(  
    __in MSIHANDLE hAny  
);
```

## Parameters

*hAny* [in]

Specifies any open installation handle.

## Return Value

Value	Meaning
ERROR_INVALID_HANDLE	An invalid handle was passed to the function.
ERROR_SUCCESS	The function succeeded.

## Remarks

**MsiCloseHandle** must be called from the same thread that requested the creation of the handle.

The following functions supply handles that should be closed after use by calling **MsiCloseHandle**:

- MsiCreateRecord**
- MsiGetActiveDatabase**
- MsiGetLastErrorRecord**
- MsiOpenPackage**
- MsiOpenProduct**

**MsiOpenDatabase**  
**MsiDatabaseOpenView**  
**MsiViewFetch**  
**MsiViewGetColumnInfo**  
**MsiDatabaseGetPrimaryKeys**  
**MsiGetSummaryInformation**  
**MsiEnableUIPreview**

Note that when writing custom actions, it is recommended to use variables of type PMSIHANDLE because the installer closes PMSIHANDLE objects as they go out of scope, whereas you must close MSIHANDLE objects by calling **MsiCloseHandle**.

For example, if you use code like this:

```
MSIHANDLE hRec = MsiCreateRecord(3);
```

Change it to:

```
PMSIHANDLE hRec = MsiCreateRecord(3);
```

## Requirements

	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Version</b>	
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[Handle Management Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiCollectUserInfo Function

The [MsiCollectUserInfo](#) function obtains and stores the user information and product ID from an installation wizard.

## Syntax

```
C++UINT MsiCollectUserInfo(  
    __in LPCTSTR szProduct  
);
```

## Parameters

*szProduct* [in]

Specifies the product code of the product for which the user information is collected.

## Return Value

Value	Meaning
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_SUCCESS	The function succeeded.
An error relating to an action	See <a href="#">Error Codes</a> .
<a href="#">Initialization Error</a>	An error relating to initialization occurred.

## Remarks

The **MsiCollectUserInfo** function is typically called by an application during the first run of the application. The application first calls [MsiGetUserInfo](#). If that call fails, the application calls

**MsiCollectUserInfo.** **MsiCollectUserInfo** opens the product's installation package and invokes a wizard sequence that collects user information. Upon completion of the sequence, user information is registered. Since this API requires an authored user interface, the user interface level should be set to full by calling **MsiSetInternalUI** as `INSTALLUILEVEL_FULL`.

The **MsiCollectUserInfo** invokes a [FirstRun Dialog](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiCollectUserInfoW</b> (Unicode) and <b>MsiCollectUserInfoA</b> (ANSI)

## See Also

[Application-Only Functions](#)

[Error Codes](#)

[Initialization Error](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiConfigureFeature Function

The **MsiConfigureFeature** function configures the installed state for a product feature.

## Syntax

```
C++UINT MsiConfigureFeature(  
    __in LPCTSTR szProduct,  
    __in LPCTSTR szFeature,  
    __in INSTALLSTATE eInstallState  
);
```

## Parameters

*szProduct* [in]

Specifies the product code for the product to be configured.

*szFeature* [in]

Specifies the feature ID for the feature to be configured.

*eInstallState* [in]

Specifies the installation state for the feature. This parameter must be one of the following values.

Value	Meaning
INSTALLSTATE_ADVERTISED	The feature is advertised
INSTALLSTATE_LOCAL	The feature is installed locally.
INSTALLSTATE_ABSENT	The feature is uninstalled.
INSTALLSTATE_SOURCE	The feature is installed to run from source.
INSTALLSTATE_DEFAULT	The feature is installed to its default location.

## Return Value

Value	Meaning
ERROR_INVALID_PARAMETER	An invalid parameter is passed to the function.
ERROR_SUCCESS	The function succeeds.
An error relating to an action	For more information, see <a href="#">Error Codes</a> .
<a href="#">Initialization Error</a>	An error that relates to the initialization has occurred.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiConfigureFeatureW</b> (Unicode) and <b>MsiConfigureFeatureA</b> (ANSI)

## See Also

[Displayed Error Messages](#)  
[Error Codes](#)

[Initialization Error](#)  
[Installation and Configuration Functions](#)  
[Multiple-Package Installations](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiConfigureProduct Function

The **MsiConfigureProduct** function installs or uninstalls a product.

## Syntax

```
C++UINT MsiConfigureProduct(  
    __in LPCTSTR szProduct,  
    __in int iInstallLevel,  
    __in INSTALLSTATE eInstallState  
);
```

## Parameters

*szProduct* [in]

Specifies the product code for the product to be configured.

*iInstallLevel* [in]

Specifies how much of the product should be installed when installing the product to its default state. The *iInstallLevel* parameter is ignored, and all features are installed, if the *eInstallState* parameter is set to any other value than `INSTALLSTATE_DEFAULT`.

This parameter can be one of the following values.

Value	Meaning
<code>INSTALLLEVEL_DEFAULT</code>	The authored default features are installed.
<code>INSTALLLEVEL_MINIMUM</code>	Only the required features are installed. You can specify a value between <code>INSTALLLEVEL_MINIMUM</code> and <code>INSTALLLEVEL_MAXIMUM</code> to install a subset of available features.
<code>INSTALLLEVEL_MAXIMUM</code>	All features are installed. You can specify a value between <code>INSTALLLEVEL_MINIMUM</code> and

INSTALLLEVEL\_MAXIMUM to install a subset of available features.

### *eInstallState* [in]

Specifies the installation state for the product. This parameter can be one of the following values.

<b>Value</b>	<b>Meaning</b>
INSTALLSTATE_LOCAL	The product is to be installed with all features installed locally.
INSTALLSTATE_ABSENT	The product is uninstalled.
INSTALLSTATE_SOURCE	The product is to be installed with all features installed to run from source.
INSTALLSTATE_DEFAULT	The product is to be installed with all features installed to the default states specified in the <a href="#">Feature Table</a> .
INSTALLSTATE_ADVERTISED	The product is advertised.

### **Return Value**

<b>Value</b>	<b>Meaning</b>
ERROR_INVALID_PARAMETER	An invalid parameter is passed to the function.
ERROR_SUCCESS	The function succeeds.
An error that relates to an action	For more information, see <a href="#">Error Codes</a> .
<a href="#">Initialization Error</a>	An error that relates to initialization.

## Remarks

The [MsiConfigureProduct](#) function displays the user interface (UI) using the current settings. User interface settings can be changed by using [MsiSetInternalUI](#), [MsiSetExternalUI](#) or [MsiSetExternalUIRecord](#).

The *installLevel* parameter is ignored, and all features of the product are installed, if the *installState* parameter is set to any other value than `INSTALLSTATE_DEFAULT`. To control the installation of individual features when the *installState* parameter is not set to `INSTALLSTATE_DEFAULT`, use [MsiConfigureFeature](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiConfigureProductW</b> (Unicode) and <b>MsiConfigureProductA</b> (ANSI)

## See Also

[Multiple-Package Installations](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiConfigureProductEx Function

The **MsiConfigureProductEx** function installs or uninstalls a product. A product command line can also be specified.

## Syntax

```
C++UINT MsiConfigureProductEx(  
    __in LPCTSTR szProduct,  
    __in int iInstallLevel,  
    __in INSTALLSTATE eInstallState,  
    __in LPCTSTR szCommandLine  
);
```

## Parameters

*szProduct* [in]

Specifies the product code for the product to be configured.

*iInstallLevel* [in]

Specifies how much of the product should be installed when installing the product to its default state. The *iInstallLevel* parameters are ignored, and all features are installed, if the *eInstallState* parameter is set to any value other than `INSTALLSTATE_DEFAULT`.

This parameter can be one of the following values.

Value	Meaning
<code>INSTALLLEVEL_DEFAULT</code>	The authored default features are installed.
<code>INSTALLLEVEL_MINIMUM</code>	Only the required features are installed. You can specify a value between <code>INSTALLLEVEL_MINIMUM</code> and <code>INSTALLLEVEL_MAXIMUM</code> to install a subset of available features.
<code>INSTALLLEVEL_MAXIMUM</code>	All features are installed. You can

	specify a value between <code>INSTALLLEVEL_MINIMUM</code> and <code>INSTALLLEVEL_MAXIMUM</code> to install a subset of available features.
--	--

*eInstallState* [in]

Specifies the installation state for the product. This parameter can be one of the following values.

Value	Meaning
<code>INSTALLSTATE_LOCAL</code>	The product is to be installed with all features installed locally.
<code>INSTALLSTATE_ABSENT</code>	The product is uninstalled.
<code>INSTALLSTATE_SOURCE</code>	The product is to be installed with all features installed to run from source.
<code>INSTALLSTATE_DEFAULT</code>	The product is to be installed with all features installed to the default states specified in the <a href="#">Feature Table</a> .
<code>INSTALLSTATE_ADVERTISED</code>	The product is advertised.

*szCommandLine* [in]

Specifies the command-line property settings. This should be a list of the format *Property=Setting Property=Setting*. For more information, see [About Properties](#).

**Return Value**

Value	Meaning
<code>ERROR_INVALID_PARAMETER</code>	An invalid parameter is passed to the function.

ERROR_SUCCESS	The function succeeded.
An error that relates to an action	For more information, see <a href="#">Error Codes</a> .
<a href="#">Initialization Error</a>	An error relating to initialization occurred.

## Remarks

The command line passed in as *szCommandLine* can contain any of the [Feature Installation Options Properties](#). In this case, the *eInstallState* passed must be INSTALLSTATE\_DEFAULT.

The *iInstallLevel* parameter is ignored and all features of the product are installed if the *eInstallState* parameter is set to any other value than INSTALLSTATE\_DEFAULT. To control the installation of individual features when the *eInstallState* parameter is not set to INSTALLSTATE\_DEFAULT use [MsiConfigureFeature](#).

The [MsiConfigureProductEx](#) function displays the user interface using the current settings. User interface settings can be changed with [MsiSetInternalUI](#), [MsiSetExternalUI](#), or [MsiSetExternalUIRecord](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

**Unicode and ANSI  
names**

**MsiConfigureProductExW** (Unicode) and  
**MsiConfigureProductExA** (ANSI)

## See Also

[Installation and Configuration Functions](#)

[Error Codes](#)

[Initialization Error](#)

[Displayed Error Messages](#)

[Multiple-Package Installations](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiDetermineApplicablePatches Function

The **MsiDetermineApplicablePatches** function takes a set of patch files, XML files, and XML blobs and determines which patches apply to a specified Windows Installer package and in what sequence. The function can account for superseded or obsolete patches. This function does not account for products or patches that are installed on the system that are not specified in the set.

## Syntax

```
C++UINT MsiDetermineApplicablePatches(  
    __in LPCTSTR szProductPackagePath,  
    __in DWORD cPatchInfo,  
    __in PMSIPATCHSEQUENCEINFO pPatchInfo  
);
```

## Parameters

*szProductPackagePath* [in]

Full path to an .msi file. The function determines the patches that are applicable to this package and in what sequence.

*cPatchInfo* [in]

Number of patches in the array. Must be greater than zero.

*pPatchInfo* [in]

Pointer to an array of **MSIPATCHSEQUENCEINFO** structures.

## Return Value

The **MsiDetermineApplicablePatches** function returns the following values.

Value	Meaning
ERROR_FUNCTION_FAILED	The function failed in a manner not covered in

	the other error codes.
ERROR_INVALID_PARAMETER	An argument is invalid.
ERROR_PATCH_NO_SEQUENCE	No valid sequence could be found for the set of patches.
ERROR_SUCCESS	The patches were successfully sorted.
ERROR_FILE_NOT_FOUND	The .msi file was not found.
ERROR_PATH_NOT_FOUND	The path to the .msi file was not found.
ERROR_INVALID_PATCH_XML	The XML patch data is invalid.
ERROR_INSTALL_PACKAGE_OPEN_FAILED	An installation package referenced by path cannot be opened.
ERROR_CALL_NOT_IMPLEMENTED	This error can be returned if the function was called from a <a href="#">custom action</a> or if MSXML 3.0 is not installed.

## Remarks

**Windows Installer 2.0:** Not supported. This function is available beginning with Windows Installer version 3.0.

If this function is called from a custom action it fails and returns

ERROR\_CALL\_NOT\_IMPLEMENTED. The function requires MSXML version 3.0 to process XML and returns ERROR\_CALL\_NOT\_IMPLEMENTED if MSXML 3.0 is not installed.

The **MsiDetermineApplicablePatches** function sets the **uStatus** and **dwOrder** members of each **MSIPATCHSEQUENCEINFO** structure pointed to by *pPatchInfo*. Each structure contains information about a particular patch.

If the function succeeds, the **MSIPATCHSEQUENCEINFO** structure of every patch that can be applied to the product returns with a **uStatus** of ERROR\_SUCCESS and a **dwOrder** greater than or equal to zero. The values of **dwOrder** greater than or equal to zero indicate the best application sequence for the patches starting with zero.

If the function succeeds, patches excluded from the best patching sequence return a **MSIPATCHSEQUENCEINFO** structure with a **dwOrder** equal to -1. In these cases, a **uStatus** field of ERROR\_SUCCESS indicates a patch that is obsolete or superseded for the product. A **uStatus** field of ERROR\_PATCH\_TARGET\_NOT\_FOUND indicates a patch that is inapplicable to the product.

If the function fails, the **MSIPATCHSEQUENCEINFO** structure for every patch returns a **dwOrder** equal to -1. In this case, the **uStatus** fields can contain errors with more information about individual patches. For example, ERROR\_PATCH\_NO\_SEQUENCE is returned for patches that have circular sequencing information.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h

<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiDetermineApplicablePatchesW</b> (Unicode) and <b>MsiDetermineApplicablePatchesA</b> (ANSI)

## See Also

[ProductCode](#)

[MsiDeterminePatchSequence](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiDeterminePatchSequence Function

The **MsiDeterminePatchSequence** function takes a set of patch files, XML files, and XML blobs and determines the best sequence of application for the patches to a specified installed product. This function accounts for patches that have already been applied to the product and accounts for obsolete and superseded patches.

## Syntax

```
C++UINT MsiDeterminePatchSequence(  
    __in      LPCTSTR szProductCode,  
    __in_opt  LPCTSTR szUserSid,  
    __in      MSIINSTALLCONTEXT dwContext,  
    __in      DWORD cPatchInfo,  
    __in      PMSIPATCHSEQUENCEINFO pPatchInfo  
);
```

## Parameters

*szProductCode* [in]

The product that is the target for the set of patches. The value must be the **ProductCode** GUID for the product.

*szUserSid* [in, optional]

Null-terminated string that specifies a security identifier (SID) of a user. This parameter restricts the context of enumeration for this user account. This parameter cannot be the special SID strings s-1-1-0 (everyone) or s-1-5-18 (local system). For the machine context *dwContext* is set to `MSIINSTALLCONTEXT_MACHINE` and *szUserSid* must be null. For the current user context *szUserSid* can be null and *dwContext* can be set to `MSIINSTALLCONTEXT_USERMANAGED` or `MSIINSTALLCONTEXT_USERUNMANAGED`.

*dwContext* [in]

Restricts the enumeration to a per-user-unmanaged, per-user-

managed, or per-machine context. This parameter can be any one of the following values.

Type of context	Meaning
MSIINSTALLCONTEXT_USERMANAGED	Patches are considered managed installations (for the user specified by <i>szUserSid</i> with this null <i>szUserSid</i> with this the current user.
MSIINSTALLCONTEXT_USERUNMANAGED	Patches are considered unmanaged installation specified by <i>szUserSid</i> <i>szUserSid</i> with this cor current user.
MSIINSTALLCONTEXT_MACHINE	Patches are considered machine installation. <i>W</i> is set to MSIINSTALLCONTE the <i>szUserSid</i> paramete

*cPatchInfo* [in]

The number of patches in the array.

*pPatchInfo* [in]

Pointer to an array of **MSIPATCHSEQUENCEINFO** structures.

## Return Value

The **MsiDeterminePatchSequence** function returns the following values.

Value	Meaning
ERROR_FUNCTION_FAILED	The function failed in a manner not covered in the other error codes.
ERROR_INVALID_PARAMETER	An argument is invalid.

ERROR_PATCH_NO_SEQUENCE	No valid sequence could be found for the set of patches.
ERROR_INSTALL_PACKAGE_OPEN_FAILED	An installation package referenced by path cannot be opened.
ERROR_SUCCESS	The patches were successfully sorted.
ERROR_FILE_NOT_FOUND	The .msi file was not found.
ERROR_PATH_NOT_FOUND	The path to the .msi file was not found.
ERROR_INVALID_PATCH_XML	The XML patch data is invalid.
ERROR_INSTALL_PACKAGE_INVALID	The installation package was invalid.
ERROR_ACCESS_DENIED	A user that is not an administrator attempted to call the function with a context of a different user.
ERROR_BAD_CONFIGURATION	The configuration data for a registered patch or product is invalid.
ERROR_UNKNOWN_PRODUCT	The <b>ProductCode</b> GUID specified is not registered.
ERROR_FUNCTION_NOT_CALLED	Windows Installer version 3.0 is required to

	determine the best patch sequence. The function was called with <i>szProductCode</i> not yet installed with Windows Installer version 3.0.
ERROR_CALL_NOT_IMPLEMENTED	This error can be returned if the function was called from a <a href="#">custom action</a> or if MSXML 3.0 is not installed.
ERROR_UNKNOWN_PATCH	The specified patch is unknown.

## Remarks

**Windows Installer 2.0:** Not supported. This function is available beginning with Windows Installer version 3.0.

Users that do not have administrator privileges can call this function only in their own or machine context. Users that are administrators can call it for other users.

If this function is called from a custom action it fails and returns ERROR\_CALL\_NOT\_IMPLEMENTED. The function requires MSXML version 3.0 to process XML and returns ERROR\_CALL\_NOT\_IMPLEMENTED if MSXML 3.0 is not installed.

The **MsiDeterminePatchSequence** function sets the **uStatus** and **dwOrder** members of each **MSIPATCHSEQUENCEINFO** structure pointed to by *pPatchInfo*. Each structure contains information about a particular patch.

If the function succeeds, the **MSIPATCHSEQUENCEINFO** structure of every patch that can be applied to the product returns with a **uStatus** of ERROR\_SUCCESS and a **dwOrder** greater than or equal to zero. The

values of **dwOrder** greater than or equal to zero indicate the best application sequence for the patches starting with zero.

If the function succeeds, patches excluded from the best patching sequence return a **MSIPATCHSEQUENCEINFO** structure with a **dwOrder** equal to -1. In these cases, a **uStatus** field of **ERROR\_SUCCESS** indicates a patch that is obsolete or superseded for the product. A **uStatus** field of **ERROR\_PATCH\_TARGET\_NOT\_FOUND** indicates a patch that is inapplicable to the product.

If the function fails, the **MSIPATCHSEQUENCEINFO** structure for every patch returns a **dwOrder** equal to -1. In this case, the **uStatus** fields can contain errors with more information about individual patches. For example, **ERROR\_PATCH\_NO\_SEQUENCE** is returned for patches that have circular sequencing information.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiDeterminePatchSequenceW</b> (Unicode) and <b>MsiDeterminePatchSequenceA</b> (ANSI)

## See Also

[ProductCode](#)  
[MsiDetermineApplicablePatches](#)

## **MSIPATCHSEQUENCEINFO**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiEnableLog Function

The **MsiEnableLog** function sets the log mode for all subsequent installations that are initiated in the calling process.

## Syntax

```
C++UINT MsiEnableLog(  
    __in  DWORD dwLogMode,  
    __in  LPCTSTR szLogFile,  
    __in  DWORD dwLogAttributes  
);
```

## Parameters

*dwLogMode* [in]

Specifies the log mode. This parameter can be one or more of the following values.

Value	Meaning
INSTALLLOGMODE_FATALEXIT	Logs out of memory or fatal information.
INSTALLLOGMODE_ERROR	Logs the error messages.
INSTALLLOGMODE_EXTRADEBUG	Sends extra debugging information as handle creation information file.  <b>Windows 2000 and Windows XP:</b> This feature is supported.
INSTALLLOGMODE_WARNING	Logs the warning messages.
INSTALLLOGMODE_USER	Logs the user requests.
INSTALLLOGMODE_INFO	Logs the status messages.

	displayed.
INSTALLLOGMODE_RESOLVESOURCE	Request to determine a va location.
INSTALLLOGMODE_OUTOFDISKSPACE	Indicates insufficient disk
INSTALLLOGMODE_ACTIONSTART	Logs the start of new inst
INSTALLLOGMODE_ACTIONDATA	Logs the data record with action.
INSTALLLOGMODE_COMMONDATA	Logs the parameters for u initialization.
INSTALLLOGMODE_PROPERTYDUMP	Logs the property values
INSTALLLOGMODE_VERBOSE	Logs the information in a modes, except for INSTALLLOGMODE_E This sends large amounts to a log file not generally May be used for technical
INSTALLLOGMODE_LOGONLYONERROR	Logging information is co less frequently saved to th can improve the performa installations, but may hav for large installations. The removed when the install If the installation fails, all information is saved to th  <b>Windows Installer 2.0</b> mode is not available.

*szLogFile* [in]

Specifies the string that holds the full path to the log file. Entering a

null disables logging, in which case *dwlogmode* is ignored. If a path is supplied, then *dwlogmode* must not be zero.

*dwLogAttributes* [in]

Specifies how frequently the log buffer is to be flushed.

Value	Meaning
INSTALLLOGATTRIBUTES_APPEND	If this value is set, the installer appends the existing log specified by <i>szLogFile</i> . If not set, any existing log specified by <i>szLogFile</i> is overwritten.
INSTALLLOGATTRIBUTES_FLUSHEACHLINE	Forces the log buffer to be flushed after each line. If this value is not set, the installer flushes the log buffer after 20 lines by calling <b>FlushFileBuffers</b> .

## Return Value

Value	Meaning
ERROR_INVALID_PARAMETER	An invalid log mode was specified.
ERROR_SUCCESS	The function succeeded.

## Remarks

For a description of the Logging policy, see [System Policy](#).

The path to the log file location must already exist when using this function. The Installer does not create the directory structure for the log file.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiEnableLogW</b> (Unicode) and <b>MsiEnableLogA</b> (ANSI)

## See Also

[Interface and Logging Functions](#)  
[Logging](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiEndTransaction Function

The **MsiEndTransaction** function can commit or roll back all the installations belonging to the transaction opened by the **MsiBeginTransaction** function. This function should be called by the current owner of the transaction.

**Windows Installer 4.0 and earlier:** Not supported. This function is available beginning with Windows Installer 4.5.

## Syntax

```
C++UINT WINAPI MsiEndTransaction(  
    __in DWORD dwTransactionState  
);
```

## Parameters

*dwTransactionState* [in]

The value of this parameter determines whether the installer commits or rolls back all the installations belonging to the transaction. The value can be one of the following.

Value	Meaning
MSITRANSACTIONSTATE_ROLLBACK	Performs a <a href="#">Rollback Installation</a> to undo changes to the system belonging to the transaction opened by the <b>MsiBeginTransaction</b> function.
MSITRANSACTIONSTATE_COMMIT	Commits all changes to the system belonging to the transaction. Runs any <a href="#">Commit Custom Actions</a> and commits to the system any changes to

	Win32 or common language runtime assemblies. Deletes the rollback script, and after using this option, the transaction's changes can no longer be undone with a Rollback Installation.
--	--

## Return Value

The **MsiEndTransaction** function returns the following values.

Value	Meaning
ERROR_ACCESS_DENIED	A transaction can be ended only by the current owner.
ERROR_INSTALL_FAILURE	An installation belonging to the transaction could not be completed.
ERROR_INSTALL_ALREADY_RUNNING	An installation belonging to the transaction is still in progress.
ERROR_ROLLBACK_DISABLED	An installation belonging to the transaction did not complete. During the installation, the <a href="#">DisableRollback</a> action disabled rollback installations of the package. The installer rolls back the installation up to the point where rollback was disabled, and the function returns this error.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.5 on Windows Vista, Windows XP, Windows Server 2003, and Windows Server 2008. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

### [Multiple Package Installations](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiEnumClients Function

The **MsiEnumClients** function enumerates the clients for a given installed component. The function retrieves one product code each time it is called.

## Syntax

```
C++UINT MsiEnumClients(  
    __in LPCTSTR szComponent,  
    __in DWORD iProductIndex,  
    __out LPTSTR lpProductBuf  
);
```

## Parameters

*szComponent* [in]

Specifies the component whose clients are to be enumerated.

*iProductIndex* [in]

Specifies the index of the client to retrieve. This parameter should be zero for the first call to the **MsiEnumClients** function and then incremented for subsequent calls. Because clients are not ordered, any new client has an arbitrary index. This means that the function can return clients in any order.

*lpProductBuf* [out]

Pointer to a buffer that receives the product code. This buffer must be 39 characters long. The first 38 characters are for the GUID, and the last character is for the terminating null character.

## Return Value

Value	Meaning
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.

ERROR_NO_MORE_ITEMS	There are no clients to return.
ERROR_NOT_ENOUGH_MEMORY	The system does not have enough memory to complete the operation. Available with Windows Server 2003.
ERROR_SUCCESS	A value was enumerated.
ERROR_UNKNOWN_COMPONENT	The specified component is unknown.

## Remarks

To enumerate clients, an application should initially call the **MsiEnumClients** function with the *iProductIndex* parameter set to zero. The application should then increment the *iProductIndex* parameter and call **MsiEnumClients** until there are no more clients (that is, until the function returns ERROR\_NO\_MORE\_ITEMS).

When making multiple calls to **MsiEnumClients** to enumerate all of the component's clients, each call should be made from the same thread.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib

**DLL**

Msi.dll

**Unicode and ANSI  
names**

**MsiEnumClientsW** (Unicode) and  
**MsiEnumClientsA** (ANSI)

## See Also

[System Status Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiEnumClientsEx Function

The **MsiEnumClientsEx** function enumerates the installed applications that use a specified component. The function retrieves a [product code](#) for an application each time it is called.

**Windows Installer 4.5 or earlier:** Not supported. This function is available beginning with Windows Installer 5.0.

## Syntax

```
C++UINT WINAPI MsiEnumClientsEx(  
    __in          LPCTSTR szComponent,  
    __in_opt     LPCTSTR szUserSid,  
    __in         DWORD dwContext,  
    __in         DWORD dwProductIndex,  
    __out_opt    TCHAR szProductBuf,  
    __out_opt    MSIINSTALLCONTEXT *pdwInstalledContext,  
    __out_opt    LPTSTR szSid,  
    __inout_opt LPDWORD pcchSid  
);
```

## Parameters

*szComponent* [in]

The component code GUID that identifies the component. The function enumerates the applications that use this component.

*szUserSid* [in, optional]

A null-terminated string value that contains a security identifier (SID.) The enumeration of applications extends to users identified by this SID. The special SID string s-1-1-0 (Everyone) enumerates all applications for all users in the system. A SID value other than s-1-1-0 specifies a user SID for a particular user and enumerates the instances of applications installed by the specified user.

SID type	Meaning
NULL	Specifies the currently logged-on user.

User SID	Specifies an enumeration for a particular user. An example of a user SID is "S-1-3-64-2415071341-1358098788-3127455600-2561".
s-1-1-0	Specifies an enumeration for all users in the system.

**Note** The special SID string s-1-5-18 (System) cannot be used to enumerate applications that exist in the per-machine installation context. Setting the SID value to s-1-5-18 returns `ERROR_INVALID_PARAMETER`. When *dwContext* is set to `MSIINSTALLCONTEXT_MACHINE` only, the value of *szUserSid* must be null.

*dwContext* [in]

A flag that extends the enumeration to instances of applications installed in the specified installation context. The enumeration includes only instances of applications that are installed by the users identified by *szUserSid*.

This can be a combination of the following values.

Context	Meaning
MSIINSTALLCONTEXT_USERMANAGED 1	Include applications in per-user-managed inst
MSIINSTALLCONTEXT_USERUNMANAGED 2	Include applications in per-user-unmanaged i context.
MSIINSTALLCONTEXT_MACHINE 4	Include applications in per-machine installatio When <i>dwInstallContex</i> MSIINSTALLCONTE only, the value of the <i>s</i> parameter must be null

*dwProductIndex* [in]

Specifies the index of the application to retrieve. The value of this parameter must be zero (0) in the first call to the function. For each subsequent call, the index must be incremented by 1. The index should only be incremented if the previous call to the function returns `ERROR_SUCCESS`.

*szProductBuf* [out, optional]

A string value that receives the product code for the application. The length of the buffer at this location should be large enough to hold a null-terminated string value containing the product code. The first 38 **TCHAR** characters receives the GUID for the component, and the 39th character receives a terminating NULL character.

*pdwInstalledContext* [out, optional]

A flag that gives the installation context of the application.

This can be a combination of the following values.

<b>Context</b>	<b>Meaning</b>
MSIINSTALLCONTEXT_USERMANAGED 1	The application is installed in the per-user-managed installation context.
MSIINSTALLCONTEXT_USERUNMANAGED 2	The application is installed in the per-user-unmanaged installation context.
MSIINSTALLCONTEXT_MACHINE 4	The application is in the per-machine installation installation context.

*szSid* [out, optional]

Receives the security identifier (SID) that identifies the user that installed the application. The location receives an empty string value if this instance of the application exists in a per-machine installation

context.

The length of the buffer should be large enough to hold a null-terminated string value containing the SID. If the buffer is too small, the function returns `ERROR_MORE_DATA` and the location pointed to by *pcchSid* receives the number of **TCHAR** in the SID, not including the terminating NULL character.

If *szSid* is set to NULL and *pcchSid* is a valid pointer to a location in memory, the function returns `ERROR_SUCCESS` and the location receives the number of **TCHAR** in the SID, not including the terminating NULL. The function can then be called again to retrieve the value, with the *szSid* buffer resized large enough to contain `*pcchSid + 1` characters.

SID type	Meaning
Empty string	The application is installed in a per-machine installation context.
User SID	The SID for the user that installed the product.

*pcchSid* [in, out, optional]

Pointer to a location in memory that contains a variable that specifies the number of **TCHAR** in the SID, not including the terminating NULL. When the function returns, this variable is set to the size of the requested SID whether or not the function can successfully copy the SID and terminating NULL into the buffer location pointed to by *szSid*. The size is returned as the number of TCHAR in the requested value, not including the terminating null character.

This parameter can be set to NULL only if *szSid* is also NULL, otherwise the function returns `ERROR_INVALID_PARAMETER`. If *szSid* and *pcchSid* are both set to NULL, the function returns `ERROR_SUCCESS` if the SID exists, without retrieving the SID value.

## Return Value

The **MsiEnumClientsEx** function returns one of the following values.

<b>Return code</b>	<b>Description</b>
ERROR_ACCESS_DENIED	Administrtrator privileges are required to enumerate components of applications installed by users other than the current user.
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INVALID_PARAMETER	An invalid parameter is passed to the function.
ERROR_NO_MORE_ITEMS	There are no more applications to enumerate.
ERROR_SUCCESS	The function succeeded.
ERROR_MORE_DATA	The provided buffer was too small to hold the entire value.
ERROR_FUNCTION_FAILED	The function failed.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiEnumClientsExW</b> (Unicode) and <b>MsiEnumClientsExA</b> (ANSI)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiEnumComponentQualifiers Function

The **MsiEnumComponentQualifiers** function enumerates the advertised qualifiers for the given component. This function retrieves one qualifier each time it is called.

## Syntax

```
C++UINT MsiEnumComponentQualifiers(  
    __in    LPTSTR  szComponent,  
    __in    DWORD   iIndex,  
    __out   LPTSTR  lpQualifierBuf,  
    __inout DWORD *pcchQualifierBuf,  
    __out   LPTSTR  lpApplicationDataBuf,  
    __inout DWORD *pcchApplicationDataBuf  
);
```

## Parameters

*szComponent* [in]

Specifies component whose qualifiers are to be enumerated.

*iIndex* [in]

Specifies the index of the qualifier to retrieve. This parameter should be zero for the first call to the **MsiEnumComponentQualifiers** function and then incremented for subsequent calls. Because qualifiers are not ordered, any new qualifier has an arbitrary index. This means that the function can return qualifiers in any order.

*lpQualifierBuf* [out]

Pointer to a buffer that receives the qualifier code.

*pcchQualifierBuf* [in, out]

Pointer to a variable that specifies the size, in characters, of the buffer pointed to by the *lpQualifierBuf* parameter. On input, this size should include the terminating null character. On return, the value does not include the null character.

*IpApplicationDataBuf* [out]

Pointer to a buffer that receives the application registered data for the qualifier. This parameter can be null.

*pcchApplicationDataBuf* [in, out]

Pointer to a variable that specifies the size, in characters, of the buffer pointed to by the *IpApplicationDataBuf* parameter. On input, this size should include the terminating null character. On return, the value does not include the null character. This parameter can be null only if the *IpApplicationDataBuf* parameter is null.

## Return Value

Value	Meaning
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_MORE_DATA	A buffer is too small to hold the requested data.
ERROR_NO_MORE_ITEMS	There are no qualifiers to return.
ERROR_NOT_ENOUGH_MEMORY	The system does not have enough memory to complete the operation. Available with Windows Server 2003.
ERROR_SUCCESS	A value was enumerated.
ERROR_UNKNOWN_COMPONENT	The specified component is unknown.

## Remarks

To enumerate qualifiers, an application should initially call the

**MsiEnumComponentQualifiers** function with the *iIndex* parameter set to zero. The application should then increment the *iIndex* parameter and call **MsiEnumComponentQualifiers** until there are no more qualifiers (that is, until the function returns `ERROR_NO_MORE_ITEMS`).

When **MsiEnumComponentQualifiers** returns, the *pcchQualifierBuf* parameter contains the length of the qualifier string stored in the buffer. The count returned does not include the terminating null character. If the buffer is not big enough, **MsiEnumComponentQualifiers** returns `ERROR_MORE_DATA`, and this parameter contains the size of the string, in characters, without counting the null character. The same mechanism applies to *pcchDescriptionBuf*.

When making multiple calls to **MsiEnumComponentQualifiers** to enumerate all of the component's advertised qualifiers, each call should be made from the same thread.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiEnumComponentQualifiersW</b> (Unicode) and <b>MsiEnumComponentQualifiersA</b> (ANSI)

## See Also

[System Status Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiEnumComponents Function

The [MsiEnumComponents](#) function enumerates the installed components for all products. This function retrieves one component code each time it is called.

## Syntax

```
C++UINT MsiEnumComponents(  
    __in    DWORD  iComponentIndex,  
    __out   LPTSTR lpComponentBuf  
);
```

## Parameters

*iComponentIndex* [in]

Specifies the index of the component to retrieve. This parameter should be zero for the first call to the [MsiEnumComponents](#) function and then incremented for subsequent calls. Because components are not ordered, any new component has an arbitrary index. This means that the function can return components in any order.

*lpComponentBuf* [out]

Pointer to a buffer that receives the component code. This buffer must be 39 characters long. The first 38 characters are for the [GUID](#), and the last character is for the terminating null character.

## Return Value

Value	Meaning
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_NO_MORE_ITEMS	There are no components to return.

ERROR_NOT_ENOUGH_MEMORY	The system does not have enough memory to complete the operation. Available with Windows Server 2003.
ERROR_SUCCESS	A value was enumerated.

## Remarks

To enumerate components, an application should initially call the **MsiEnumComponents** function with the *iComponentIndex* parameter set to zero. The application should then increment the *iComponentIndex* parameter and call **MsiEnumComponents** until there are no more components (that is, until the function returns ERROR\_NO\_MORE\_ITEMS).

When making multiple calls to **MsiEnumComponents** to enumerate all of the product's components, each call should be made from the same thread.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiEnumComponentsW</b> (Unicode) and <b>MsiEnumComponentsA</b> (ANSI)

---

## See Also

[System Status Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiEnumComponentsEx Function

The **MsiEnumComponentsEx** function enumerates installed components. The function retrieves the component code for one component each time it is called. The component code is the string GUID unique to the component, version, and language.

**Windows Installer 4.5 or earlier:** Not supported. This function is available beginning with Windows Installer 5.0.

## Syntax

```
C++UINT WINAPI MsiEnumComponentsEx(  
    __in_opt LPCTSTR szUserSid,  
    __in      DWORD dwContext,  
    __in      DWORD dwIndex,  
    __out_opt TCHAR szInstalledComponentCode[39],  
    __out_opt MSIINSTALLCONTEXT *pdwInstalledContext,  
    __out_opt LPTSTR szSid,  
    __inout  LPDWORD pcchSid  
);
```

## Parameters

*szUserSid* [in, optional]

A null-terminated string that contains a security identifier (SID.) The enumeration of installed components extends to users identified by this SID. The special SID string s-1-1-0 (Everyone) specifies an enumeration of all installed components across all products of all users in the system. A SID value other than s-1-1-0 specifies a user SID for a particular user and restricts the enumeration to instances of applications installed by the specified user.

SID type	Meaning
NULL	Specifies the currently logged-on user.
User SID	An enumeration for a specific user in the system. An example of an user SID is "S-

	1-3-64-2415071341-1358098788-3127455600-2561".
s-1-1-0	Specifies all users in the system.

### Note

The special SID string s-1-5-18 (System) cannot be used to enumerate applications installed in the per-machine installation context. Setting the SID value to s-1-5-18 returns `ERROR_INVALID_PARAMETER`. When *dwContext* is set to `MSIINSTALLCONTEXT_MACHINE` only, *szUserSid* must be null.

### *dwContext* [in]

A flag that restricts the enumeration of installed component to instances of products installed in the specified installation context. The enumeration includes only product instances installed by the users specified by *szUserSid*.

Flag	Meaning
MSIINSTALLCONTEXT_USERMANAGED 1	Include products that e user-managed installat
MSIINSTALLCONTEXT_USERUNMANAGED 2	Include products that e user-unmanaged instal
MSIINSTALLCONTEXT_MACHINE 4	Include products that e machine installation co <i>dwInstallContext</i> is set <code>MSIINSTALLCONTE</code> only, the <i>szUserSID</i> pa null.

### *dwIndex* [in]

Specifies the index of the component to retrieve. This parameter must be zero (0) for the first call to **MsiEnumComponentsEx** function. For each subsequent call, the index must be incremented

by 1. The index should only be incremented if the previous call to the function returns `ERROR_SUCCESS`. Components are not ordered and can be returned by the function in any order.

*szInstalledComponentCode* [out, optional]

An output buffer that receives the component code GUID for the installed component. The length of the buffer should be large enough to hold a null-terminated string value containing the component code. The first 38 **TCHAR** characters receives the GUID for the component, and the 39th character receives a terminating NULL character.

*pdwInstalledContext* [out, optional]

A flag that gives the installation context the application that installed the component.

Flag	Meaning
MSIINSTALLCONTEXT_USERMANAGED 1	The application is installed in the per-user-managed installation context.
MSIINSTALLCONTEXT_USERUNMANAGED 2	The application is installed in the per-user-unmanaged installation context.
MSIINSTALLCONTEXT_MACHINE 4	The application is installed in the per-machine installation context.

*szSid* [out, optional]

Receives the security identifier (SID) that identifies the user that installed the application that owns the component. The location receives an empty string if this instance of the application is installed in a per-machine installation context.

The length of the buffer at this location should be large enough to

hold a null-terminated string value containing the SID. If the buffer is too small, the function returns `ERROR_MORE_DATA` and the location pointed to by *pcchSid* receives the number of **TCHAR** in the SID, not including the terminating NULL character.

If *szSid* is set to NULL and *pcchSid* is a valid pointer to a location in memory, the function returns `ERROR_SUCCESS` and the location receives the number of **TCHAR** in the SID, not including the terminating NULL. The function can then be called again to retrieve the value, with the *szSid* buffer resized large enough to contain `*pcchSid + 1` characters.

SID type	Meaning
Empty string	The application is installed in a per-machine installation context.
User SID	The SID for the user in the system that installed the application.

*pcchSid* [in, out]

Receives the number of **TCHAR** in the SID, not including the terminating NULL. When the function returns, this variable is set to the size of the requested SID whether or not the function can successfully copy the SID and terminating NULL into the buffer location pointed to by *szSid*. The size is returned as the number of **TCHAR** in the requested value, not including the terminating null character.

This parameter can be set to NULL only if *szSid* is also NULL, otherwise the function returns `ERROR_INVALID_PARAMETER`. If *szSid* and *pcchSid* are both set to NULL, the function returns `ERROR_SUCCESS` if the SID exists, without retrieving the SID value.

## Return Value

The [MsiEnumProductsEx](#) function returns one of the following values.

Return code	Description
-------------	-------------

ERROR_ACCESS_DENIED	Administrator privileges are required to enumerate components of applications installed by users other than the current user.
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INVALID_PARAMETER	An invalid parameter is passed to the function.
ERROR_NO_MORE_ITEMS	There are no more components to enumerate.
ERROR_SUCCESS	The function succeeded.
ERROR_MORE_DATA	The provided buffer was too small to hold the entire value.
ERROR_FUNCTION_FAILED	The function failed.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiEnumComponentsExW</b> (Unicode) and <b>MsiEnumComponentsExA</b> (ANSI)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiEnumFeatures Function

The [MsiEnumFeatures](#) function enumerates the published features for a given product. This function retrieves one feature ID each time it is called.

## Syntax

```
C++UINT MsiEnumFeatures(  
    __in    LPCTSTR szProduct,  
    __in    DWORD  iFeatureIndex,  
    __out   LPTSTR lpFeatureBuf,  
    __out   LPTSTR lpParentBuf  
);
```

## Parameters

*szProduct* [in]

Null-terminated string specifying the product code of the product whose features are to be enumerated.

*iFeatureIndex* [in]

Specifies the index of the feature to retrieve. This parameter should be zero for the first call to the [MsiEnumFeatures](#) function and then incremented for subsequent calls. Because features are not ordered, any new feature has an arbitrary index. This means that the function can return features in any order.

*lpFeatureBuf* [out]

Pointer to a buffer that receives the feature ID. The size of the buffer must hold a string value of length MAX\_FEATURE\_CHARS+1. The function returns ERROR\_MORE\_DATA if the length of the feature ID exceeds MAX\_FEATURE\_CHARS.

*lpParentBuf* [out]

Pointer to a buffer that receives the feature ID of the parent of the feature. The size of the buffer must hold a string value of length MAX\_FEATURE\_CHARS+1. If the length of the feature ID of the parent feature exceeds MAX\_FEATURE\_CHARS, only the first MAX\_FEATURE\_CHARS characters get copied into the buffer.

## Return Value

Value	Meaning
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_MORE_DATA	A buffer is too small to hold the requested data.
ERROR_NO_MORE_ITEMS	There are no features to return.
ERROR_SUCCESS	A value was enumerated.
ERROR_UNKNOWN_PRODUCT	The specified product is unknown.

## Remarks

To enumerate features, an application should initially call the [MsiEnumFeatures](#) function with the *iFeatureIndex* parameter set to zero. The application should then increment the *iFeatureIndex* parameter and call [MsiEnumFeatures](#) until there are no more features (that is, until the function returns ERROR\_NO\_MORE\_ITEMS).

## Requirements

Version
Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.

<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiEnumFeaturesW</b> (Unicode) and <b>MsiEnumFeaturesA</b> (ANSI)

## See Also

[System Status Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiEnumPatches Function

The **MsiEnumPatches** function enumerates all of the patches that have been applied to a product. The function returns the patch code GUID for each patch that has been applied to the product and returns a list of transforms from each patch that apply to the product. Note that patches may have many transforms only some of which are applicable to a particular product. The list of transforms are returned in the same format as the value of the **TRANSFORMS** property.

**Note** *pcchTransformsBuf* is not set to the number of characters copied to *lpTransformsBuf* upon a successful return of **MsiEnumPatches**.

## Syntax

```
C++UINT MsiEnumPatches(  
    __in    LPCTSTR szProduct,  
    __in    DWORD  iPatchIndex,  
    __out   LPTSTR lpPatchBuf,  
    __out   LPTSTR lpTransformsBuf,  
    __inout DWORD *pcchTransformsBuf  
);
```

## Parameters

*szProduct* [in]

Specifies the product code of the product for which patches are to be enumerated.

*iPatchIndex* [in]

Specifies the index of the patch to retrieve. This parameter should be zero for the first call to the **MsiEnumPatches** function and then incremented for subsequent calls.

*lpPatchBuf* [out]

Pointer to a buffer that receives the patch's GUID. This argument is required.

*lpTransformsBuf* [out]

Pointer to a buffer that receives the list of transforms in the patch

that are applicable to the product. This argument is required and cannot be Null.

*pcchTransformsBuf* [in, out]

Set to the number of characters copied to *lpTransformsBuf* upon an unsuccessful return of the function. Not set for a successful return. On input, this is the full size of the buffer, including a space for a terminating null character. If the buffer passed in is too small, the count returned does not include the terminating null character.

## Return Value

Value	Meaning
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_NO_MORE_ITEMS	There are no patches to return.
ERROR_SUCCESS	A value was enumerated.
ERROR_MORE_DATA	A buffer is too small to hold the requested data.

## Remarks

To enumerate patches, an application should initially call the **MsiEnumPatches** function with the *iPatchIndex* parameter set to zero. The application should then increment the *iPatchIndex* parameter and call **MsiEnumPatches** until there are no more products (until the function returns ERROR\_NO\_MORE\_ITEMS).

If the buffer is too small to hold the requested data, **MsiEnumPatches** returns ERROR\_MORE\_DATA and *pcchTransformsBuf* contains the number of characters copied to *lpTransformsBuf*, without counting the Null character.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiEnumPatchesW</b> (Unicode) and <b>MsiEnumPatchesA</b> (ANSI)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiEnumPatchesEx Function

The **MsiEnumPatchesEx** function enumerates all patches in a specific context or across all contexts. Patches already applied to products are enumerated. Patches that have been registered but not yet applied to products are also enumerated.

**Windows Installer 2.0:** Not supported. This function is available beginning with Windows Installer version 3.0.

## Syntax

```
C++UINT MsiEnumPatchesEx(  
    __in_opt    LPCTSTR szProductCode,  
    __in_opt    LPCTSTR szUserSid,  
    __in        DWORD dwContext,  
    __in        DWORD dwFilter,  
    __in        DWORD dwIndex,  
    __out_opt   TCHAR szPatchCode[39],  
    __out_opt   TCHAR szTargetProductCode[39],  
    __out_opt   MSIINSTALLCONTEXT *pdwTargetProductContext,  
    __out_opt   LPTSTR szTargetUserSid,  
    __inout_opt LPDWORD pcchTargetUserSid  
);
```

## Parameters

*szProductCode* [in, optional]

A null-terminated string that specifies the **ProductCode** GUID of the product whose patches are enumerated. If non-null, patch enumeration is restricted to instances of this product under the user and context specified by *szUserSid* and *dwContext*. If null, the patches for all products under the specified context are enumerated.

*szUserSid* [in, optional]

A null-terminated string that specifies a security identifier (SID) that restricts the context of enumeration. The special SID string s-1-1-0 (Everyone) specifies enumeration across all users in the system. A SID value other than s-1-1-0 is considered a user SID and restricts

enumeration to that user. When enumerating for a user other than current user, any patches that were applied in a per-user-unmanaged context using a version less than Windows Installer version 3.0, are not enumerated. This parameter can be set to null to specify the current user.

SID type	Meaning
NULL	Specifies the currently logged-on user.
User SID	An enumeration for a specific user in the system. An example of user SID is "S-1-3-64-2415071341-1358098788-3127455600-2561".
s-1-1-0	An enumeration across all users in the system.

**Note** The special SID string s-1-5-18 (System) cannot be used to enumerate products or patches installed as per-machine. Setting the SID value to s-1-5-18 returns ERROR\_INVALID\_PARAMETER. When *dwContext* is set to MSIINSTALLCONTEXT\_MACHINE only, *szUserSid* must be null.

*dwContext* [in]

Restricts the enumeration to one or a combination of contexts. This parameter can be any one or a combination of the following values.

Context	Meaning
MSIINSTALLCONTEXT_USERMANAGED	The enumeration that includes per-user-managed installed users that <i>szUserSid</i> specifies an invalid SID returns no
MSIINSTALLCONTEXT_USERUNMANAGED	In this context, only patches with Windows Installer version less than 3.0 are enumerated for users other than the current user. For the current user, the function enumerates all

	new patches. An invalid <i>szUserSid</i> returns no it
MSIINSTALLCONTEXT_MACHINE	An enumeration that is per-machine installatio <i>dwContext</i> is set to MSIINSTALLCONTE only, the <i>szUserSid</i> par null.

*dwFilter* [in]

The filter for enumeration. This parameter can be one or a combination of the following parameters.

Filter	Meaning
MSIPATCHSTATE_APPLIED 1	The enumeration includes patches that have been applied. Enumeration does not include superseded or obsolete patches.
MSIPATCHSTATE_SUPERSEDED 2	The enumeration includes patches that are marked as superseded.
MSIPATCHSTATE_OBSOLETED 4	The enumeration includes patches that are marked as obsolete.
MSIPATCHSTATE_REGISTERED 8	The enumeration includes patches that are registered but not yet applied. The <a href="#">MsiSourceListAddSourceEx</a> function can register new patches. <b>Note</b> Patches registered for users other than current user and applied in the per-user-unmanaged context are not enumerated.

MSIPATCHSTATE\_ALL  
15

The enumeration includes all applied, obsolete, superseded, and registered patches.

*dwIndex* [in]

The index of the patch to retrieve. This parameter must be zero for the first call to the **MsiEnumPatchesEx** function and then incremented for subsequent calls. The *dwIndex* parameter should be incremented only if the previous call returned `ERROR_SUCCESS`.

*szPatchCode* [out, optional]

An output buffer to contain the GUID of the patch being enumerated. The buffer should be large enough to hold the GUID. This parameter can be null.

*szTargetProductCode* [out, optional]

An output buffer to contain the **ProductCode** GUID of the product that receives this patch. The buffer should be large enough to hold the GUID. This parameter can be null.

*pdwTargetProductContext* [out, optional]

Returns the context of the patch being enumerated. The output value can be `MSIINSTALLCONTEXT_USERMANAGED`, `MSIINSTALLCONTEXT_USERUNMANAGED`, or `MSIINSTALLCONTEXT_MACHINE`. This parameter can be null.

*szTargetUserSid* [out, optional]

An output buffer that receives the string SID of the account under which this patch instance exists. This buffer returns an empty string for a per-machine context.

This buffer should be large enough to contain the SID. If the buffer is too small, the function returns `ERROR_MORE_DATA` and sets *\*pcchTargetUserSid* to the number of **TCHAR** in the value, not including the terminating NULL character.

If the *szTargetUserSid* is set to NULL and *pcchTargetUserSid* is set to a valid pointer, the function returns `ERROR_SUCCESS` and sets *\*pcchTargetUserSid* to the number of **TCHAR** in the value, not including the terminating NULL character. The function can then be called again to retrieve the value, with *szTargetUserSid* buffer large

enough to contain *\*pcchTargetUserSid* + 1 characters.

If *szTargetUserSid* and *pcchTargetUserSid* are both set to NULL, the function returns `ERROR_SUCCESS` if the value exists, without retrieving the value.

*pcchTargetUserSid* [in, out, optional]

A pointer to a variable that specifies the number of **TCHAR** in the *szTargetUserSid* buffer. When the function returns, this parameter is set to the size of the requested value whether or not the function copies the value into the specified buffer. The size is returned as the number of **TCHAR** in the requested value, not including the terminating null character.

This parameter can be set to NULL only if *szTargetUserSid* is also NULL, otherwise the function returns `ERROR_INVALID_PARAMETER`.

## Return Value

The `MsiEnumPatchesEx` function returns one of the following values.

Return code	Description
<code>ERROR_ACCESS_DENIED</code>	The function fails trying to access a resource with insufficient privileges.
<code>ERROR_BAD_CONFIGURATION</code>	The configuration data is corrupt.
<code>ERROR_INVALID_PARAMETER</code>	An invalid parameter is passed to the function.
<code>ERROR_NO_MORE_ITEMS</code>	There are no more patches to enumerate.
<code>ERROR_SUCCESS</code>	The patch is successfully enumerated.
<code>ERROR_UNKNOWN_PRODUCT</code>	The product that <i>szProduct</i> specifies is not installed on the computer in the specified contexts.
<code>ERROR_MORE_DATA</code>	This is returned when

*pcchTargetUserSid* points to a buffer size less than required to copy the SID. In this case, the user can fix the buffer and call **MsiEnumPatchesEx** again for the same index value.

## Remarks

Non-administrators can enumerate patches within their visibility only. Administrators can enumerate patches for other user contexts.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiEnumPatchesExW</b> (Unicode) and <b>MsiEnumPatchesExA</b> (ANSI)

## See Also

[MsiSourceListAddSourceEx](#)  
[ProductCode](#)  
[Installation Context](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiEnumProducts Function

The **MsiEnumProducts** function enumerates through all the products currently advertised or installed. Products that are installed in both the per-user and per-machine **installation context** and advertisements are enumerated.

## Syntax

```
C++UINT MsiEnumProducts(  
    __in    DWORD iProductIndex,  
    __out   LPTSTR lpProductBuf  
);
```

## Parameters

*iProductIndex* [in]

Specifies the index of the product to retrieve. This parameter should be zero for the first call to the **MsiEnumProducts** function and then incremented for subsequent calls. Because products are not ordered, any new product has an arbitrary index. This means that the function can return products in any order.

*lpProductBuf* [out]

Pointer to a buffer that receives the product code. This buffer must be 39 characters long. The first 38 characters are for the GUID, and the last character is for the terminating null character.

## Return Value

Value	Meaning
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_NO_MORE_ITEMS	There are no products to return.

ERROR_NOT_ENOUGH_MEMORY	The system does not have enough memory to complete the operation. Available with Windows Server 2003.
ERROR_SUCCESS	A value was enumerated.

## Remarks

To enumerate products, an application should initially call the [MsiEnumProducts](#) function with the *iProductIndex* parameter set to zero. The application should then increment the *iProductIndex* parameter and call [MsiEnumProducts](#) until there are no more products (until the function returns ERROR\_NO\_MORE\_ITEMS).

When making multiple calls to [MsiEnumProducts](#) to enumerate all of the products, each call should be made from the same thread.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiEnumProductsW</b> (Unicode) and <b>MsiEnumProductsA</b> (ANSI)

## See Also

System Status Functions  
Determining Installation Context

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiEnumProductsEx Function

The **MsiEnumProductsEx** function enumerates through one or all the instances of products that are currently advertised or installed in the specified contexts. This function supersedes [MsiEnumProducts](#).

## Syntax

```
C++UINT MsiEnumProductsEx(  
    __in_opt LPCTSTR szProductCode,  
    __in LPCTSTR szUserSid,  
    __in DWORD dwContext,  
    __in DWORD dwIndex,  
    __out_opt TCHAR szInstalledProductCode[39],  
    __out_opt MSIINSTALLCONTEXT *pdwInstalledContext,  
    __out_opt LPTSTR szSid,  
    __inout_opt LPDWORD pcchSid  
);
```

## Parameters

*szProductCode* [in, optional]

**ProductCode** GUID of the product to be enumerated. Only instances of products within the scope of the context specified by the *szUserSid* and *dwContext* parameters are enumerated. This parameter can be set to NULL to enumerate all products in the specified context.

*szUserSid* [in]

Null-terminated string that specifies a security identifier (SID) that restricts the context of enumeration. The special SID string s-1-1-0 (Everyone) specifies enumeration across all users in the system. A SID value other than s-1-1-0 is considered a user-SID and restricts enumeration to the current user or any user in the system. This parameter can be set to null to restrict the enumeration scope to the current user.

SID type	Meaning
NULL	Specifies the currently logged-on user.

User SID	Specifies enumeration for a particular user in the system. An example of user SID is "S-1-3-64-2415071341-1358098788-3127455600-2561".
s-1-1-0	Specifies enumeration across all users in the system.

**Note** The special SID string s-1-5-18 (System) cannot be used to enumerate products or patches installed as per-machine. When *dwContext* is set to MSIINSTALLCONTEXT\_MACHINE only, *szUserSid* must be null.

*dwContext* [in]

Restricts the enumeration to a context. This parameter can be any one or a combination of the values shown in the following table.

Context	Meaning
MSIINSTALLCONTEXT_USERMANAGED	Enumeration extended managed installations specified by <i>szUserSid</i> . SID returns no items.
MSIINSTALLCONTEXT_USERUNMANAGED	Enumeration extended unmanaged installations specified by <i>szUserSid</i> . SID returns no items.
MSIINSTALLCONTEXT_MACHINE	Enumeration extended machine installations. If <i>dwInstallContext</i> is set to MSIINSTALLCONTEXT_MACHINE only, the <i>szUserSID</i> parameter must be null.

*dwIndex* [in]

Specifies the index of the product to retrieve. This parameter must be zero for the first call to the **MsiEnumProductsEx** function and then incremented for subsequent calls. The index should be incremented, only if the previous call has returned **ERROR\_SUCCESS**. Because products are not ordered, any new product has an arbitrary index. This means that the function can return products in any order.

*szInstalledProductCode* [out, optional]

Null-terminated string of **TCHAR** that gives the **ProductCode** GUID of the product instance being enumerated. This parameter can be null.

*pdwInstalledContext* [out, optional]

Returns the context of the product instance being enumerated. The output value can be **MSIINSTALLCONTEXT\_USERMANAGED**, **MSIINSTALLCONTEXT\_USERUNMANAGED**, or **MSIINSTALLCONTEXT\_MACHINE**. This parameter can be null.

*szSid* [out, optional]

An output buffer that receives the string SID of the account under which this product instance exists. This buffer returns an empty string for an instance installed in a per-machine context.

This buffer should be large enough to contain the SID. If the buffer is too small, the function returns **ERROR\_MORE\_DATA** and sets *\*pcchSid* to the number of **TCHAR** in the SID, not including the terminating NULL character.

If *szSid* is set to NULL and *pcchSid* is set to a valid pointer, the function returns **ERROR\_SUCCESS** and sets *\*pcchSid* to the number of **TCHAR** in the value, not including the terminating NULL. The function can then be called again to retrieve the value, with the *szSid* buffer large enough to contain *\*pcchSid* + 1 characters.

If *szSid* and *pcchSid* are both set to NULL, the function returns **ERROR\_SUCCESS** if the value exists, without retrieving the value.

*pcchSid* [in, out, optional]

When calling the function, this parameter should be a pointer to a variable that specifies the number of **TCHAR** in the *szSid* buffer. When the function returns, this parameter is set to the size of the requested value whether or not the function copies the value into the

specified buffer. The size is returned as the number of **TCHAR** in the requested value, not including the terminating null character.

This parameter can be set to NULL only if *szSid* is also NULL, otherwise the function returns **ERROR\_INVALID\_PARAMETER**.

## Return Value

The **MsiEnumProductsEx** function returns one of the following values.

Return code	Description
ERROR_ACCESS_DENIED	If the scope includes users other than the current user, you need administrator privileges.
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_NO_MORE_ITEMS	There are no more products to enumerate.
ERROR_SUCCESS	A product is enumerated.
ERROR_MORE_DATA	The <i>szSid</i> parameter is too small to get the user SID.
ERROR_UNKNOWN_PRODUCT	The product is not installed on the computer in the specified context.
ERROR_FUNCTION_FAILED	An unexpected internal failure.

## Remarks

To enumerate products, an application must initially call the **MsiEnumProductsEx** function with the *iIndex* parameter set to zero. The

application must then increment the *iProductIndex* parameter and call **MsiEnumProductsEx** until it returns `ERROR_NO_MORE_ITEMS` and there are no more products to enumerate.

When making multiple calls to **MsiEnumProductsEx** to enumerate all of the products, each call must be made from the same thread.

A user must have administrator privileges to enumerate products across all user accounts or a user account other than the current user account. The enumeration skips products that are advertised only (such as products not installed) in the per-user-unmanaged context when enumerating across all users or a user other than the current user.

Use [MsiGetProductInfoEx](#) to get the state or other information about each product instance enumerated by **MsiEnumProductsEx**.

**Windows Installer 2.0:** Not supported. This function is available beginning with Windows Installer version 3.0.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiEnumProductsExW</b> (Unicode) and <b>MsiEnumProductsExA</b> (ANSI)

## See Also

**Removing Patches**  
**MsiEnumProducts**  
**ProductCode**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiEnumRelatedProducts Function

The **MsiEnumRelatedProducts** function enumerates products with a specified upgrade code. This function lists the currently installed and advertised products that have the specified **UpgradeCode** property in their **Property table**.

## Syntax

```
C++UINT MsiEnumRelatedProducts(  
    __in    LPCTSTR lpUpgradeCode,  
    __in    DWORD dwReserved,  
    __in    DWORD iProductIndex,  
    __out   LPTSTR lpProductBuf  
);
```

## Parameters

*lpUpgradeCode* [in]

The null-terminated string specifying the upgrade code of related products that the installer is to enumerate.

*dwReserved* [in]

This parameter is reserved and must be 0.

*iProductIndex* [in]

The zero-based index into the registered products.

*lpProductBuf* [out]

A buffer to receive the product code GUID. This buffer must be 39 characters long. The first 38 characters are for the **GUID**, and the last character is for the terminating null character.

## Return Value

Value	Meaning
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to

	the function.
ERROR_NO_MORE_ITEMS	There are no products to return.
ERROR_NOT_ENOUGH_MEMORY	The system does not have enough memory to complete the operation. Available starting with Windows Server 2003.
ERROR_SUCCESS	A value was enumerated.

## Remarks

See [UpgradeCode](#) property.

To enumerate currently installed and advertised products that have a specific upgrade code, an application should initially call the [MsiEnumRelatedProducts](#) function with the *iProductIndex* parameter set to zero. The application should then increment the *iProductIndex* parameter and call [MsiEnumRelatedProducts](#) until the function returns ERROR\_NO\_MORE\_ITEMS, which means there are no more products with the specified upgrade code.

When making multiple calls to [MsiEnumRelatedProducts](#) to enumerate all of the related products, each call should be made from the same thread.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiEnumRelatedProductsW</b> (Unicode) and <b>MsiEnumRelatedProductsA</b> (ANSI)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiExtractPatchXMLData Function

The **MsiExtractPatchXMLData** function extracts information from a patch that can be used to determine if the patch applies to a target system. The function returns an XML string that can be provided to **MsiDeterminePatchSequence** and **MsiDetermineApplicablePatches** instead of the full patch file. The returned information can be used to determine whether the patch is applicable.

## Syntax

```
C++UINT MsiExtractPatchXMLData(  
    __in          LPCTSTR szPatchPath,  
    __in          DWORD dwReserved,  
    __out_opt     LPTSTR szXMLData,  
    __inout_opt   DWORD *pcchXMLData  
);
```

## Parameters

*szPatchPath* [in]

The full path to the patch that is being queried. Pass in as a null-terminated string. This parameter cannot be null.

*dwReserved* [in]

A reserved argument that must be 0 (zero).

*szXMLData* [out, optional]

A pointer to a buffer to hold the XML string that contains the extracted patch information. This buffer should be large enough to contain the received information. If the buffer is too small, the function returns **ERROR\_MORE\_DATA** and sets *\*pcchXMLData* to the number of **TCHAR** in the value, not including the terminating NULL character.

If *szXMLData* is set to NULL and *pcchXMLData* is set to a valid pointer, the function returns **ERROR\_SUCCESS** and sets *\*pcchXMLData* to the number of **TCHAR** in the value, not including the terminating NULL character. The function can then be called

again to retrieve the value, with *szXMLData* buffer large enough to contain *\*pcchXMLData* + 1 characters.

*pcchXMLData* [in, out, optional]

A pointer to a variable that specifies the number of **TCHAR** in the *szXMLData* buffer. When the function returns, this parameter is set to the size of the requested value whether or not the function copies the value into the specified buffer. The size is returned as the number of **TCHAR** in the requested value, not including the terminating null character.

If this parameter is set to NULL, the function returns **ERROR\_INVALID\_PARAMETER**.

## Return Value

The **MsiExtractPatchXMLData** function can return the following values.

Return code	Description
<b>ERROR_FUNCTION_FAILED</b>	The function failed in a way that is not identified by any of the return values in this table.
<b>ERROR_INVALID_PARAMETER</b>	An invalid parameter was passed to the function.
<b>ERROR_MORE_DATA</b>	The value does not fit in the provided buffer.
<b>ERROR_PATCH_OPEN_FAILED</b>	The patch file could not be opened.
<b>ERROR_SUCCESS</b>	The function was successful.
<b>ERROR_PATCH_PACKAGE_INVALID</b>	The patch file could not be opened.
<b>ERROR_CALL_NOT_IMPLEMENTED</b>	This error can be returned if MSXML 3.0 is not installed.

## Remarks

**Windows Installer 2.0:** Not supported. This function is available beginning with Windows Installer version 3.0.

The [ExtractPatchXMLData](#) method of the [Installer](#) object uses the [MsiExtractPatchXMLData](#) function.

## Requirements

<b>Version</b>	Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiExtractPatchXMLDataW</b> (Unicode) and <b>MsiExtractPatchXMLDataA</b> (ANSI)

## See Also

[MsiDeterminePatchSequence](#)  
[MsiDetermineApplicablePatches](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetComponentPath Function

The **MsiGetComponentPath** function returns the full path to an installed component. If the key path for the component is a registry key then the registry key is returned.

## Syntax

```
C++INSTALLSTATE MsiGetComponentPath(  
    __in    LPCTSTR szProduct,  
    __in    LPCTSTR szComponent,  
    __out   LPTSTR lpPathBuf,  
    __inout DWORD *pcchBuf  
);
```

## Parameters

*szProduct* [in]

Specifies the product code for the client product.

*szComponent* [in]

Specifies the component ID of the component to be located.

*lpPathBuf* [out]

Pointer to a variable that receives the path to the component. This parameter can be null. If the component is a registry key, the registry roots are represented numerically. If this is a registry subkey path, there is a backslash at the end of the Key Path. If this is a registry value key path, there is no backslash at the end. For example, a registry path on a 32-bit operating system of **HKEY\_CURRENT\_USER\SOFTWARE\Microsoft** is returned as "01:\SOFTWARE\Microsoft". The registry roots returned on 32-bit operating systems are defined as shown in the following table.

**Note** On 64-bit operating systems, a value of 20 is added to the numerical registry roots in this table to distinguish them from registry key paths on 32-bit operating systems. For example, a registry key path of **HKEY\_CURRENT\_USER\SOFTWARE\Microsoft** is returned as "21:\SOFTWARE\Microsoft", if the component path is a

registry key on a 64-bit operating system.

Root	Meaning
HKEY_CLASSES_ROOT	00
HKEY_CURRENT_USER	01
HKEY_LOCAL_MACHINE	02
HKEY_USERS	03

*pcchBuf* [in, out]

Pointer to a variable that specifies the size, in characters, of the buffer pointed to by the *lpPathBuf* parameter. On input, this is the full size of the buffer, including a space for a terminating null character. If the buffer passed in is too small, the count returned does not include the terminating null character.

If *lpPathBuf* is null, *pcchBuf* can be null.

## Return Value

The **MsiGetComponentPath** function returns the following values.

Value	Meaning
INSTALLSTATE_NOTUSED	The component being requested is disabled on the computer.
INSTALLSTATE_ABSENT	The component is not installed.
INSTALLSTATE_INVALIDARG	One of the function parameters is invalid.
INSTALLSTATE_LOCAL	The component is installed locally.
INSTALLSTATE_SOURCE	The component is installed to run from source.

INSTALLSTATE_SOURCEABSENT	The component source is inaccessible.
INSTALLSTATE_UNKNOWN	The product code or component ID is unknown.

## Remarks

Upon success of the **MsiGetComponentPath** function, the *pcchBuf* parameter contains the length of the string in *lpPathBuf*.

The **MsiGetComponentPath** function might return `INSTALLSTATE_ABSENT` or `INSTALL_STATE_UNKNOWN`, for the following reasons:

- `INSTALLSTATE_ABSENT`  
The application did not properly ensure that the feature was installed by calling **MsiUseFeature** and, if necessary, **MsiConfigureFeature**.
- `INSTALLSTATE_UNKNOWN`  
The feature is not published. The application should have determined this earlier by calling **MsiQueryFeatureState** or **MsiEnumFeatures**. The application makes these calls while it initializes. An application should only use features that are known to be published. Since `INSTALLSTATE_UNKNOWN` should have been returned by **MsiUseFeature** as well, either **MsiUseFeature** was not called, or its return value was not properly checked.

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See

<b>Version</b>	the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiGetComponentPathW</b> (Unicode) and <b>MsiGetComponentPathA</b> (ANSI)

## See Also

[Component-Specific Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetComponentPathEx Function

The **MsiGetComponentPathEx** function returns the full path to an installed component. If the key path for the component is a registry key then the function returns the registry key.

This function extends the existing **MsiGetComponentPath** function to enable searches for components across user accounts and installation contexts.

## Syntax

```
C++INSTALLSTATE MsiGetComponentPathEx(  
    __in          LPCTSTR szProductCode,  
    __in          LPCTSTR szComponentCode,  
    __in_opt     LPCTSTR szUserSid,  
    __in_opt     MSIINSTALLCONTEXT dwContext,  
    __out_opt    LPTSTR  szPathBuf,  
    __inout_opt  LPDWORD pcchBuf  
);
```

## Parameters

*szProductCode* [in]

A null-terminated string value that specifies an application's product code GUID. The function gets the path of installed components used by this application.

*szComponentCode* [in]

A null-terminated string value that specifies a component code GUID. The function gets the path of an installed component having this component code.

*szUserSid* [in, optional]

A null-terminated string value that specifies the security identifier (SID) for a user in the system. The function gets the paths of installed components of applications installed under the user accounts identified by this SID. The special SID string s-1-1-0 (Everyone) specifies all users in the system. If this parameter is NULL, the function gets the path of an installed component for the

currently logged-on user only.

SID type	Meaning
NULL	Specifies the currently logged-on user.
User SID	Specifies a particular user in the system. An example of an user SID is "S-1-3-64-2415071341-1358098788-3127455600-2561".
s-1-1-0	Specifies all users in the system.

**Note** The special SID string s-1-5-18 (System) cannot be used to search applications installed in the per-machine installation context. Setting the SID value to s-1-5-18 returns ERROR\_INVALID\_PARAMETER. When *dwContext* is set to MSIINSTALLCONTEXT\_MACHINE only, *szUserSid* must be null.

*dwContext* [in, optional]

A flag that specifies the installation context. The function gets the paths of installed components of applications installed in the specified installation context. This parameter can be a combination of the following values.

Context	Meaning
MSIINSTALLCONTEXT_USERMANAGED 1	Include applications in per-user-managed inst
MSIINSTALLCONTEXT_USERUNMANAGED 2	Include applications in per-user-unmanaged i context.
MSIINSTALLCONTEXT_MACHINE 4	Include applications in per-machine installatio When <i>dwInstallContex</i> MSIINSTALLCONTE only, the <i>szUserSID</i> pa null.

*szPathBuf* [out, optional]

A string value that receives the path to the component. This parameter can be null. If the component is a registry key, the registry roots are represented numerically. If this is a registry subkey path, there is a backslash at the end of the Key Path. If this is a registry value key path, there is no backslash at the end. For example, a registry path on a 32-bit operating system of

**HKEY\_CURRENT\_USER\SOFTWARE\Microsoft** is returned as "01:\SOFTWARE\Microsoft". The registry roots returned on 32-bit operating systems are defined as shown in the following table.

**Note** On 64-bit operating systems, a value of 20 is added to the numerical registry roots in this table to distinguish them from registry key paths on 32-bit operating systems. For example, a registry key path of **HKEY\_CURRENT\_USER\SOFTWARE\Microsoft** is returned as "21:\SOFTWARE\Microsoft", if the component path is a registry key on a 64-bit operating system.

<b>Root</b>	<b>Meaning</b>
HKEY_CLASSES_ROOT	00
HKEY_CURRENT_USER	01
HKEY_LOCAL_MACHINE	02
HKEY_USERS	03

*pcchBuf* [in, out, optional]

Pointer to a location that receives the size of the buffer, in **TCHAR**, pointed to by the *szPathBuf* parameter. The value in this location should be set to the count of **TCHAR** in the string including the terminating null character. If the size of the buffer is too small, this parameter receives the length of the string value without including the terminating null in the count.

## **Return Value**

The **MsiGetComponentPathEx** function returns the following values.

Value	Meaning
INSTALLSTATE_NOTUSED	The component being requested is disabled on the computer.
INSTALLSTATE_BADCONFIG	Configuration data is corrupt.
INSTALLSTATE_ABSENT	The component is not installed.
INSTALLSTATE_INVALIDARG	One of the function parameters is invalid.
INSTALLSTATE_LOCAL	The component is installed locally.
INSTALLSTATE_SOURCE	The component is installed to run from source.
INSTALLSTATE_SOURCEABSENT	The component source is inaccessible.
INSTALLSTATE_UNKNOWN	The product code or component ID is unknown.
INSTALLSTATE_BROKEN	The component is corrupt or partially missing in some way and requires repair.

## Remarks

The **MsiGetComponentPathEx** function might return **INSTALLSTATE\_ABSENT** or **INSTALL\_STATE\_UNKNOWN**, for the following reasons:

- **INSTALLSTATE\_ABSENT**  
The application did not properly ensure that the feature was installed by calling **MsiUseFeature** and, if necessary, **MsiConfigureFeature**.

- **INSTALLSTATE\_UNKNOWN**

The feature is not published. The application should have determined this earlier by calling [MsiQueryFeatureState](#) or [MsiEnumFeatures](#). The application makes these calls while it initializes. An application should only use features that are known to be published. Since **INSTALLSTATE\_UNKNOWN** should have been returned by **MsiUseFeature** as well, either **MsiUseFeature** was not called, or its return value was not properly checked.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiGetComponentPathExW</b> (Unicode) and <b>MsiGetComponentPathExA</b> (ANSI)

## See Also

### [Component-Specific Functions](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetFeatureInfo Function

The **MsiGetFeatureInfo** function returns descriptive information for a feature.

## Syntax

```
C++UINT MsiGetFeatureInfo(  
    __in          MSIHANDLE hProduct,  
    __in          LPCTSTR szFeature,  
    __out_opt     LPDWORD lpAttributes,  
    __out_opt     LPTSTR lpTitleBuf,  
    __inout_opt  LPDWORD pcchTitleBuf,  
    __out_opt     LPTSTR lpHelpBuf,  
    __inout_opt  LPDWORD pcchHelpBuf  
);
```

## Parameters

*hProduct* [in]

Handle to the product that owns the feature. This handle is obtained from **MsiOpenProduct**.

*szFeature* [in]

Feature code for the feature about which information should be returned.

*lpAttributes* [out, optional]

Pointer to a location containing one or more of the following Attribute flags.

Flag	Value
INSTALLFEATUREATTRIBUTE_FAVORLOCAL	1
INSTALLFEATUREATTRIBUTE_FAVORSOURCE	2
INSTALLFEATUREATTRIBUTE_FOLLOWPARENT	4
INSTALLFEATUREATTRIBUTE_FAVORADVERTISE	8

INSTALLFEATUREATTRIBUTE_DISALLOWADVERTISE	16
INSTALLFEATUREATTRIBUTE_NOUNSUPPORTEDADVERTISE	32

For more information, see [Feature Table](#). The values that **MsiGetFeatureInfo** returns are double the values in the Attributes column of the Feature Table.

*lpTitleBuf* [out, optional]

Pointer to a buffer to receive the localized name of the feature, which corresponds to the Title field in the Feature Table. This parameter is optional and can be null.

*pcchTitleBuf* [in, out, optional]

As input, the size of *lpTitleBuf*. As output, the number of characters returned in *lpTitleBuf*. On input, this is the full size of the buffer, and includes a space for a terminating null character. If the buffer that is passed in is too small, the count returned does not include the terminating null character.

*lpHelpBuf* [out, optional]

Pointer to a buffer to receive the localized description of the feature, which corresponds to the Description field for the feature in the Feature table. This parameter is optional and can be null.

*pcchHelpBuf* [in, out, optional]

As input, the size of *lpHelpBuf*. As output, the number of characters returned in *lpHelpBuf*. On input, this is the full size of the buffer, and includes a space for a terminating null character. If the buffer passed in is too small, the count returned does not include the terminating null character.

## Return Value

Return code	Description
ERROR_INVALID_HANDLE	The product handle is invalid.
ERROR_INVALID_PARAMETER	One of the parameters is invalid.

ERROR_MORE_DATA	A buffer is too small to hold the requested data.
ERROR_SUCCESS	The function returns successfully.
ERROR_UNKNOWN_FEATURE	The feature is not known.

## Remarks

The buffer sizes for the **MsiGetFeatureInfo** function should include an extra character for the terminating null character. If a buffer is too small, the returned string is truncated with null, and the buffer size contains the number of characters in the whole string, not including the terminating null character. For more information, see [Calling Database Functions From Programs](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiGetFeatureInfoW</b> (Unicode) and <b>MsiGetFeatureInfoA</b> (ANSI)

## See Also

## Product Query Functions

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetFeatureUsage Function

The **MsiGetFeatureUsage** function returns the usage metrics for a product feature.

## Syntax

```
C++UINT MsiGetFeatureUsage(  
    __in    LPCTSTR szProduct,  
    __in    LPCTSTR szFeature,  
    __out   DWORD  *pdwUseCount,  
    __out   WORD   *pwDateUsed  
);
```

## Parameters

*szProduct* [in]

Specifies the product code for the product that contains the feature.

*szFeature* [in]

Specifies the feature code for the feature for which metrics are to be returned.

*pdwUseCount* [out]

Indicates the number of times the feature has been used.

*pwDateUsed* [out]

Specifies the date that the feature was last used. The date is in the MS-DOS date format, as shown in the following table.

Bits	Meaning
0 – 4	Day of the month (1-31)
5 – 8	Month (1 = January, 2 = February, and so on)
9 – 15	Year offset from 1980 (add 1980 to get actual year)

## Return Value

The [MsiGetFeatureUsage](#) function returns the following values.

Value	Meaning
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INSTALL_FAILURE	No usage information is available or the product or feature is invalid.
ERROR_SUCCESS	The function completed successfully.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiGetFeatureUsageW</b> (Unicode) and <b>MsiGetFeatureUsageA</b> (ANSI)

## See Also

[System Status Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetFileHash Function

The [MsiGetFileHash](#) function takes the path to a file and returns a 128-bit hash of that file. Authoring tools may use [MsiGetFileHash](#) to obtain the file hash needed to populate the [MsiFileHash table](#).

Windows Installer uses file hashing as a means to detect and eliminate unnecessary file copying. A file hash stored in the MsiFileHash table may be compared to a hash of an existing file on the user's computer.

## Syntax

```
C++UINT MsiGetFileHash(  
    __in    LPCTSTR szFilePath,  
    __in    DWORD dwOptions,  
    __out   PMSIFILEHASHINFO pHash  
);
```

## Parameters

*szFilePath* [in]

Path to file that is to be hashed.

*dwOptions* [in]

The value in this column must be 0. This parameter is reserved for future use.

*pHash* [out]

Pointer to the returned file hash information.

## Return Value

Value	Meaning
ERROR_SUCCESS	The function completed successfully.
ERROR_FILE_NOT_FOUND	The file does not exist.
ERROR_ACCESS_DENIED	The file could not be opened to get version information.

E_FAIL	Unexpected error has occurred.
--------	--------------------------------

## Remarks

The entire 128-bit file hash is returned as four 32-bit fields. The numbering of the four fields is zero-based. The values returned by [MsiGetFileHash](#) correspond to the four fields of the [MSIFILEHASHINFO](#) structure. The first field corresponds to the HashPart1 column of the MsiFileHash table, the second field corresponds to the HashPart2 column, the third field corresponds to the HashPart3 column, and the fourth field corresponds to the HashPart4 column.

The hash information entered into the MsiFileHash table must be obtained by calling [MsiGetFileHash](#) or the [FileHash](#) method. Do not attempt to use other methods to generate the file hash.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiGetFileHashW</b> (Unicode) and <b>MsiGetFileHashA</b> (ANSI)

## See Also

MsiFileHash table  
**MSIFILEHASHINFO**  
Default File Versioning

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetFileSignatureInformation Function

The [MsiGetFileSignatureInformation](#) function takes the path to a file that has been digitally signed and returns the file's signer certificate and hash. [MsiGetFileSignatureInformation](#) may be called to obtain the signer certificate and hash needed to populate the [MsiDigitalCertificate](#), [MsiPatchCertificate](#), and [MsiDigitalSignature](#) tables.

**Windows Installer 3.0 and later:** Beginning with Windows Installer 3.0, the Windows Installer can verify the digital signatures of patches (.msp files) by using the [MsiPatchCertificate](#) and [MsiDigitalCertificate](#) tables. For more information see [Guidelines for Authoring Secure Installations](#) and [User Account Control \(UAC\) Patching](#).

**Windows Installer 2.0:** Digital signatures of patches is not supported. Windows Installer 2.0 uses digital signatures as a means to detect corrupted resources, and can only verify the digital signatures of external cabinets, and only by the use of the [MsiDigitalSignature](#) and [MsiDigitalCertificate](#) tables.

## Syntax

```
C++HRESULT MsiGetFileSignatureInformation(  
    __in    LPCTSTR szSignedObjectPath,  
    __in    DWORD dwFlags,  
    __out   PCCERT_CONTEXT *ppcCertContext,  
    __out   BYTE *pbHashData,  
    __inout DWORD *pcbHashData  
);
```

## Parameters

*szSignedObjectPath* [in]

Pointer to a null-terminated string specifying the full path to the file that contains the digital signature.

*dwFlags* [in]

Special error case flags.

Flag	Meaning
MSI_INVALID_HASH_IS_FATAL 0x1	Without this flag set, and when requesting only the certificate context, an invalid hash in the digital signature does not cause <b>MsiGetFileSignatureInformation</b> to return a fatal error.  To return a fatal error for an invalid hash, set the MSI_INVALID_HASH_IS_FATAL flag.

*ppcCertContext* [out]

Returned signer certificate context

*pbHashData* [out]

Returned hash buffer. This parameter can be null if the hash data is not being requested.

*pcbHashData* [in, out]

Pointer to a variable that specifies the size, in bytes, of the buffer pointed to by the *pbHashData* parameter. This parameter cannot be null if *pbHashData* is non-Null. If ERROR\_MORE\_DATA is returned, *pbHashData* gives the size of the buffer required to hold the hash data. If ERROR\_SUCCESS is returned, it gives the number of bytes written to the hash buffer. The *pcbHashData* parameter is ignored if *pbHashData* is null.

## Return Value

Value	Meaning
ERROR_SUCCESS/S_OK	Successful completion.
ERROR_INVALID_PARAMETER	Invalid parameter was specified.

ERROR_FUNCTION_FAILED	<b>WinVerifyTrust</b> is not available on the system. <b>MsiGetFileSignatureInformation</b> requires the presence of the Wintrust.dll file on the system.
ERROR_MORE_DATA	A buffer is too small to hold the requested data. If ERROR_MORE_DATA is returned, <i>pcbHashData</i> gives the size of the buffer required to hold the hash data.
TRUST_E_NOSIGNATURE	File is not signed
TRUST_E_BAD_DIGEST	The file's current hash is invalid according to the hash stored in the file's digital signature.
CERT_E_REVOKED	The file's signer certificate has been revoked. The file's digital signature is compromised.
TRUST_E_SUBJECT_NOT_TRUSTED	The subject failed the specified verification action. Most trust providers return a more detailed error code that describes the reason for the failure.
TRUST_E_PROVIDER_UNKNOWN	The trust provider is not recognized on this system.
TRUST_E_ACTION_UNKNOWN	The trust provider does not support the specified action.
TRUST_E_SUBJECT_FORM_UNKNOWN	The trust provider does not support the form specified for the subject

**MsiGetFileSignatureInformation** also returns all the Win32 error values mapped to their equivalent **HRESULT** data type by **HRESULT\_FROM\_WIN32**.

## Remarks

When requesting only the certificate context, an invalid hash in the digital signature does not cause **MsiGetFileSignatureInformation** to return a fatal error. To return a fatal error for an invalid hash, set the **MSI\_INVALID\_HASH\_IS\_FATAL** flag in the *dwFlags* parameter.

The certificate context and hash information is extracted from the file by a call to **WinVerifyTrust**. The *ppcCertContext* parameter is a duplicate of the signer certificate context from the signature. It is the responsibility of the caller to call *CertFreeCertificateContext* to free the certificate context when finished.

Note that **MsiGetFileSignatureInformation** requires the presence of the *Wintrust.dll* file on the system.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiGetFileSignatureInformationW</b> (Unicode) and <b>MsiGetFileSignatureInformationA</b> (ANSI)

## See Also

[MsiDigitalCertificate table](#)  
[MsiDigitalSignature table](#)  
[Digital Signatures and Windows Installer](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetFileVersion Function

The **MsiGetFileVersion** returns the version string and language string in the format that the installer expects to find them in the database. If you want only version information, set *lpLangBuf* and *pcchLangBuf* to 0 (zero). If you just want language information, set *lpVersionBuf* and *pcchVersionBuf* to 0 (zero).

## Syntax

```
C++UINT MsiGetFileVersion(  
    __in    LPCTSTR szFilePath,  
    __out   LPTSTR lpVersionBuf,  
    __inout DWORD *pcchVersionBuf,  
    __out   LPTSTR lpLangBuf,  
    __inout DWORD *pcchLangBuf  
);
```

## Parameters

*szFilePath* [in]

Specifies the path to the file.

*lpVersionBuf* [out]

Returns the file version.

Set to 0 for language information only.

*pcchVersionBuf* [in, out]

In and out buffer count as the number of **TCHAR**.

Set to 0 (zero) for language information only. On input, this is the full size of the buffer, including a space for a terminating null character. If the buffer passed in is too small, the count returned does not include the terminating null character.

*lpLangBuf* [out]

Returns the file language.

Set to 0 (zero) for version information only.

*pcchLangBuf* [in, out]

In and out buffer count as the number of **TCHAR**.

Set to 0 (zero) for version information only. On input, this is the full size of the buffer, including a space for a terminating null character. If the buffer passed in is too small, the count returned does not include the terminating null character.

## Return Value

Value	Meaning
ERROR_SUCCESS	Successful completion.
ERROR_FILE_NOT_FOUND	File does not exist.
ERROR_ACCESS_DENIED	File cannot be opened to get version information.
ERROR_FILE_INVALID	File does not contain version information.
ERROR_INVALID_DATA	The version information is invalid.
E_FAIL	Unexpected error.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib

**DLL**

Msi.dll

**Unicode and ANSI  
names**

**MsiGetFileVersionW** (Unicode) and  
**MsiGetFileVersionA** (ANSI)

## See Also

[System Status Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetPatchFileList Function

The **MsiGetPatchFileList** function is provided a list of .msp files, delimited by semicolons, and retrieves the list of files that can be updated by the patches.

## Syntax

```
C++UINT WINAPI MsiGetPatchFileList(  
    __in    LPCTSTR szProductCode,  
    __in    LPCTSTR szPatchList,  
    __inout LPDWORD pcFiles,  
    __inout MSIHANDLE **    pphFileRecords  
);
```

## Parameters

*szProductCode* [in]

A null-terminated string value containing the **ProductCode** (GUID) of the product which is the target of the patches. This parameter cannot be NULL.

*szPatchList* [in]

A null-terminated string value that contains the list of Windows Installer patches (.msp files). Each patch can be specified by the full path to the patch package. The patches in the list are delimited by semicolons. At least one patch must be specified.

*pcFiles* [in, out]

A pointer to a location that receives the number of files that will be updated on this system by this list of patches specified by *szPatchList*. This parameter is required.

*pphFileRecords* [in, out]

A pointer to a location that receives a pointer to an array of records. The first field (0-index) of each record contains the full file path of a file that can be updated when the list of patches in *szPatchList* are applied on this computer. This parameter is required.

## Return Value

The **MsiGetPatchFileList** function returns the following values.

Value	Meaning
ERROR_SUCCESS	The function completed successfully.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_FUNCTION_FAILED	The function failed.

## Remarks

For example, *szPatchList* could have the value:

```
"c:\sus\download\cache\Office\sp1.msp;  
c:\sus\download\cache\Office\QFE1.msp;  
c:\sus\download\cache\Office\QFEn.msp".
```

This function runs in the context of the caller. The product code is searched in the order of user-unmanaged context, user-managed context, and machine context.

You must close all **MSIHANDLE** objects that are returned by this function by calling the **MsiCloseHandle** function.

If the function fails, you can obtain extended error information by using the **MsiGetLastErrorRecord** function.

For more information about using the **MsiGetPatchFileList** function see [Listing the Files that can be Updated](#).

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 4.5 on Windows

<b>Version</b>	Server 2003 and Windows XP. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiGetPatchFileListW</b> (Unicode) and <b>MsiGetPatchFileListA</b> (ANSI)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetPatchInfo Function

The **MsiGetPatchInfo** function returns information about a patch.

## Syntax

```
C++UINT MsiGetPatchInfo(  
    __in    LPCTSTR szPatch,  
    __in    LPCTSTR szAttribute,  
    __out   LPTSTR lpValueBuf,  
    __inout DWORD *pcchValueBuf  
);
```

## Parameters

*szPatch* [in]

Specifies the patch code for the patch package.

*szAttribute* [in]

Specifies the attribute to be retrieved.

Attribute	Meaning
INSTALLPROPERTY_LOCALPACKAGE	Local cached package.

*lpValueBuf* [out]

Pointer to a buffer that receives the property value. This parameter can be null.

*pcchValueBuf* [in, out]

Pointer to a variable that specifies the size, in characters, of the buffer pointed to by the *lpValueBuf* parameter. On input, this is the full size of the buffer, including a space for a terminating null character. If the buffer passed in is too small, the count returned does not include the terminating null character.

If *lpValueBuf* is null, *pcchValueBuf* can be null.

## Return Value

The **MsiGetPatchInfo** function returns the following values.

Value	Meaning
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_MORE_DATA	A buffer is too small to hold the requested data.
ERROR_SUCCESS	The function completed successfully.
ERROR_UNKNOWN_PRODUCT	The patch package is not installed.
ERROR_UNKNOWN_PROPERTY	The property is unrecognized.

## Remarks

When the **MsiGetPatchInfo** function returns, the *pcchValueBuf* parameter contains the length of the class string stored in the buffer. The count returned does not include the terminating null character.

If the buffer is too small to hold the requested data, **MsiGetPatchInfo** returns **ERROR\_MORE\_DATA**, and *pcchValueBuf* contains the number of characters copied to *lpValueBuf*, without counting the null character.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the [Windows Installer Run-Time Requirements](#) for information about the minimum Windows service pack that is required by a Windows Installer version.

<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiGetPatchInfoW</b> (Unicode) and <b>MsiGetPatchInfoA</b> (ANSI)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetPatchInfoEx Function

The **MsiGetPatchInfoEx** function queries for information about the application of a patch to a specified instance of a product.

## Syntax

```
C++UINT MsiGetPatchInfoEx(  
    __in      LPCTSTR szPatchCode,  
    __in      LPCTSTR szProductCode,  
    __in      LPCTSTR szUserSid,  
    __in      MSIINSTALLCONTEXT dwContext,  
    __in      LPCTSTR szProperty,  
    __out_opt LPTSTR lpValue,  
    __inout   DWORD *pcchValue  
);
```

## Parameters

*szPatchCode* [in]

A null-terminated string that contains the GUID of the patch. This parameter cannot be null.

*szProductCode* [in]

A null-terminated string that contains the **ProductCode** GUID of the product instance. This parameter cannot be null.

*szUserSid* [in]

A null-terminated string that specifies the security identifier (SID) under which the instance of the patch being queried exists. Using a null value specifies the current user.

SID	Meaning
NULL	Specifies the user that is logged on.
User SID	Specifies the enumeration for a specific user ID in the system. The following example identifies a possible user SID: "S-1-3-64-2415071341-1358098788-

3127455600-2561".
-------------------

**Note** The special SID string s-1-5-18 (system) cannot be used to enumerate products installed as per-machine. If *dwContext* is `MSIINSTALLCONTEXT_MACHINE`, *szUserSid* must be null.

*dwContext* [in]

Restricts the enumeration to a per-user-unmanaged, per-user-managed, or per-machine context. This parameter can be any one of the following values.

Context	Meaning
<code>MSIINSTALLCONTEXT_USERMANAGED</code> 1	Query that is extended to all per-user-managed installations for the users that <i>szUserSid</i> specifies.
<code>MSIINSTALLCONTEXT_USERUNMANAGED</code> 2	Query that is extended to all per-user-unmanaged installations for the users that <i>szUserSid</i> specifies.
<code>MSIINSTALLCONTEXT_MACHINE</code> 4	Query that is extended to all per-machine installations.

*szProperty* [in]

A null-terminated string that specifies the property value to retrieve. The *szProperty* parameter can be one of the following:

Name	Meaning

INSTALLPROPERTY_LOCALPACKAGE "LocalPackage"	Gets the cached patch file that the product uses.
INSTALLPROPERTY_TRANSFORMS "Transforms"	Gets the set of patch transforms that the last patch installation applied to the product. This value may not be available for per-user, non-managed applications if the user is not logged on.
INSTALLPROPERTY_INSTALLDATE "InstallDate"	Get the date and time when the patch is applied to the product.
INSTALLPROPERTY_UNINSTALLABLE "Uninstallable"	Returns "1" if the patch is marked as possible to uninstall from the product. In this case, the installer can still block the uninstallation if this patch is required by another patch that cannot be uninstalled.
INSTALLPROPERTY_PATCHSTATE "State"	Returns "1" if this patch is currently applied to the product. Returns "2" if this patch is superseded by another patch. Returns "4" if this patch is obsolete. These values correspond to the constants the <i>dwFilter</i> parameter of <a href="#">MsiEnumPatchesEx</a> uses.

INSTALLPROPERTY_DISPLAYNAME "DisplayName"	Get the registered display name for the patch. For patches that do not include the DisplayName property in the <a href="#">MsiPatchMetadata</a> table, the returned display name is an empty string ("").
INSTALLPROPERTY_MOREINFOURL "MoreInfoURL"	Get the registered support information URL for the patch. For patches that do not include the MoreInfoURL property in the MsiPatchMetadata table, the returned support information URL is an empty string ("").

*lpValue* [out, optional]

This parameter is a pointer to a buffer that receives the property value. This buffer should be large enough to contain the information. If the buffer is too small, the function returns `ERROR_MORE_DATA` and sets *pcchValue* to the number of **TCHAR** in the property value, not including the terminating NULL character.

If *lpValue* is set to NULL and *pcchValue* is set to a valid pointer, the function returns `ERROR_SUCCESS` and sets *pcchValue* to the number of **TCHAR** in the value, not including the terminating NULL character. The function can then be called again to retrieve the value, with *lpValue* buffer large enough to contain *pcchValue* + 1 characters.

If *lpValue* and *pcchValue* are both set to NULL, the function returns `ERROR_SUCCESS` if the value exists, without retrieving the value.

*pcchValue* [in, out]

When calling the function, this parameter should be a pointer to a variable that specifies the number of **TCHAR** in the *lpValue* buffer.

When the function returns, this parameter is set to the size of the requested value whether or not the function copies the value into the specified buffer. The size is returned as the number of **TCHAR** in the requested value, not including the terminating null character.

This parameter can be set to NULL only if *lpValue* is also NULL. Otherwise, the function returns `ERROR_INVALID_PARAMETER`.

## Return Value

The `MsiGetPatchInfoEx` function returns the following values.

Return code	Description
<code>ERROR_ACCESS_DENIED</code>	The function fails trying to access a resource with insufficient privileges.
<code>ERROR_BAD_CONFIGURATION</code>	The configuration data is corrupt.
<code>ERROR_FUNCTION_FAILED</code>	The function fails and the error is not identified in other error codes.
<code>ERROR_INVALID_PARAMETER</code>	An invalid parameter is passed to the function.
<code>ERROR_MORE_DATA</code>	The value does not fit in the provided buffer.
<code>ERROR_SUCCESS</code>	The patch is enumerated successfully.
<code>ERROR_UNKNOWN_PRODUCT</code>	The product that <i>szProduct</i> specifies is not installed on the computer.
<code>ERROR_UNKNOWN_PROPERTY</code>	The property is unrecognized.
<code>ERROR_UNKNOWN_PATCH</code>	The patch is unrecognized.

## Remarks

**Windows Installer 2.0:** Not supported. This function is available beginning with Windows Installer version 3.0.

A user may query patch data for any product instance that is visible. The administrator group can query patch data for any product instance and any user on the computer. Not all values are guaranteed to be available for per-user, non-managed applications if the user is not logged on.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiGetPatchInfoExW</b> (Unicode) and <b>MsiGetPatchInfoExA</b> (ANSI)

## See Also

### [Removing Patches](#) [ProductCode](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetProductCode Function

The **MsiGetProductCode** function returns the product code of an application by using the component code of an installed or advertised component of the application. During initialization, an application must determine under which product code it has been installed or advertised.

## Syntax

```
C++UINT MsiGetProductCode(  
    __in LPCTSTR szComponent,  
    __out LPTSTR lpProductBuf  
);
```

## Parameters

*szComponent* [in]

This parameter specifies the component code of a component that has been installed by the application. This will be typically the component code of the component containing the executable file of the application.

*lpProductBuf* [out]

Pointer to a buffer that receives the product code. This buffer must be 39 characters long. The first 38 characters are for the GUID, and the last character is for the terminating null character.

## Return Value

Value	Meaning
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INSTALL_FAILURE	The product code could not be determined.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.

ERROR_SUCCESS	The function completed successfully.
ERROR_UNKNOWN_COMPONENT	The specified component is unknown.

## Remarks

During initialization, an application must determine the product code under which it was installed. An application can be part of different products in different installations. For example, an application can be part of a suite of applications, or it can be installed by itself.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiGetProductCodeW</b> (Unicode) and <b>MsiGetProductCodeA</b> (ANSI)

## See Also

[Application-Only Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetProductInfo Function

The **MsiGetProductInfo** function returns product information for published and installed products.

## Syntax

```
C++UINT MsiGetProductInfo(  
    __in      LPCTSTR szProduct,  
    __in      LPCTSTR szProperty,  
    __out     LPTSTR lpValueBuf,  
    __inout   DWORD *pcchValueBuf  
);
```

## Parameters

*szProduct* [in]

Specifies the product code for the product.

*szProperty* [in]

Specifies the property to be retrieved.

The [Required Properties](#) are guaranteed to be available, but other properties are available only if that property is set. For more information, see [Properties](#). The properties in the following list can be retrieved only from applications that are installed.

Property	Meaning
INSTALLPROPERTY_HELPLINK	Support link. For information, see <a href="#">ARPHELPLINK</a> .
INSTALLPROPERTY_HELPTELEPHONE	Support telephor information, see <a href="#">ARPHELPTEL</a> property.
INSTALLPROPERTY_INSTALLDATE	Installation date.

INSTALLPROPERTY_INSTALLEDLANGUAGE	Installed language  <b>Windows Installer and earlier:</b> not supported.
INSTALLPROPERTY_INSTALLEDPRODUCTNAME	Installed product name. For more information, see <a href="#">ProductName</a> property.
INSTALLPROPERTY_INSTALLLOCATION	Installation location. For more information, see <a href="#">ARPINSTALLLOCATION</a> property.
INSTALLPROPERTY_INSTALLSOURCE	Installation source information, see <a href="#">SourceDir</a> property.
INSTALLPROPERTY_LOCALPACKAGE	Local cached package.
INSTALLPROPERTY_PUBLISHER	Publisher. For more information, see <a href="#">Manufacturer</a> property.
INSTALLPROPERTY_URLINFOABOUT	URL information about the product. For more information, see <a href="#">ARPURLINFOABOUT</a> property.
INSTALLPROPERTY_URLUPDATEINFO	URL update information. For more information, see <a href="#">ARPURLUPDATEINFO</a> property.
INSTALLPROPERTY_VERSIONMINOR	Minor product version derived from the <a href="#">ProductVersion</a> property.

INSTALLPROPERTY_VERSIONMAJOR	Major product version derived from the <a href="#">ProductVersion</a>
INSTALLPROPERTY_VERSIONSTRING	Product version information, see <a href="#">ProductVersion</a>

To retrieve the product ID, registered owner, or registered company from applications that are installed, set *szProperty* to one of the following text string values.

Value	Description
ProductID	The product identifier for the product. For more information, see the <a href="#">ProductID</a> property.
RegCompany	The company registered to use this product.
RegOwner	The owner registered to use this product.

To retrieve the instance type of the product, set *szProperty* to the following value. This property is available for advertised or installed products.

Value	Description
InstanceType	A missing value or a value of 0 (zero) indicates a normal product installation. A value of 1 (one) indicates a product installed using a multiple instance transform and the MSNEWINSTANCE property. Available with the installer running Windows Server 2003 or Windows XP with SP1. For more information see, <a href="#">Installing Multiple Instances of Products and Patches</a> .

The advertised properties in the following list can be retrieved from applications that are advertised or installed.

Property	Description
----------	-------------

INSTALLPROPERTY_TRANSFORMS	Transforms.
INSTALLPROPERTY_LANGUAGE	Product language.
INSTALLPROPERTY_PRODUCTNAME	Human readable product name. For more information, see the <a href="#">ProductName</a> property.
INSTALLPROPERTY_ASSIGNMENTTYPE	Equals 0 (zero) if the product is advertised installed per-user. Equals 1 (one) if the product is advertised installed per-machine for all users.
INSTALLPROPERTY_PACKAGECODE	Identifier of the package this product was installed from. For more information, see <a href="#">Package Codes</a> .
INSTALLPROPERTY_VERSION	Product version derived from the <a href="#">ProductVersion</a> property.
INSTALLPROPERTY_PRODUCTICON	Primary icon for the package. For more information, see the <a href="#">ARPPRODUCTICON</a> property.
INSTALLPROPERTY_PACKAGENAME	Name of the original installation package.
INSTALLPROPERTY_AUTHORIZED_LUA_APP	A value of one (1) indicates a product that can be serviced by non-administrators using

User Account Control (UAC) Patching. A missing value or a value of 0 (zero) indicates that least-privilege patching is not enabled. Available in Windows Installer 3.0 or later.

*lpValueBuf* [out]

Pointer to a buffer that receives the property value. This parameter can be null.

*pcchValueBuf* [in, out]

Pointer to a variable that specifies the size, in characters, of the buffer pointed to by the *lpValueBuf* parameter. On input, this is the full size of the buffer, including a space for a terminating null character. If the buffer passed in is too small, the count returned does not include the terminating null character.

If *lpValueBuf* is null, *pcchValueBuf* can be null. In this case, the function checks that the property is registered correctly with the product.

## Return Value

Value	Meaning
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_MORE_DATA	A buffer is too small to hold the requested data.
ERROR_SUCCESS	The function completed successfully.

ERROR_UNKNOWN_PRODUCT	The product is unadvertised or uninstalled.
ERROR_UNKNOWN_PROPERTY	The property is unrecognized.  <b>Note</b> The <b>MsiGetProductInfo</b> function returns ERROR_UNKNOWN_PROPERTY if the application being queried is advertised and not installed.

## Remarks

When the **MsiGetProductInfo** function returns, the *pcchValueBuf* parameter contains the length of the string stored in the buffer. The count returned does not include the terminating null character. If the buffer is not large enough, **MsiGetProductInfo** returns ERROR\_MORE\_DATA and *pcchValueBuf* contains the size of the string, in characters, without counting the null character.

**MsiGetProductInfo**(INSTALLPROPERTY\_LOCALPACKAGE) does not necessarily return a path to the cached package. The cached package is for internal use only. Maintenance mode installations should be invoked through the [MsiConfigureFeature](#), [MsiConfigureProduct](#), or [MsiConfigureProductEx](#) functions.

If you attempt to use **MsiGetProductInfo** to query an advertised product for a property that is only available to installed products, the function returns ERROR\_UNKNOWN\_PROPERTY. For example, if the application is advertised and not installed, a query for the INSTALLPROPERTY\_INSTALLLOCATION property returns an error of ERROR\_UNKNOWN\_PROPERTY.

## Requirements

	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or
--	--

<b>Version</b>	Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiGetProductInfoW</b> (Unicode) and <b>MsiGetProductInfoA</b> (ANSI)

## See Also

[Determining Installation Context](#)  
[System Status Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetProductInfoEx Function

The **MsiGetProductInfoEx** function returns product information for advertised and installed products. This function can retrieve information about an instance of a product that is installed under a user account other than the current user.

The calling process must have administrative privileges for a user who is different from the current user. The **MsiGetProductInfoEx** function cannot query an instance of a product that is advertised under a per-user-unmanaged context for a user account other than the current user.

This function is an extension of the [MsiGetProductInfo](#) function.

## Syntax

```
C++UINT MsiGetProductInfoEx(  
    __in          LPCTSTR szProductCode,  
    __in          LPCTSTR szUserSid,  
    __in          MSIINSTALLCONTEXT dwContext,  
    __in          LPCTSTR szProperty,  
    __out_opt     LPTSTR lpValue,  
    __inout_opt  LPDWORD pcchValue  
);
```

## Parameters

*szProductCode* [in]

The **ProductCode** GUID of the product instance that is being queried.

*szUserSid* [in]

The security identifier (SID) of the account under which the instance of the product that is being queried exists. A null specifies the current user SID.

SID	Meaning
NULL	The currently logged-on user.
User SID	The enumeration for a specific user in the

system. An example of user SID is "S-1-3-64-2415071341-1358098788-3127455600-2561".

**Note** The special SID string s-1-5-18 (system) cannot be used to enumerate products installed as per-machine. If *dwContext* is MSIINSTALLCONTEXT\_MACHINE, *szUserSid* must be null.

*dwContext* [in]

The installation context of the product instance that is being queried.

Name	Meaning
MSIINSTALLCONTEXT_USERMANAGED	Retrieves the product property for the per-user-managed instance of the product.
MSIINSTALLCONTEXT_USERUNMANAGED	Retrieves the product property for the per-user-unmanaged instance of the product.
MSIINSTALLCONTEXT_MACHINE	Retrieves the product property for the per-machine instance of the product.

*szProperty* [in]

Property being queried.

The property to be retrieved. The properties in the following table can only be retrieved from applications that are already installed. All required properties are guaranteed to be available, but other properties are available only if the property is set. For more

information, see [Required Properties](#) and [Properties](#).

Property	Meaning
INSTALLPROPERTY_PRODUCTSTATE	The state of the product returned in string form for advertised and installed.
INSTALLPROPERTY_HELPLINK	The support link information, see <a href="#">ARPHELPLINK</a> .
INSTALLPROPERTY_HELPTELEPHONE	The support telephone number for more information, see <a href="#">ARPHELPTELEPHONE</a> property.
INSTALLPROPERTY_INSTALLDATE	Installation date.
INSTALLPROPERTY_INSTALLEDLANGUAGE	Installed language.  <b>Windows Installer 5.0 and earlier:</b> Not supported.
INSTALLPROPERTY_INSTALLEDPRODUCTNAME	The installed product name. For more information, see the <a href="#">ProductName</a> property.
INSTALLPROPERTY_INSTALLLOCATION	The installation location. For more information, see the <a href="#">ARPINSTALLLOCATION</a> property.
INSTALLPROPERTY_INSTALLSOURCE	The installation source. For more information, see the <a href="#">SourceDir</a> property.
INSTALLPROPERTY_LOCALPACKAGE	The local cached package.

INSTALLPROPERTY_PUBLISHER	The publisher. For information, see <a href="#">Manufacturer</a> property.
INSTALLPROPERTY_URLINFOABOUT	URL information, see <a href="#">ARPURLINFO</a> property.
INSTALLPROPERTY_URLUPDATEINFO	The URL update information. For information, see <a href="#">ARPURLUPDATE</a> property.
INSTALLPROPERTY_VERSIONMINOR	The minor product version that is derived from <a href="#">ProductVersion</a> property.
INSTALLPROPERTY_VERSIONMAJOR	The major product version that is derived from <a href="#">ProductVersion</a> property.
INSTALLPROPERTY_VERSIONSTRING	The product version string. For more information, see <a href="#">ProductVersion</a> property.

To retrieve the product ID, registered owner, or registered company from applications that are installed, set *szProperty* to one of the following text string values.

Value	Description
ProductID	The product identifier. For more information, see the <a href="#">ProductID</a> property.
RegCompany	The company that is registered to use the product.
RegOwner	The owner who is registered to use the product.

To retrieve the instance type of the product, set *szProperty* to the following value. This property is available for advertised or installed products.

<b>Value</b>	<b>Description</b>
InstanceType	A missing value or a value of 0 (zero) indicates a normal product installation. A value of one (1) indicates a product installed using a multiple instance transform and the <b>MSINEWINSTANCE</b> property. Available with the Windows Installer running Windows Server 2003 or Windows XP with SP1. For more information, see <a href="#">Installing Multiple Instances of Products and Patches</a> .

The properties in the following table can be retrieved from applications that are advertised or installed. These properties cannot be retrieved for product instances that are installed under a per-user-unmanaged context for user accounts other than current user account.

<b>Property</b>	<b>Description</b>
INSTALLPROPERTY_TRANSFORMS	Transforms.
INSTALLPROPERTY_LANGUAGE	Product language.
INSTALLPROPERTY_PRODUCTNAME	Human readable product name. For more information, see the <b>ProductName</b> property.
INSTALLPROPERTY_ASSIGNMENTTYPE	Equals 0 (zero) if the product is advertised installed per-user. Equals one (1) if the product is advertised installed per-computer for all users.

INSTALLPROPERTY_PACKAGECODE	Identifier of the package that a product is installed from. For more information, see the <a href="#">Package Codes</a> property.
INSTALLPROPERTY_VERSION	Product version derived from the <a href="#">ProductVersion</a> property.
INSTALLPROPERTY_PRODUCTICON	Primary icon for the package. For more information, see the <a href="#">ARPPRODUCTICON</a> property.
INSTALLPROPERTY_PACKAGENAME	Name of the original installation package.
INSTALLPROPERTY_AUTHORIZED_LUA_APP	A value of one (1) indicates a product that can be serviced by non-administrators using <a href="#">User Account Control (UAC) Patching</a> . A missing value or a value of 0 (zero) indicates that least-privilege patching is not enabled. Available in Windows Installer 3.0 or later.

*lpValue* [out, optional]

A pointer to a buffer that receives the property value. This buffer should be large enough to contain the information. If the buffer is too small, the function returns `ERROR_MORE_DATA` and sets *\*pcchValue* to the number of **TCHAR** in the value, not including the

terminating NULL character.

If *lpValue* is set to NULL and *pcchValue* is set to a valid pointer, the function returns `ERROR_SUCCESS` and sets *\*pcchValue* to the number of **TCHAR** in the value, not including the terminating NULL character. The function can then be called again to retrieve the value, with *lpValue* buffer large enough to contain *\*pcchValue* + 1 characters.

If *lpValue* and *pcchValue* are both set to NULL, the function returns `ERROR_SUCCESS` if the value exists, without retrieving the value.

*pcchValue* [in, out, optional]

A pointer to a variable that specifies the number of **TCHAR** in the *lpValue* buffer. When the function returns, this parameter is set to the size of the requested value whether or not the function copies the value into the specified buffer. The size is returned as the number of **TCHAR** in the requested value, not including the terminating null character.

This parameter can be set to NULL only if *lpValue* is also NULL. Otherwise, the function returns `ERROR_INVALID_PARAMETER`.

## Return Value

The **MsiGetProductInfoEx** function returns the following values.

Return code	Description
<code>ERROR_ACCESS_DENIED</code>	The calling process must have administrative privileges to get information for a product installed for a user other than the current user.
<code>ERROR_BAD_CONFIGURATION</code>	The configuration data is corrupt.
<code>ERROR_INVALID_PARAMETER</code>	An invalid parameter is passed to the function.
<code>ERROR_MORE_DATA</code>	A buffer is too small to hold the requested data.

ERROR_SUCCESS	The function completed successfully.
ERROR_UNKNOWN_PRODUCT	The product is unadvertised or uninstalled.
ERROR_UNKNOWN_PROPERTY	The property is unrecognized. <b>Note</b> The <b>MsiGetProductInfo</b> function returns ERROR_UNKNOWN_PROPERTY if the application being queried is advertised and not installed.
ERROR_FUNCTION_FAILED	An unexpected internal failure.

## Remarks

When the **MsiGetProductInfoEx** function returns, the *pcchValue* parameter contains the length of the string that is stored in the buffer. The count returned does not include the terminating null character. If the buffer is not big enough, **MsiGetProductInfoEx** returns ERROR\_MORE\_DATA, and the *pcchValue* parameter contains the size of the string, in **TCHAR**, without counting the null character.

The **MsiGetProductInfoEx** function (INSTALLPROPERTY\_LOCALPACKAGE) returns a path to the cached package. The cached package is for internal use only. Maintenance mode installations must be invoked through the [MsiConfigureFeature](#), [MsiConfigureProduct](#), or [MsiConfigureProductEx](#) functions.

**Windows Installer 2.0:** Not supported. This function is available beginning with Windows Installer version 3.0.

The **MsiGetProductInfo** function returns ERROR\_UNKNOWN\_PROPERTY if the application being queried is advertised and not installed. For example, if the application is advertised and not installed, a query for INSTALLPROPERTY\_INSTALLLOCATION returns an error of ERROR\_UNKNOWN\_PROPERTY.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiGetProductInfoExW</b> (Unicode) and <b>MsiGetProductInfoExA</b> (ANSI)

## See Also

[ARPHELPLINK](#)  
[ARPHELPTELEPHONE](#)  
[ARPINSTALLLOCATION](#)  
[ARPPRODUCTICON](#)  
[ARPURLINFOABOUT](#)  
[ARPURLUPDATEINFO](#)  
[Manufacturer](#)  
[Package Codes](#)  
[ProductID](#)  
[ProductName](#)  
[ProductVersion](#)  
[Properties](#)  
[Required Properties](#)  
[SourceDir](#)  
[System Status Functions](#)  
[MsiConfigureFeature](#)  
[MsiConfigureProduct](#)

**MsiConfigureProductEx**  
**ProductCode**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetProductInfoFromScript Function

The **MsiGetProductInfoFromScript** function returns product information for a Windows Installer script file.

## Syntax

```
C++UINT MsiGetProductInfoFromScript(  
    __in    LPCTSTR szScriptFile,  
    __out   LPTSTR lpProductBuf39,  
    __out   LANGID *plgidLanguage,  
    __out   DWORD *pdwVersion,  
    __out   LPTSTR lpNameBuf,  
    __inout DWORD *pcchNameBuf,  
    __out   LPTSTR lpPackageBuf,  
    __inout DWORD *pcchPackageBuf  
);
```

## Parameters

*szScriptFile* [in]

A null-terminated string specifying the full path to the script file. The script file is the advertise script that was created by calling [MsiAdvertiseProduct](#) or [MsiAdvertiseProductEx](#).

*lpProductBuf39* [out]

Points to a buffer that receives the product code. The buffer must be 39 characters long. The first 38 characters are for the product code [GUID](#), and the last character is for the terminating null character.

*plgidLanguage* [out]

Points to a variable that receives the product language.

*pdwVersion* [out]

Points to a buffer that receives the product version.

*lpNameBuf* [out]

Points to a buffer that receives the product name. The buffer

includes a terminating null character.

*pcchNameBuf* [in, out]

Points to a variable that specifies the size, in characters, of the buffer pointed to by the *lpNameBuf* parameter. This size should include the terminating null character. When the function returns, this variable contains the length of the string stored in the buffer. The count returned does not include the terminating null character. If the buffer is not large enough, the function returns `ERROR_MORE_DATA`, and the variable contains the size of the string in characters, without counting the null character.

*lpPackageBuf* [out]

Points to a buffer that receives the package name. The buffer includes the terminating null character.

*pcchPackageBuf* [in, out]

Points to a variable that specifies the size, in characters, of the buffer pointed to by the *lpPackageNameBuf* parameter. This size should include the terminating null character. When the function returns, this variable contains the length of the string stored in the buffer. The count returned does not include the terminating null character. If the buffer is not large enough, the function returns `ERROR_MORE_DATA`, and the variable contains the size of the string in characters, without counting the null character.

## Return Value

Value	Meaning
<code>ERROR_SUCCESS</code>	The function completed successfully.
<code>ERROR_INVALID_PARAMETER</code>	An invalid argument was passed to the function.
<code>ERROR_MORE_DATA</code>	A buffer was too small to hold the entire value.
<code>ERROR_INSTALL_FAILURE</code>	Could not get script information.

ERROR_CALL_NOT_IMPLEMENTED	This function is only available on Windows 2000 and Windows XP.
----------------------------	---

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiGetProductInfoFromScriptW</b> (Unicode) and <b>MsiGetProductInfoFromScriptA</b> (ANSI)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetProductProperty Function

The **MsiGetProductProperty** function retrieves product properties. These properties are in the product database.

## Syntax

```
C++UINT MsiGetProductProperty(  
    __in     MSIHANDLE hProduct,  
    __in     LPCTSTR szProperty,  
    __out    LPTSTR lpValueBuf,  
    __inout  DWORD *pcchValueBuf  
);
```

## Parameters

*hProduct* [in]

Handle to the product obtained from **MsiOpenProduct**.

*szProperty* [in]

Specifies the property to retrieve. This is case-sensitive.

*lpValueBuf* [out]

Pointer to a buffer that receives the property value. The value is truncated and null-terminated if *lpValueBuf* is too small. This parameter can be null.

*pcchValueBuf* [in, out]

Pointer to a variable that specifies the size, in characters, of the buffer pointed to by the *lpValueBuf* parameter. On input, this is the full size of the buffer, including a space for a terminating null character. If the buffer passed in is too small, the count returned does not include the terminating null character.

If *lpValueBuf* is null, *pcchValueBuf* can be null.

## Return Value

The **MsiGetProductProperty** function return the following values.



Value	Meaning
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_INVALID_HANDLE	An invalid handle was passed to the function.
ERROR_MORE_DATA	A buffer is too small to hold the entire property value.
ERROR_SUCCESS	The function completed successfully.

## Remarks

When the **MsiGetProductProperty** function returns, the *pcchValueBuf* parameter contains the length of the string stored in the buffer. The count returned does not include the terminating null character. If the buffer is not big enough, **MsiGetProductProperty** returns **ERROR\_MORE\_DATA**, and **MsiGetProductProperty** contains the size of the string, in characters, without counting the null character.

## Requirements

	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Version</b>	
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

**Unicode and ANSI  
names**

**MsiGetProductPropertyW (Unicode) and  
MsiGetProductPropertyA (ANSI)**

## See Also

[Product Query Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetShortcutTarget Function

The [MsiGetShortcutTarget](#) function examines a shortcut and returns its product, feature name, and component if available.

## Syntax

```
C++UINT MsiGetShortcutTarget(  
    __in    LPCTSTR szShortcutTarget,  
    __out   LPTSTR  szProductCode,  
    __out   LPTSTR  szFeatureId,  
    __out   LPTSTR  szComponentCode  
);
```

## Parameters

*szShortcutTarget* [in]

A null-terminated string specifying the full path to a shortcut.

*szProductCode* [out]

A GUID for the product code of the shortcut. This string buffer must be 39 characters long. The first 38 characters are for the [GUID](#), and the last character is for the terminating null character. This parameter can be null.

*szFeatureId* [out]

The feature name of the shortcut. The string buffer must be `MAX_FEATURE_CHARS+1` characters long. This parameter can be null.

*szComponentCode* [out]

A GUID of the component code. This string buffer must be 39 characters long. The first 38 characters are for the [GUID](#), and the last character is for the terminating null character. This parameter can be null.

## Return Value

`ERROR_SUCCESS`

The function succeeded.

ERROR\_FUNCTION\_FAILED

The function failed.

## Remarks

If the function fails, and the shortcut exists, the regular contents of the shortcut may be accessed through the **IShellLink** interface.

Otherwise, the state of the target may be determined by using the [Installer Selection Functions](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiGetShortcutTargetW</b> (Unicode) and <b>MsiGetShortcutTargetA</b> (ANSI)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetUserInfo Function

The **MsiGetUserInfo** function returns the registered user information for an installed product.

## Syntax

```
C++USERINFOSTATE MsiGetUserInfo(  
    __in    LPCTSTR szProduct,  
    __out   LPTSTR lpUserNameBuf,  
    __inout DWORD *pcchUserNameBuf,  
    __out   LPTSTR lpOrgNameBuf,  
    __inout DWORD *pcchOrgNameBuf,  
    __in    LPTSTR lpSerialBuf,  
    __inout DWORD *pcchSerialBuf  
);
```

## Parameters

*szProduct* [in]

Specifies the product code for the product to be queried.

*lpUserNameBuf* [out]

Pointer to a variable that receives the name of the user.

*pcchUserNameBuf* [in, out]

Pointer to a variable that specifies the size, in characters, of the buffer pointed to by the *lpUserNameBuf* parameter. This size should include the terminating null character.

*lpOrgNameBuf* [out]

Pointer to a buffer that receives the organization name.

*pcchOrgNameBuf* [in, out]

Pointer to a variable that specifies the size, in characters, of the buffer pointed to by the *lpOrgNameBuf* parameter. On input, this is the full size of the buffer, including a space for a terminating null character. If the buffer passed in is too small, the count returned does not include the terminating null character.

*lpSerialBuf* [in]

Pointer to a buffer that receives the product ID.

*pcchSerialBuf* [in, out]

Pointer to a variable that specifies the size, in characters, of the buffer pointed to by the *lpSerialBuf* parameter. On input, this is the full size of the buffer, including a space for a terminating null character. If the buffer passed in is too small, the count returned does not include the terminating null character.

## Return Value

Value	Meaning
USERINFOSTATE_ABSENT	Some or all of the user information is absent.
USERINFOSTATE_INVALIDARG	One of the function parameters was invalid.
USERINFOSTATE_MOREDATA	A buffer is too small to hold the requested data.
USERINFOSTATE_PRESENT	The function completed successfully.
USERINFOSTATE_UNKNOWN	The product code does not identify a known product.

## Remarks

When the **MsiGetUserInfo** function returns, the *pcchNameBuf* parameter contains the length of the class string stored in the buffer. The count returned does not include the terminating null character. If the buffer is not big enough, the **MsiGetUserInfo** function returns USERINFOSTATE\_MOREDATA, and **MsiGetUserInfo** contains the size of the string, in characters, without counting the null character.

The user information is considered to be present even in the absence of a company name.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiGetUserInfoW</b> (Unicode) and <b>MsiGetUserInfoA</b> (ANSI)

## See Also

[System Status Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiInstallMissingComponent Function

The **MsiInstallMissingComponent** function installs files that are unexpectedly missing.

## Syntax

```
C++UINT MsiInstallMissingComponent(  
    __in LPCTSTR szProduct,  
    __in LPCTSTR szComponent,  
    __in INSTALLSTATE eInstallState  
);
```

## Parameters

*szProduct* [in]

Specifies the product code for the product that owns the component to be installed.

*szComponent* [in]

Identifies the component to be installed.

*eInstallState* [in]

Specifies the way the component should be installed. This parameter must be one of the following values.

Value	Meaning
INSTALLSTATE_LOCAL	The component should be locally installed.
INSTALLSTATE_SOURCE	The component should be installed to run from the source.
INSTALLSTATE_DEFAULT	The component should be installed according to the installer defaults.

## Return Value

Value	Meaning
ERROR_BAD_CONFIGURATION	The configuration information is corrupt.
ERROR_INSTALL_FAILURE	The installation failed.
ERROR_INSTALL_SOURCE_ABSENT	The source was unavailable.
ERROR_INSTALL_SUSPEND	The installation was suspended.
ERROR_INSTALL_USEREXIT	The user exited the installation.
ERROR_INVALID_PARAMETER	One of the parameters is invalid.
ERROR_SUCCESS	The function completed successfully.
ERROR_UNKNOWN_PRODUCT	The product code is unrecognized.

For more information about error messages, see [Displayed Error Messages](#)

## Remarks

The **MsilInstallMissingComponent** function resolves the feature(s) that the component belongs to. Then, the product feature that requires the least additional disk space is installed.

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or

<b>Version</b>	Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiInstallMissingComponentW</b> (Unicode) and <b>MsiInstallMissingComponentA</b> (ANSI)

## See Also

[Multiple-Package Installations](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiInstallMissingFile Function

The **MsiInstallMissingFile** function installs files that are unexpectedly missing.

## Syntax

```
C++UINT MsiInstallMissingFile(  
    __in LPCTSTR szProduct,  
    __in LPCTSTR szFile  
);
```

## Parameters

*szProduct* [in]

Specifies the product code for the product that owns the file to be installed.

*szFile* [in]

Specifies the file to be installed.

## Return Value

Value	Meaning
ERROR_BAD_CONFIGURATION	The configuration information is corrupt.
ERROR_INSTALL_FAILURE	The installation failed.
ERROR_INSTALL_SOURCE_ABSENT	The source was unavailable.
ERROR_INSTALL_SUSPEND	The installation was suspended.
ERROR_INSTALL_USEREXIT	The user exited the installation.
ERROR_INVALID_PARAMETER	A parameter was invalid.
ERROR_SUCCESS	The function completed

	successfully.
ERROR_UNKNOWN_PRODUCT	The product code is unrecognized.

For more information about error messages, see [Displayed Error Messages](#).

## Remarks

The [MsiInstallMissingFile](#) function obtains the component that the file belongs to from the file table. Then, the product feature that requires the least additional disk space is installed.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiInstallMissingFileW</b> (Unicode) and <b>MsiInstallMissingFileA</b> (ANSI)

## See Also

[Installation and Configuration Functions](#)  
[Multiple-Package Installations](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiInstallProduct Function

The **MsiInstallProduct** function installs or uninstalls a product.

## Syntax

```
C++UINT MsiInstallProduct(  
    __in LPCTSTR szPackagePath,  
    __in LPCTSTR szCommandLine  
);
```

## Parameters

*szPackagePath* [in]

A null-terminated string that specifies the path to the location of the Windows Installer package. The string value can contain a URL (e.g. `http://packageLocation/package/package.msi`), a network path (e.g. `\\packageLocation\package.msi`), a file path (e.g. `file://packageLocation/package.msi`), or a local path (e.g. `D:\packageLocation\package.msi`).

*szCommandLine* [in]

A null-terminated string that specifies the command line property settings. This should be a list of the format *Property=Setting* *Property=Setting*. For more information, see [About Properties](#).

To perform an administrative installation, include `ACTION=ADMIN` in *szCommandLine*. For more information, see the [ACTION](#) property.

## Return Value

Value	Meaning
ERROR_SUCCESS	The function completes successfully.
An error relating to an action	For more information, see <a href="#">Error Codes</a> .
<a href="#">Initialization Error</a>	An error that relates to initialization occurred.

---

For more information, see [Displayed Error Messages](#).

## Remarks

The **MsiInstallProduct** function displays the user interface with the current settings and log mode.

- You can change user interface settings by using the [MsiSetInternalUI](#), [MsiSetExternalUI](#), or [MsiSetExternalUIRecord](#) functions.
- You can set the log mode by using the [MsiEnableLog](#) function.
- You can completely remove a product by setting REMOVE=ALL in *szCommandLine*.

For more information, see [REMOVE](#) Property.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiInstallProductW</b> (Unicode) and <b>MsiInstallProductA</b> (ANSI)

## See Also

[Displayed Error Messages](#)

[Error Codes](#)

[Initialization Error](#)

[Installation and Configuration Functions](#)

[Multiple-Package Installations](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsilsProductElevated Function

The **MsilsProductElevated** function returns whether or not the product is managed. Only applications that require elevated privileges for installation and being installed through advertisement are considered managed, which means that an application installed per-machine is always considered managed.

An application that is installed per-user is only considered managed if it is advertised by a local system process that is impersonating the user. For more information, see [Advertising a Per-User Application to be Installed with Elevated Privileges](#).

**MsilsProductElevated** verifies that the local system owns the product registry data. The function does not refer to account policies such as [AlwaysInstallElevated](#).

## Syntax

```
C++UINT MsiIsProductElevated(  
    __in LPCTSTR szProductCode,  
    __out BOOL *pfElevated  
);
```

## Parameters

*szProductCode* [in]

The full product code GUID of the product.

This parameter is required and cannot be null or empty.

*pfElevated* [out]

A pointer to a BOOL for the result.

This parameter cannot be null.

## Return Value

If the function succeeds, the return value is `ERROR_SUCCESS`, and *pfElevated* is set to `TRUE` if the product is a managed application.

If the function fails, the return value is one of the error codes identified in the following table.

<b>Return code</b>	<b>Description</b>
ERROR_UNKNOWN_PRODUCT	The product is not currently known.
ERROR_INVALID_PARAMETER	An invalid argument is passed to the function.
ERROR_BAD_CONFIGURATION	The configuration information for the product is invalid.
ERROR_FUNCTION_FAILED	The function failed.
ERROR_CALL_NOT_IMPLEMENTED	The function is not available for a specific platform.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiIsProductElevatedW</b> (Unicode) and <b>MsiIsProductElevatedA</b> (ANSI)

## See Also

[Determining Installation Context](#)

[Installing a Package with Elevated Privileges for a Non-Admin](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiJoinTransaction Function

The **MsiJoinTransaction** function requests that the Windows Installer make the current process the owner of the *transaction* installing the multiple-package installation.

**Windows Installer 4.0 and earlier:** Not supported. This function is available beginning with Windows Installer 4.5.

## Syntax

```
C++UINT WINAPI MsiJoinTransaction(  
    __in    MSIHANDLE hTransactionID,  
    __in    DWORD dwTransactionAttributes,  
    __out   HANDLE *phChangeOfOwnerEvent  
);
```

## Parameters

*hTransactionID* [in]

The transaction ID, which identifies the transaction and is the identifier returned by the **MsiBeginTransaction** function.

*dwTransactionAttributes* [in]

Attributes of the multiple-package installation.

Value	Meaning
0	When 0 or no value is set, Windows Installer closes the UI from the previous installation.

MSITRANSACTION_CHAIN_EMBEDDEDUI	Set this attribute to request that the Windows Installer not shutdown the embedded UI until the transaction is complete.
MSITRANSACTION_JOIN_EXISTING_EMBEDDEDUI	Set this attribute to request that the Windows Installer transfer the embedded UI from the original installation. If the original installation has no embedded UI, setting this attribute does nothing.

*phChangeOfOwnerEvent* [out]

This parameter returns a handle to an event that is set when the **MsiJoinTransaction** function changes the owner of the transaction to a new owner. The current owner can use this to determine when ownership of the transaction has changed. Leaving a transaction without an owner will roll back the transaction.

## Return Value

The **MsiJoinTransaction** function can return the following values.

Value	Meaning
ERROR_ACCESS_DENIED	The user that owns the transaction and the user that joins the transaction are not the same.
ERROR_INVALID_PARAMETER	A parameter that is not valid is passed to the function.
ERROR_INSTALL_ALREADY_RUNNING	The owner cannot be changed while an active installation is in progress.
ERROR_INVALID_HANDLE_STATE	The transaction ID provided is not valid.

## Remarks

Because a transaction can be owned by no more than one process at a time, the functions authored into the [MsiEmbeddedChainer table](#) can use **MsiJoinTransaction** to request ownership of the transaction before using the Windows Installer API to configure or install an application. The installer verifies that there is no installation in progress. The installer verifies that the process requesting ownership and the process that currently owns the transaction share a parent process in the same process tree. If the function succeeds, the process that calls

**MsiJoinTransaction** becomes the current owner of the transaction.

**MsiJoinTransaction** sets the internal UI of the new installation to the UI level of the original installation. After the new installation owns the transaction, it can call [MsiSetInternalUI](#) to change the UI level. This enables the new installation to run at a higher UI level than the original installation.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.5 on Windows Vista, Windows XP, Windows Server 2003, and Windows Server 2008. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

### [Multiple Package Installations](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiLocateComponent Function

The **MsiLocateComponent** function returns the full path to an installed component without a product code. This function attempts to determine the product using **MsiGetProductCode**, but is not guaranteed to find the correct product for the caller. **MsiGetComponentPath** should always be called when possible.

## Syntax

```
C++INSTALLSTATE MsiLocateComponent(  
    __in    LPCTSTR szComponent,  
    __out   LPTSTR lpPathBuf,  
    __inout DWORD *pcchBuf  
);
```

## Parameters

*szComponent* [in]

Specifies the component ID of the component to be located.

*lpPathBuf* [out]

Pointer to a variable that receives the path to the component. The variable includes the terminating null character.

*pcchBuf* [in, out]

Pointer to a variable that specifies the size, in characters, of the buffer pointed to by the *lpPathBuf* parameter. On input, this is the full size of the buffer, including a space for a terminating null character. Upon success of the **MsiLocateComponent** function, the variable pointed to by *pcchBuf* contains the count of characters not including the terminating null character. If the size of the buffer passed in is too small, the function returns `INSTALLSTATE_MOREDATA`.

If *lpPathBuf* is null, *pcchBuf* can be null.

## Return Value

Value	Meaning
-------	---------

INSTALLSTATE_NOTUSED	The component being requested is disabled on the computer.
INSTALLSTATE_ABSENT	The component is not installed. See Remarks.
INSTALLSTATE_INVALIDARG	One of the function parameters is invalid.
INSTALLSTATE_LOCAL	The component is installed locally.
INSTALLSTATE_MOREDATA	The buffer provided was too small.
INSTALLSTATE_SOURCE	The component is installed to run from source.
INSTALLSTATE_SOURCEABSENT	The component source is inaccessible.
INSTALLSTATE_UNKNOWN	The product code or component ID is unknown. See Remarks.

## Remarks

The **MsiLocateComponent** function might return `INSTALLSTATE_ABSENT` or `INSTALL_STATE_UNKNOWN`, for the following reasons:

- `INSTALLSTATE_ABSENT`  
The application did not properly ensure that the feature was installed by calling **MsiUseFeature** and, if necessary, **MsiConfigureFeature**.
- `INSTALLSTATE_UNKNOWN`  
The feature is not published. The application should have determined this earlier by calling **MsiQueryFeatureState** or **MsiEnumFeatures**. The application makes these calls while it

initializes. An application should only use features that are known to be published. Since `INSTALLSTATE_UNKNOWN` should have been returned by **MsiUseFeature** as well, either **MsiUseFeature** was not called, or its return value was not properly checked.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiLocateComponentW</b> (Unicode) and <b>MsiLocateComponentA</b> (ANSI)

## See Also

[Component-Specific Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiNotifySidChange Function

The **MsiNotifySidChange** function notifies and updates the Windows Installer internal information with changes to user SIDs.

## Syntax

```
C++UINT MsiNotifySidChange(  
    __in LPCTSTR szOldSid,  
    __in LPCTSTR szNewSid  
);
```

## Parameters

*szOldSid* [in]

Null-terminated string that specifies the string value of the previous security identifier(SID).

*szNewSid* [in]

Null-terminated string that specifies the string value of the new security identifier(SID).

## Return Value

Value	Meaning
ERROR_INVALID_PARAMETER	An invalid parameter is passed to the function. This error returned if any of the parameters is NULL.
ERROR_SUCCESS	The function succeeded.
ERROR_OUTOFMEMORY	Insufficient memory was available.
ERROR_FUNCTION_FAILED	Internal failure during execution.

## Remarks

**Windows Installer 2.0 and Windows Installer 3.0:** Not supported. This function is available beginning with Windows Installer 3.1.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.1 on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiNotifySidChangeW</b> (Unicode) and <b>MsiNotifySidChangeA</b> (ANSI)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiOpenPackage Function

The **MsiOpenPackage** function opens a package to use with the functions that access the product database. The **MsiCloseHandle** function must be called with the handle when the handle is not needed.

**Note** Initialize COM on the same thread before calling the **MsiOpenPackage**, **MsiOpenPackageEx**, or **MsiOpenProduct** function.

## Syntax

```
C++UINT MsiOpenPackage(  
    __in LPCTSTR szPackagePath,  
    __out MSIHANDLE *hProduct  
);
```

## Parameters

*szPackagePath* [in]

The path to the package.

*hProduct* [out]

A pointer to a variable that receives the product handle.

## Return Value

Value	Meaning
ERROR_BAD_CONFIGURATION	The configuration information is corrupt.
ERROR_INSTALL_FAILURE	The product could not be opened.
ERROR_INSTALL_REMOTE_PROHIBITED	Windows Installer does not permit installation from a remote desktop connection.
ERROR_INVALID_PARAMETER	An invalid parameter is

	passed to the function.
ERROR_SUCCESS	The function completes successfully.

If this function fails, it may return a system error code. For more information, see **System Error Codes**.

## Remarks

MsiOpenPackage can accept an opened database handle in the form "#nnnn", where nnnn is the database handle in string form, i.e. #123, instead of a path to the package. This is intended for development tasks such as running validation actions, or for use with database management tools.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiOpenPackageW</b> (Unicode) and <b>MsiOpenPackageA</b> (ANSI)

## See Also

[Product Query Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiOpenPackageEx Function

The **MsiOpenPackageEx** function opens a package to use with functions that access the product database. The **MsiCloseHandle** function must be called with the handle when the handle is no longer needed.

**Note** Initialize COM on the same thread before calling the **MsiOpenPackage**, **MsiOpenPackageEx**, or **MsiOpenProduct** function.

## Syntax

```
C++UINT WINAPI MsiOpenPackageEx(  
    __in LPCTSTR szPackagePath,  
    __in DWORD dwOptions,  
    __out MSIHANDLE *hProduct  
);
```

## Parameters

*szPackagePath* [in]

The path to the package.

*dwOptions* [in]

The bit flags to indicate whether or not to ignore the computer state. Pass in 0 (zero) to use **MsiOpenPackage** behavior.

Constant	Meaning
MSIOPENPACKAGEFLAGS_IGNOREMACHINESTATE 1	Ignore the computer state when creating the product handle.

*hProduct* [out]

A pointer to a variable that receives the product handle.

## Return Value

Value	Meaning
ERROR_BAD_CONFIGURATION	The configuration information is corrupt.
ERROR_INSTALL_FAILURE	The product could not be opened.
ERROR_INSTALL_REMOTE_PROHIBITED	Windows Installer does not permit installation from a remote desktop connection.
ERROR_INVALID_PARAMETER	An invalid parameter is passed to the function.
ERROR_SUCCESS	The function completes successfully.

If this function fails, it may return a system error code. For more information, see System Error Codes.

## Remarks

To create a restricted product handle that is independent of the current machine state and incapable of changing the current machine state, use **MsiOpenPackageEx** with `MSIOPENPACKAGEFLAGS_IGNOREMACHINESTATE` set in *dwOptions*.

Note that if *dwOptions* is `MSIOPENPACKAGEFLAGS_IGNOREMACHINESTATE` or 1, **MsiOpenPackageEx** ignores the current machine state when creating the product handle. If the value of *dwOptions* is 0, **MsiOpenPackageEx** is the same as **MsiOpenPackage** and creates a product handle that is dependent upon whether the package specified by *szPackagePath* is already installed on the computer.

The restricted handle created by using **MsiOpenPackageEx** with `MSIOPENPACKAGEFLAGS_IGNOREMACHINESTATE` only permits execution of dialogs, a subset of the standard actions, and custom actions that set properties ( [Custom Action Type 35](#), [Custom Action Type 51](#), and [Custom Action Type 19](#)). The restricted handle prevents the use of custom actions that run [Dynamic-Link Libraries](#), [Executable Files](#) or [Scripts](#).

You can call **MsiDoAction** on the following standard actions using the restricted handle. All other actions return `ERROR_FUNCTION_NOT_CALLED` if called with the restricted handle.

- [ADMIN](#)
- [ADVERTISE](#)
- [INSTALL](#)
- [SEQUENCE](#)
- [AppSearch](#) action
- [CCPSearch](#)
- [CostFinalize](#)
- [CostInitialize](#)
- [FileCost](#)
- [FindRelatedProducts](#)
- [IsolateComponents](#) action
- [LaunchConditions](#)
- [MigrateFeatureStates](#)
- [ResolveSource](#)
- [RMCCPSearch](#)
- [ValidateProductID](#)

The **MsiCloseHandle** function must be called when the handle is not needed.

## Requirements

---

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiOpenPackageExW</b> (Unicode) and <b>MsiOpenPackageExA</b> (ANSI)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiOpenProduct Function

The **MsiOpenProduct** function opens a product for use with the functions that access the product database. The **MsiCloseHandle** function must be called with the handle when the handle is no longer needed.

**Note** Initialize COM on the same thread before calling the **MsiOpenPackage**, **MsiOpenPackageEx**, or **MsiOpenProduct** function.

## Syntax

```
C++UINT MsiOpenProduct(  
    __in    LPCTSTR szProduct,  
    __out   MSIHANDLE *hProduct  
);
```

## Parameters

*szProduct* [in]

Specifies the product code of the product to be opened.

*hProduct* [out]

Pointer to a variable that receives the product handle.

## Return Value

Value	Meaning
ERROR_BAD_CONFIGURATION	The configuration information is corrupt.
ERROR_INSTALL_FAILURE	The product could not be opened.
ERROR_INSTALL_SOURCE_ABSENT	The source was unavailable.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.

ERROR_SUCCESS	The function completed successfully.
ERROR_UNKNOWN_PRODUCT	The product code was unrecognized.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiOpenProductW</b> (Unicode) and <b>MsiOpenProductA</b> (ANSI)

## See Also

[Product Query Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiProcessAdvertiseScript Function

The [MsiProcessAdvertiseScript](#) function processes an advertise script file into the specified locations.

## Syntax

```
C++UINT MsiProcessAdvertiseScript(  
    __in LPCTSTR szScriptFile,  
    __in LPCTSTR szIconFolder,  
    __in HKEY hRegData,  
    __in BOOL fShortcuts,  
    __in BOOL fRemoveItems  
);
```

## Parameters

*szScriptFile* [in]

The full path to a script file generated by [MsiAdvertiseProduct](#) or [MsiAdvertiseProductEx](#).

*szIconFolder* [in]

An optional path to a folder in which advertised icon files and transform files are located. If this parameter is null, no icon or transform files are written.

*hRegData* [in]

A registry key under which registry data is to be written. If this parameter is null, the installer writes the registry data under the appropriate key, based on whether the advertisement is per-user or per-machine. If this parameter is non-null, the script will write the registry data under the specified registry key rather than the normal location. In this case, the application will not get advertised to the user.

*fShortcuts* [in]

TRUE if shortcuts should be created. If a special folder is returned by [SHGetSpecialFolderLocation](#) it will hold the shortcuts.

*fRemoveItems* [in]

TRUE if specified items are to be removed instead of created.

## Return Value

Value	Meaning
ERROR_SUCCESS	The function completed successfully.
ERROR_ACCESS_DENIED	The calling process was not running under the LocalSystem account.
An error relating to an action	See <a href="#">Error Codes</a> .
<a href="#">Initialization Error</a>	An error relating to initialization occurred.
ERROR_CALL_NOT_IMPLEMENTED	This function is not available for this platform.

## Remarks

The process calling this function must be running under the LocalSystem account. To advertise an application for per-user installation to a targeted user, the thread that calls this function must impersonate the targeted user. If the thread calling this function is not impersonating a targeted user, the application is advertised to all users for installation with elevated privileges.

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See

<b>Version</b>	the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiProcessAdvertiseScriptW</b> (Unicode) and <b>MsiProcessAdvertiseScriptA</b> (ANSI)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiProvideAssembly Function

The **MsiProvideAssembly** function returns the full path to a Windows Installer component that contains an assembly. The function prompts for a source and performs any necessary installation. **MsiProvideAssembly** increments the usage count for the feature.

## Syntax

```
C++UINT MsiProvideAssembly(  
    __in    LPCTSTR szAssemblyName,  
    __in    LPCTSTR szAppContext,  
    __in    DWORD  dwInstallMode,  
    __in    DWORD  dwAssemblyInfo,  
    __out   LPTSTR lpPathBuf,  
    __inout DWORD  *pcchPathBuf  
);
```

## Parameters

*szAssemblyName* [in]

The assembly name as a string.

*szAppContext* [in]

Set to null for global assemblies. For private assemblies, set *szAppContext* to the full path of the application configuration file or to the full path of the executable file of the application to which the assembly has been made private.

*dwInstallMode* [in]

Defines the installation mode. This parameter can be one of the following values.

Value	Meaning
INSTALLMODE_DEFAULT	Provide the component and any installation necessary to provide the component. If the key file is missing, the requested feature, or a component is missing, reinstall the feature.

	<p><b>MsiReinstallFeature</b> with bits set: REINSTALLMC  REINSTALLMODE_FIL  REINSTALLMODE_FIL  REINSTALLMODE_MA  REINSTALLMODE_US  REINSTALLMODE_SH</p>
INSTALLMODE_EXISTING	<p>Provide the component or exists. Otherwise return ERROR_FILE_NOT_FOUND</p> <p>This mode verifies that the component exists.</p>
INSTALLMODE_NODETECTION	<p>Provide the component or exists. Otherwise return ERROR_FILE_NOT_FOUND</p> <p>This mode only checks that registered and does not verify if the component exists.</p>
INSTALLMODE_NOSOURCERESOLUTION	<p>Provide the component or installation state is INSTALLSTATE_LOCAL installation state is INSTALLSTATE_SOURCE  ERROR_INSTALL_SOURCES  Otherwise return ERROR_FILE_NOT_FOUND  only checks that the component and does not verify that the</p>
INSTALLMODE_NODETECTION_ANY	<p>Provide the component if any installed product. Otherwise return ERROR_FILE_NOT_FOUND  only checks that the component and does not verify that the component exists. This file</p>

	INSTALLMODE_NODE except that with this flag product that has installed opposed to the last produ the INSTALLMODE_NC This flag can only be use <b>MsiProvideAssembly</b> .
combination of the REINSTALLMODE flags	Call <b>MsiReinstallFeature</b> using this parameter for t parameter, and then provi

*dwAssemblyInfo* [in]

Assembly information and assembly type. Set to one of the following values.

Value	Meaning
MSIASSEMBLYINFO_NETASSEMBLY 0	.NET Assembly
MSIASSEMBLYINFO_WIN32ASSEMBLY 1	Win32 Assembly

*lpPathBuf* [out]

Pointer to a variable that receives the path to the component. This parameter can be null.

*pcchPathBuf* [in, out]

Pointer to a variable that specifies the size, in characters, of the buffer pointed to by the *lpPathBuf* parameter. On input, this is the full size of the buffer, including a space for a terminating null character. If the buffer passed in is too small, the count returned does not include the terminating null character.

If *lpPathBuf* is null, *pcchPathBuf* can be null.

## Return Value

<b>Value</b>	<b>Meaning</b>
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_FILE_NOT_FOUND	The feature is absent or broken. This error is returned for dwInstallMode = INSTALLMODE_EXISTING.
ERROR_INSTALL_FAILURE	The installation failed.
ERROR_INSTALL_NOTUSED	The component being requested is disabled on the computer.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_SUCCESS	The function completed successfully.
ERROR_UNKNOWN_FEATURE	The feature ID does not identify a known feature.
ERROR_UNKNOWN_COMPONENT	The component ID does not specify a known component.
ERROR_UNKNOWN_PRODUCT	The product code does not identify a known product.
INSTALLSTATE_UNKNOWN	An unrecognized product or a feature name was passed to the function.
ERROR_MORE_DATA	The buffer overflow is returned.
ERROR_NOT_ENOUGH_MEMORY	The system does not have enough memory to complete the operation. Available with Windows Server 2003.

ERROR_INSTALL_SOURCE_ABSENT	Unable to detect a source.
-----------------------------	----------------------------

For more information, see [Displayed Error Messages](#).

## Remarks

When the **MsiProvideAssembly** function succeeds, the *pcchPathBuf* parameter contains the length of the string in *lpPathBuf*.

The `INSTALLMODE_EXISTING` option cannot be used in combination with the `REINSTALLMODE` flag.

Features with components that contain a corrupted file or the wrong version of a file must be explicitly reinstalled by the user, or by having the application call **MsiReinstallFeature**.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiProvideAssemblyW</b> (Unicode) and <b>MsiProvideAssemblyA</b> (ANSI)

## See Also

[Component-Specific Functions](#)  
[Multiple-Package Installations](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiProvideComponent Function

The **MsiProvideComponent** function returns the full component path, performing any necessary installation. This function prompts for source if necessary and increments the usage count for the feature.

## Syntax

```
C++UINT MsiProvideComponent(  
    __in    LPCTSTR szProduct,  
    __in    LPCTSTR szFeature,  
    __in    LPCTSTR szComponent,  
    __in    DWORD  dwInstallMode,  
    __out   LPTSTR lpPathBuf,  
    __inout DWORD  *pcchPathBuf  
);
```

## Parameters

*szProduct* [in]

Specifies the product code for the product that contains the feature with the necessary component.

*szFeature* [in]

Specifies the feature ID of the feature with the necessary component.

*szComponent* [in]

Specifies the component code of the necessary component.

*dwInstallMode* [in]

Defines the installation mode. This parameter can be one of the following values.

Value	Meaning
INSTALLMODE_DEFAULT	Provide the component an installation necessary to p component. If the key file the requested feature, or a

	<p>missing, reinstall the feat  <b>MsiReinstallFeature</b> wit  bits set: REINSTALLMC  REINSTALLMODE_FIL  REINSTALLMODE_FIL  REINSTALLMODE_MA  REINSTALLMODE_US  REINSTALLMODE_SH</p>
INSTALLMODE_EXISTING	<p>Provide the component o:  exists. Otherwise return  ERROR_FILE_NOT_FC</p> <p>This mode verifies that th  component exists.</p>
INSTALLMODE_NODETECTION	<p>Provide the component o:  exists. Otherwise return  ERROR_FILE_NOT_FC</p> <p>This mode only checks th  registered and does not v  of the component exists.</p>
combination of the REINSTALLMODE flags	<p>Call <b>MsiReinstallFeatur</b>  using this parameter for t  parameter, and then provi</p>
INSTALLMODE_NOSOURCERESOLUTION	<p>Provide the component o:  installation state is  INSTALLSTATE_LOCA  installation state is  INSTALLSTATE_SOUR  ERROR_INSTALL_SOU  Otherwise return  ERROR_FILE_NOT_FC</p> <p>only checks that the comp  and does not verify that tl</p>

*lpPathBuf* [out]

Pointer to a variable that receives the path to the component. This parameter can be null.

*pcchPathBuf* [in, out]

Pointer to a variable that specifies the size, in characters, of the buffer pointed to by the *lpPathBuf* parameter. On input, this is the full size of the buffer, including a space for a terminating null character. If the buffer passed in is too small, the count returned does not include the terminating null character.

If *lpPathBuf* is null, *pcchBuf* can be null.

## Return Value

Value	Meaning
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_FILE_NOT_FOUND	The feature is absent or broken. this error is returned for dwInstallMode = INSTALLMODE_EXISTING.
ERROR_INSTALL_FAILURE	The installation failed.
ERROR_INSTALL_NOTUSED	The component being requested is disabled on the computer.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_SUCCESS	The function completed successfully.
ERROR_UNKNOWN_FEATURE	The feature ID does not identify a known feature.
ERROR_UNKNOWN_PRODUCT	The product code does not identify a known product.

INSTALLSTATE_UNKNOWN	An unrecognized product or a feature name was passed to the function.
ERROR_MORE_DATA	The buffer overflow is returned.
ERROR_INSTALL_SOURCE_ABSENT	Unable to detect a source.

For more information, see [Displayed Error Messages](#).

## Remarks

Upon success of the **MsiProvideComponent** function, the *pcchPathBuf* parameter contains the length of the string in *lpPathBuf*.

The **MsiProvideComponent** function combines the functionality of [MsiUseFeature](#), [MsiConfigureFeature](#), and [MsiGetComponentPath](#). You can use the **MsiProvideComponent** function to simplify the calling sequence. However, because this function increments the usage count, use it with caution to prevent inaccurate usage counts. The **MsiProvideComponent** function also provides less flexibility than the series of individual calls.

If the application is recovering from an unexpected situation, the application has probably already called **MsiUseFeature** and incremented the usage count. In this case, the application should call **MsiConfigureFeature** instead of **MsiProvideComponent** to avoid incrementing the count again.

The INSTALLMODE\_EXISTING option cannot be used in combination with the REINSTALLMODE flag.

Features with components containing a corrupted file or the wrong version of a file must be explicitly reinstalled by the user or by having the application call [MsiReinstallFeature](#).

## Requirements

	Windows Installer 5.0 on Windows Server 2008 R2
--	---

<b>Version</b>	or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiProvideComponentW</b> (Unicode) and <b>MsiProvideComponentA</b> (ANSI)

## See Also

[Component-Specific Functions](#)

[Multiple-Package Installations](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiProvideQualifiedComponent Function

The **MsiProvideQualifiedComponent** function returns the full component path for a qualified component and performs any necessary installation. This function prompts for source if necessary, and increments the usage count for the feature.

## Syntax

```
C++UINT MsiProvideQualifiedComponent(  
    __in    LPCTSTR szComponent,  
    __in    LPCTSTR szQualifier,  
    __in    DWORD  dwInstallMode,  
    __out   LPTSTR lpPathBuf,  
    __inout DWORD  *pcchPathBuf  
);
```

## Parameters

*szComponent* [in]

Specifies the component ID for the requested component. This may not be the GUID for the component itself, but rather a server that provides the correct functionality, as in the ComponentId column of the [PublishComponent table](#).

*szQualifier* [in]

Specifies a qualifier into a list of advertising components (from [PublishComponent Table](#)).

*dwInstallMode* [in]

Defines the installation mode. This parameter can be one of the following values.

Value	Meaning
INSTALLMODE_DEFAULT	Provide the component as an installation necessary to publish the component. If the key file

	<p>the requested feature, or a missing, reinstall the feature with <b>MsiReinstallFeature</b> with bits set: REINSTALLMODE_FIL REINSTALLMODE_FIL REINSTALLMODE_MA REINSTALLMODE_US REINSTALLMODE_SH</p>
INSTALLMODE_EXISTING	<p>Provide the component or exists. Otherwise return ERROR_FILE_NOT_FOUND</p> <p>This mode verifies that the component exists.</p>
INSTALLMODE_NODETECTION	<p>Provide the component or exists. Otherwise return ERROR_FILE_NOT_FOUND</p> <p>This mode only checks the registered and does not verify if the component exists.</p>
combination of the REINSTALLMODE flags	<p>Call <b>MsiReinstallFeature</b> feature using this parameter <i>dwReinstallMode</i> parameter the component.</p>
INSTALLMODE_NOSOURCERESOLUTION	<p>Provide the component or installation state is INSTALLSTATE_LOCAL installation state is INSTALLSTATE_SOURCE ERROR_INSTALL_SOURCES Otherwise, it returns ERROR_FILE_NOT_FOUND only checks that the component and does not verify that the</p>

---

*lpPathBuf* [out]

Pointer to a variable that receives the path to the component. This parameter can be null.

*pcchPathBuf* [in, out]

Pointer to a variable that specifies the size, in characters, of the buffer pointed to by the *lpPathBuf* parameter. On input, this is the full size of the buffer, including a space for a terminating null character. If the buffer passed in is too small, the count returned does not include the terminating null character.

If *lpPathBuf* is null, *pcchBuf* can be null.

## Return Value

Value	Meaning
ERROR_INDEX_ABSENT	The component qualifier is invalid or absent.
ERROR_SUCCESS	The function completed successfully.
ERROR_FILE_NOT_FOUND	The feature is absent or broken. This error is returned for <code>dwInstallMode = INSTALLMODE_EXISTING</code> .
ERROR_UNKNOWN_COMPONENT	The specified component is unknown.
An error relating to an action	See <a href="#">Error Codes</a> .
<a href="#">Initialization Error</a>	An error relating to initialization occurred.

## Remarks

Upon success of the **MsiProvideQualifiedComponent** function, the *pcchPathBuf* parameter contains the length of the string in *lpPathBuf*.

Features with components containing a corrupted file or the wrong version of a file must be explicitly reinstalled by the user or by having the application call **MsiReinstallFeature**.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiProvideQualifiedComponentW</b> (Unicode) and <b>MsiProvideQualifiedComponentA</b> (ANSI)

## See Also

[Component-Specific Functions](#)

[Error Codes](#)

[Displayed Error Messages](#)

[Initialization Error](#)

[Multiple-Package Installations](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiProvideQualifiedComponentEx Function

The **MsiProvideQualifiedComponentEx** function returns the full component path for a qualified component that is published by a product and performs any necessary installation. This function prompts for source if necessary and increments the usage count for the feature.

## Syntax

```
C++UINT MsiProvideQualifiedComponentEx(  
    __in    LPCTSTR szComponent,  
    __in    LPCTSTR szQualifier,  
    __in    DWORD  dwInstallMode,  
    __in    LPTSTR  szProduct,  
    __in    DWORD  dwUnused1,  
    __in    DWORD  dwUnused2,  
    __out   LPTSTR  lpPathBuf,  
    __inout DWORD  *pcchPathBuf  
);
```

## Parameters

*szComponent* [in]

Specifies the component ID that for the requested component. This may not be the GUID for the component itself but rather a server that provides the correct functionality, as in the ComponentId column of the [PublishComponent table](#).

*szQualifier* [in]

Specifies a qualifier into a list of advertising components (from [PublishComponent Table](#)).

*dwInstallMode* [in]

Defines the installation mode. This parameter can be one of the following values.

Value	Meaning

INSTALLMODE_DEFAULT	Provide the component an installation necessary to p component. If the key file the requested feature, or a missing, reinstall the feat <b>MsiReinstallFeature</b> with bits set: REINSTALLMC REINSTALLMODE_FIL REINSTALLMODE_FIL REINSTALLMODE_MA REINSTALLMODE_US REINSTALLMODE_SH
INSTALLMODE_EXISTING	Provide the component o exists. Otherwise return ERROR_FILE_NOT_FC  This mode verifies that th component exists.
INSTALLMODE_NODETECTION	Provide the component o exists. Otherwise return ERROR_FILE_NOT_FC  This mode only checks th registered and does not v of the component exists.
INSTALLMODE_EXISTING	Provide the component o exists, else return ERROR_FILE_NOT_FC
combination of the REINSTALLMODE flags	Call <b>MsiReinstallFeatur</b> using this parameter for t parameter, and then provi
INSTALLMODE_NOSOURCERESOLUTION	Provide the component o installation state is INSTALLSTATE_LOCA installation state is

INSTALLSTATE\_SOUR  
ERROR\_INSTALL\_SOU  
Otherwise return  
ERROR\_FILE\_NOT\_FC  
only checks that the comj  
and does not verify that tl

**szProduct** [in]

Specifies the product to match that has published the qualified component. If this is null, then this API works the same as [MsiProvideQualifiedComponent](#).

**dwUnused1** [in]

Reserved. Must be zero.

**dwUnused2** [in]

Reserved. Must be zero.

**lpPathBuf** [out]

Pointer to a variable that receives the path to the component. This parameter can be null.

**pcchPathBuf** [in, out]

Pointer to a variable that specifies the size, in characters, of the buffer pointed to by the *lpPathBuf* parameter. On input, this is the full size of the buffer, including a space for a terminating null character. If the buffer passed in is too small, the count returned does not include the terminating null character.

If *lpPathBuf* is null, *pcchBuf* can be null.

## Return Value

Value	Meaning
ERROR_INDEX_ABSENT	Component qualifier invalid or not present.
ERROR_SUCCESS	The function completed successfully.

ERROR_FILE_NOT_FOUND	The feature is absent or broken. this error is returned for <i>dwInstallMode</i> = INSTALLMODE_EXISTING.
ERROR_UNKNOWN_COMPONENT	The specified component is unknown.
An error relating to an action	See <a href="#">Error Codes</a> .
<a href="#">Initialization Error</a>	An error relating to initialization occurred.

## Remarks

Upon success of the **MsiProvideQualifiedComponentEx** function, the *pcchPathBuf* parameter contains the length of the string in *lpPathBuf*.

Features with components containing a corrupted file or the wrong version of a file must be explicitly reinstalled by the user or by having the application call **MsiReinstallFeature**.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI</b>	<b>MsiProvideQualifiedComponentExW</b> (Unicode)

**names**

and **MsiProvideQualifiedComponentExA** (ANSI)

## See Also

[Component-Specific Functions](#)

[Error Codes](#)

[Initialization Error](#)

[Displayed Error Messages](#)

[Multiple-Package Installations](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiQueryComponentState Function

The **MsiQueryComponentState** function returns the installed state for a component. This function can query for a component of an instance of a product that is installed under user accounts other than the current user provided the product is not advertised under the per-user-unmanaged context for a user account other than the current user. The calling process must have administrative privileges to get information for a product installed for a user other than the current user.

## Syntax

```
C++UINT MsiQueryComponentState(  
    __in    LPTSTR  szProductCode,  
    __in    LPTSTR  szUserSid,  
    __in    MSIINSTALLCONTEXT  dwContext,  
    __in    LPCTSTR szComponent,  
    __out   INSTALLSTATE *pdwState  
);
```

## Parameters

*szProductCode* [in]

Specifies the **ProductCode** GUID for the product that contains the component.

*szUserSid* [in]

Specifies the security identifier (SID) of the account under which the instance of the product being queried exists. If *dwContext* is not `MSIINSTALLCONTEXT_MACHINE`, null specifies the current user.

Type of SID	Meaning
NULL	NULL denotes the currently logged on user.
User SID	Specifies enumeration for a particular user in the system. An example of user SID is "S-1-3-64-2415071341-1358098788-

3127455600-2561".
-------------------

**Note** The special SID string s-1-5-18 (system) cannot be used to enumerate products installed as per-machine. If *dwContext* is `MSIINSTALLCONTEXT_MACHINE`, *szUserSid* must be null.

*dwContext* [in]

The installation context of the product instance being queried.

Name	Meaning
<code>MSIINSTALLCONTEXT_USERMANAGED</code>	Retrieves the component's state for the per-user-managed instance of the product.
<code>MSIINSTALLCONTEXT_USERUNMANAGED</code>	Retrieves the component's state for the per-user-non-managed instance of the product.
<code>MSIINSTALLCONTEXT_MACHINE</code>	Retrieves the component's state for the per-machine instance of the product.

*szComponent* [in]

Specifies the component being queried. Component code GUID of the component as found in the `ComponentID` column of the [Component](#) table.

*pdwState* [out]

Installation state of the component for the specified product instance. This parameter can return one of the following or null values.

---

<b>Value</b>	<b>Meaning</b>
INSTALLSTATE_LOCAL	The component is installed locally.
INSTALLSTATE_SOURCE	The component is installed to run from the source.

## Return Value

The **MsiQueryComponentState** function returns the following values.

<b>Value</b>	<b>Meaning</b>
ERROR_ACCESS_DENIED	The calling process must have administrative privileges to get information for a product installed for a user other than the current user.
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_SUCCESS	The function completed successfully.
ERROR_UNKNOWN_COMPONENT	The component ID does not identify a known component.
ERROR_UNKNOWN_PRODUCT	The product code does not identify a known product.
ERROR_FUNCTION_FAILED	Failures that cannot be ascribed to any Windows error code.
ERROR_MORE_DATA	Buffer too small to get the user SID.

For more information, see [Displayed Error Messages](#).

## Remarks

**Windows Installer 2.0:** Not supported. The **MsiQueryComponentState** function is available beginning with Windows Installer 3.0.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiQueryComponentStateW</b> (Unicode) and <b>MsiQueryComponentStateA</b> (ANSI)

## See Also

[Component Displayed Error Messages Installer Selection Functions](#)  
**ProductCode**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiQueryFeatureState Function

The [MsiQueryFeatureState](#) function returns the installed state for a product feature.

## Syntax

```
C++INSTALLSTATE MsiQueryFeatureState(  
    __in LPCTSTR szProduct,  
    __in LPCTSTR szFeature  
);
```

## Parameters

*szProduct* [in]

Specifies the product code for the product that contains the feature of interest.

*szFeature* [in]

Identifies the feature of interest.

## Return Value

Value	Meaning
INSTALLSTATE_ABSENT	The feature is not installed.
INSTALLSTATE_ADVERTISED	The feature is advertised
INSTALLSTATE_LOCAL	The feature is installed locally.
INSTALLSTATE_SOURCE	The feature is installed to run from source.
INSTALLSTATE_INVALIDARG	An invalid parameter was passed to the function.
INSTALLSTATE_UNKNOWN	The product code or feature ID is unknown.

## Remarks

The [MsiQueryFeatureState](#) function does not validate that the feature is actually accessible.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiQueryFeatureStateW</b> (Unicode) and <b>MsiQueryFeatureStateA</b> (ANSI)

## See Also

[System Status Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiQueryFeatureStateEx Function

The **MsiQueryFeatureStateEx** function returns the installed state for a product feature. This function can be used to query any feature of an instance of a product installed under the machine account or any context under the current user account or the per-user-managed context under any user account other than the current user. A user must have administrative privileges to get information for a product installed for a user other than the current user.

## Syntax

```
C++UINT MsiQueryFeatureStateEx(  
    __in      LPTSTR  szProductCode,  
    __in      LPTSTR  szUserSid,  
    __in      MSIINSTALLCONTEXT  dwContext,  
    __in      LPCTSTR szFeature,  
    __out_opt INSTALLSTATE *pdwState  
);
```

## Parameters

*szProductCode* [in]

**ProductCode** GUID of the product that contains the feature of interest.

*szUserSid* [in]

Specifies the security identifier (SID) of the account, under which, the instance of the product being queried exists. If *dwContext* is not `MSIINSTALLCONTEXT_MACHINE`, a null value specifies the current user.

Type of SID	Meaning
NULL	NULL denotes the currently logged on user.
User SID	Specifies enumeration for a particular user in the system. An example of user SID is

"S-1-3-64-2415071341-1358098788-3127455600-2561".

**Note** The special SID string s-1-5-18 (system) cannot be used to enumerate features of products installed as per-machine. If *dwContext* is MSIINSTALLCONTEXT\_MACHINE, *szUserSid* must be null.

*dwContext* [in]

The installation context of the product instance being queried.

<b>Name</b>	<b>Meaning</b>
MSIINSTALLCONTEXT_USERMANAGED	Retrieves the feature state for the per-user-managed instance of the product.
MSIINSTALLCONTEXT_USERUNMANAGED	Retrieves the feature state for the per-user-unmanaged instance of the product.  <b>Note</b> When the query is made on a product installed under the per-user-unmanaged context for a user account other than the current user, the function fails.
MSIINSTALLCONTEXT_MACHINE	Retrieves the feature state for the per-machine instance of the

	product.
--	----------

*szFeature* [in]

Specifies the feature being queried. Identifier of the feature as found in the **Feature** column of the [Feature table](#).

*pdwState* [out, optional]

Installation state of the feature for the specified product instance. This parameter can return one of the following or null.

Value	Meaning
INSTALLSTATE_ADVERTISED	This feature is advertised.
INSTALLSTATE_LOCAL	The feature is installed locally.
INSTALLSTATE_SOURCE	The feature is installed to run from source.

## Return Value

The **MsiQueryFeatureStateEx** function returns the following values.

Value	Meaning
ERROR_ACCESS_DENIED	A user must have administrative privileges to get information for a product installed for a user other than the current user.
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_SUCCESS	The function completed successfully.
ERROR_UNKNOWN_FEATURE	The feature ID does not identify a

	known feature.
ERROR_UNKNOWN_PRODUCT	The product code does not identify a known product.
ERROR_FUNCTION_FAILED	An unexpected internal failure.

For more information, see [Displayed Error Messages](#).

## Remarks

The **MsiQueryFeatureStateEx** function does not validate that the feature is actually accessible. The **MsiQueryFeatureStateEx** function does not validate the feature ID. ERROR\_UNKNOWN\_FEATURE is returned for any unknown feature ID. When the query is made on a product installed under the per-user-unmanaged context for a user account other than the current user, the function fails. In this case the function returns ERROR\_UNKNOWN\_FEATURE, or if the product is advertised only (not installed), ERROR\_UNKNOWN\_PRODUCT is returned.

**Windows Installer 2.0:** Not supported. The **MsiQueryFeatureStateEx** function is available beginning with Windows Installer 3.0.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h

<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiQueryFeatureStateExW</b> (Unicode) and <b>MsiQueryFeatureStateExA</b> (ANSI)

## See Also

[Displayed Error Messages](#)

[Feature Table](#)

[ProductCode](#)

[System Status Functions](#)

[MsiQueryFeatureState](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiQueryProductState Function

The [MsiQueryProductState](#) function returns the installed state for a product.

## Syntax

```
C++INSTALLSTATE MsiQueryProductState(  
    __in LPCTSTR szProduct  
);
```

## Parameters

*szProduct* [in]

Specifies the product code that identifies the product to be queried.

## Return Value

Value	Meaning
INSTALLSTATE_ABSENT	The product is installed for a different user.
INSTALLSTATE_ADVERTISED	The product is advertised but not installed.
INSTALLSTATE_DEFAULT	The product is installed for the current user.
INSTALLSTATE_INVALIDARG	An invalid parameter was passed to the function.
INSTALLSTATE_UNKNOWN	The product is neither advertised or installed.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiQueryProductStateW</b> (Unicode) and <b>MsiQueryProductStateA</b> (ANSI)

## See Also

[System Status Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiReinstallFeature Function

The **MsiReinstallFeature** function reinstalls features.

## Syntax

```
C++UINT MsiReinstallFeature(  
    __in LPCTSTR szProduct,  
    __in LPCTSTR szFeature,  
    __in DWORD dwReinstallMode  
);
```

## Parameters

*szProduct* [in]

Specifies the product code for the product that contains the feature to be reinstalled.

*szFeature* [in]

Specifies the feature to be reinstalled. The parent feature or child feature of the specified feature is not reinstalled. To reinstall the parent or child feature, you must call the **MsiReinstallFeature** function for each separately or use the **MsiReinstallProduct** function.

*dwReinstallMode* [in]

Specifies what to install. This parameter can be one or more of the following values.

Value	Meaning
REINSTALLMODE_FILEMISSING	Reinstall only if the file is missing.
REINSTALLMODE_FILEOLDERVERSION	Reinstall if the file is missing or is an older version.
REINSTALLMODE_FILEEQUALVERSION	Reinstall if the file is missing or is an equal or older version.

REINSTALLMODE_FILEEXACT	Reinstall if the file is missing or is a different version.
REINSTALLMODE_FILEVERIFY	Verify the checksum values and reinstall the file if they are missing or corrupt. This flag only repairs files that have the msidbFileAttributesChecksum in the Attributes column of the <a href="#">File table</a> .
REINSTALLMODE_FILEREPLACE	Force all files to be reinstalled regardless of checksum or version.
REINSTALLMODE_USERDATA	Rewrite all required registry entries from the <a href="#">Registry Table</a> that go to the <b>HKEY_CURRENT_USER</b> or <b>HKEY_USERS</b> registry hive.
REINSTALLMODE_MACHINEDATA	Rewrite all required registry entries from the <a href="#">Registry Table</a> that go to the <b>HKEY_LOCAL_MACHINE</b> or <b>HKEY_CLASSES_ROOT</b> registry hive. Rewrite all information from the <a href="#">Class Table</a> , <a href="#">Verb Table</a> , <a href="#">PublishComponent Table</a> , <a href="#">ProgID Table</a> , <a href="#">MIME Table</a> , <a href="#">Icon Table</a> , <a href="#">Extension Table</a> and <a href="#">AppID Table</a> regardless of machine or user assignment.

	<p>Reinstall all <a href="#">qualified components</a>.</p> <p>When reinstalling an application, this option runs the <a href="#">RegisterTypeLibraries</a> and <a href="#">InstallODBC</a> actions.</p>
REINSTALLMODE_SHORTCUT	<p>Reinstall all shortcuts and re-cache all icons overwriting any existing shortcuts and icons.</p>
REINSTALLMODE_PACKAGE	<p>Use to run from the source package and re-cache the local package. Do not use for the first installation of an application or feature.</p>

## Return Value

Return code	Description
ERROR_INSTALL_FAILURE	The installation failed.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_INSTALL_SERVICE_FAILURE	The installation service could not be accessed.
ERROR_INSTALL_SUSPEND	The installation was suspended and is incomplete.
ERROR_INSTALL_USEREXIT	The user canceled the installation.
ERROR_SUCCESS	The function completed

	successfully.
ERROR_UNKNOWN_FEATURE	The feature ID does not identify a known feature.
ERROR_UNKNOWN_PRODUCT	The product code does not identify a known product.

For more information, see [Displayed Error Messages](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiReinstallFeatureW</b> (Unicode) and <b>MsiReinstallFeatureA</b> (ANSI)

## See Also

[Installation and Configuration Functions](#)

[REINSTALLMODE Property](#)

[Multiple-Package Installations](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiReinstallProduct Function

The **MsiReinstallProduct** function reinstalls products.

## Syntax

```
C++UINT MsiReinstallProduct(  
    __in LPCTSTR szProduct,  
    __in DWORD dwReinstallMode  
);
```

## Parameters

*szProduct* [in]

Specifies the product code for the product to be reinstalled.

*dwReinstallMode* [in]

Specifies the reinstall mode. This parameter can be one or more of the following values.

Value	Meaning
REINSTALLMODE_FILEMISSING	Reinstall only if the file is missing.
REINSTALLMODE_FILEOLDERVERSION	Reinstall if the file is missing or is an older version.
REINSTALLMODE_FILEEQUALVERSION	Reinstall if the file is missing or is an equal or older version.
REINSTALLMODE_FILEEXACT	Reinstall if the file is missing or is a different version.
REINSTALLMODE_FILEVERIFY	Verify the checksum values and reinstall the file if they are missing or corrupt. This flag only repairs files that have the <code>msidbFileAttributesChecksum</code> attribute set.

	in the Attributes column of <a href="#">File table</a> .
REINSTALLMODE_FILEREPLACE	Force all files to be reinstalled regardless of checksum or version.
REINSTALLMODE_USERDATA	Rewrite all required registry entries from the <a href="#">Registry Table</a> that go to the <b>HKEY_CURRENT_USER</b> or <b>HKEY_USERS</b> registry hive.
REINSTALLMODE_MACHINEDATA	Rewrite all required registry entries from the <a href="#">Registry Table</a> that go to the <b>HKEY_LOCAL_MACHINE</b> or <b>HKEY_CLASSES_ROOT</b> registry hive. Rewrite all information from the <a href="#">Class Table</a> , <a href="#">Verb Table</a> , <a href="#">PublishComponent Table</a> , <a href="#">ProgID Table</a> , <a href="#">MIMET Table</a> , <a href="#">Icon Table</a> , <a href="#">Extension Table</a> and <a href="#">AppID Table</a> regardless of machine or user assignment. Reinstall all <b>qualified components</b> . When reinstalling an application, this option runs the <a href="#">RegisterTypeLibraries</a> and <a href="#">InstallODBC</a> actions.
REINSTALLMODE_SHORTCUT	Reinstall all shortcuts and i

	cache all icons overwriting any existing shortcuts and icons.
REINSTALLMODE_PACKAGE	Use to run from the source package and re-cache the local package. Do not use for the first installation of an application or feature.

## Return Value

Return code	Description
ERROR_INSTALL_FAILURE	The installation failed.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_INSTALL_SERVICE_FAILURE	The installation service could not be accessed.
ERROR_INSTALL_SUSPEND	The installation was suspended and is incomplete.
ERROR_INSTALL_USEREXIT	The user canceled the installation.
ERROR_SUCCESS	The function completed successfully.
ERROR_UNKNOWN_PRODUCT	The product code does not identify a known product.

For more information, see [Displayed Error Messages](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiReinstallProductW</b> (Unicode) and <b>MsiReinstallProductA</b> (ANSI)

## See Also

[Installation and Configuration Functions](#)

[REINSTALLMODE Property](#)

[Multiple Package Installations](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiRemovePatches Function

The **MsiRemovePatches** function removes one or more patches from a single product. To remove a patch from multiple products, **MsiRemovePatches** must be called for each product.

## Syntax

```
C++UINT MsiRemovePatches(  
    __in      LPCTSTR szPatchList,  
    __in      LPCTSTR szProductCode,  
    __in      INSTALLTYPE eUninstallType,  
    __in_opt  LPCTSTR szPropertyList  
);
```

## Parameters

*szPatchList* [in]

A null-terminated string that represents the list of patches to remove. Each patch can be specified by the GUID of the patch or the full path to the patch package. The patches in the list are delimited by semicolons.

*szProductCode* [in]

A null-terminated string that is the **ProductCode** (GUID) of the product from which the patches are removed. This parameter cannot be NULL.

*eUninstallType* [in]

Value that indicates the type of patch removal to perform. This parameter must be `INSTALLTYPE_SINGLE_INSTANCE`.

Value	Meaning
<code>INSTALLTYPE_SINGLE_INSTANCE</code>	The patch is uninstalled for only the product specified by <i>szProduct</i> .

*szPropertyList* [in, optional]

A null-terminated string that specifies command-line property settings. For more information see [About Properties](#) and [Setting Public Property Values on the Command Line](#). This parameter can be NULL.

## Return Value

The **MsiRemovePatches** function returns the following values.

Value	Meaning
ERROR_INVALID_PARAMETER	An invalid parameter was included.
ERROR_PATCH_PACKAGE_OPEN_FAILED	The patch package could not be opened.
ERROR_SUCCESS	The patch was successfully removed.
ERROR_UNKNOWN_PRODUCT	The product specified by <i>szProductList</i> is not installed either per-machine or per-user for the caller of <b>MsiRemovePatches</b> .
ERROR_PATCH_PACKAGE_OPEN_FAILED	The patch package could not be opened.
ERROR_PATCH_PACKAGE_INVALID	The patch package is invalid.
ERROR_PATCH_PACKAGE_UNSUPPORTED	The patch package cannot be processed by this version of the Windows Installer service.
ERROR_PATCH_REMOVAL_UNSUPPORTED	The patch package is not

	removable.
ERROR_UNKNOWN_PATCH	The patch has not been applied to this product.
ERROR_PATCH_REMOVAL_DISALLOWED	Patch removal was disallowed by policy.

## Remarks

**Windows Installer 2.0:** Not supported. The **MsiRemovePatches** function is available beginning with Windows Installer 3.0.

See [Uninstalling Patches](#) for an example that demonstrates how an application can remove a patch from all products that are available to the user.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 and later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiRemovePatchesW</b> (Unicode) and <b>MsiRemovePatchesA</b> (ANSI)

## See Also

### [Removing Patches](#)

[Uninstalling Patches](#)

[About Properties](#)

[Setting Public Property Values on the Command Line](#)

### [MsiApplyPatch](#)

### [ProductCode](#)

[Multiple-Package Installations](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSetExternalUI Function

The **MsiSetExternalUI** function enables an external user-interface handler. This external UI handler is called before the normal internal user-interface handler. The external UI handler has the option to suppress the internal UI by returning a non-zero value to indicate that it has handled the messages. For more information, see [About the User Interface](#).

## Syntax

```
C++INSTALLUI_HANDLER MsiSetExternalUI(  
    __in  INSTALLUI_HANDLER puiHandler,  
    __in  DWORD dwMessageFilter,  
    __in  LPVOID pvContext  
);
```

## Parameters

*puiHandler* [in]

Specifies a callback function that conforms to the **INSTALLUI\_HANDLER** specification.

*dwMessageFilter* [in]

Specifies which messages to handle using the external message handler. If the external handler returns a non-zero result, then that message will not be sent to the UI, instead the message will be logged if logging has been enabled. For more information, see the **MsiEnableLog** function.

Value	Meaning
INSTALLLOGMODE_FILESinUSE	Files in use information. When this message is received, a <a href="#">FilesInUse Dialog</a> should be displayed.
INSTALLLOGMODE_FATALEXIT	Premature termination of

	installation.
INSTALLLOGMODE_ERROR	The error messages are logged.
INSTALLLOGMODE_WARNING	The warning messages are logged.
INSTALLLOGMODE_USER	The user requests are logged.
INSTALLLOGMODE_INFO	The status messages that are not displayed are logged.
INSTALLLOGMODE_RESOLVESOURCE	Request to determine a valid source location.
INSTALLLOGMODE_RMFILESINUSE	Files in use information. When this message is received, a <a href="#">MsiRMFilesInUse Dialog</a> should be displayed.
INSTALLLOGMODE_OUTOFDISKSPACE	There was insufficient disk space.
INSTALLLOGMODE_ACTIONSTART	The start of new installation actions are logged.
INSTALLLOGMODE_ACTIONDATA	The data record with the installation action is logged.
INSTALLLOGMODE_COMMONDATA	The parameters for user-interface initialization are logged.

INSTALLLOGMODE_PROGRESS	<p><i>Progress bar</i> information. This message includes information on units so far and total number of units. For an explanation of the message format, see the <b>MsiProcessMessage</b> function. This message is only sent to an external user interface and is not logged.</p>
INSTALLLOGMODE_INITIALIZE	<p>If this is not a quiet installation, then the <i>basic UI</i> has been initialized. If this is a <i>full UI</i> installation, the full UI is not yet initialized. This message is only sent to an external user interface and is not logged.</p>
INSTALLLOGMODE_TERMINATE	<p>If a full UI is being used, the full UI has ended. If this is not a quiet installation, the basic UI has not yet ended. This message is only sent to an external user interface and is not logged.</p>
INSTALLLOGMODE_SHOWDIALOG	<p>Sent prior to display of the full UI dialog. This message is only sent to an external user</p>

	interface and is not logged.
INSTALLLOGMODE_INSTALLSTART	Installation of product begins. The message contains the product's ProductName and ProductCode.
INSTALLLOGMODE_INSTALLEND	Installation of product ends. The message contains the product's ProductName, ProductCode, and return value.

*pvContext* [in]

Pointer to an application context that is passed to the callback function. This parameter can be used for error checking.

## Return Value

The return value is the previously set external handler, or zero (0) if there was no previously set handler.

## Remarks

To restore the previous UI handler, second call is made to **MsiSetExternalUI** using the **INSTALLUI\_HANDLER** returned by the first call to **MsiSetExternalUI** and specifying zero (0) for dwMessageFilter.

The external user interface handler pointed to by the *puiHandler* parameter does not have full control over the external user interface unless **MsiSetInternalUI** is called with the *dwUILevel* parameter set to **INSTALLUILEVEL\_NONE**. If **MsiSetInternalUI** is not called, the internal

user interface level defaults to `INSTALLUILEVEL_BASIC`. As a result, any message not handled by the external user interface handler is handled by Windows Installer. The initial "Preparing to install. . ." dialog always appears even if the external user interface handler handles all messages.

**MsiSetExternalUI** should only be called from a [Bootstrapping](#) application. You cannot call **MsiSetExternalUI** from a custom action.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiSetExternalUIW</b> (Unicode) and <b>MsiSetExternalUIA</b> (ANSI)

## See Also

[Interface and Logging Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSetExternalUIRecord Function

The [MsiSetExternalUIRecord](#) function enables an external user-interface (UI) handler.

## Syntax

```
C++UINT MsiSetExternalUIRecord(  
    __in      PINSTALLUI_HANDLER_RECORD puiHandler,  
    __in      DWORD dwMessageFilter,  
    __in      LPVOID pvContext,  
    __out_opt PINSTALLUI_HANDLER_RECORD ppuiPrevHandler  
);
```

## Parameters

*puiHandler* [in]

Specifies a callback function that conforms to the [INSTALLUI\\_HANDLER\\_RECORD](#) specification.

To disable the current external UI handler, call the function with this parameter set to a NULL value.

*dwMessageFilter* [in]

Specifies which messages to handle using the external message handler. If the external handler returns a non-zero result, then that message is not sent to the UI, instead the message is logged if logging is enabled. For more information, see [MsiEnableLog](#).

Value	Meaning
INSTALLLOGMODE_FILESinUSE	Files in use information. When this message is received, a <a href="#">FilesInUse Dialog</a> should be displayed.
INSTALLLOGMODE_FATALEXIT	Premature termination of installation.

INSTALLLOGMODE_ERROR	The error messages are logged.
INSTALLLOGMODE_WARNING	The warning messages are logged.
INSTALLLOGMODE_USER	The user requests are logged.
INSTALLLOGMODE_INFO	The status messages that are not displayed are logged.
INSTALLLOGMODE_RESOLVESOURCE	Request to determine a valid source location.
INSTALLLOGMODE_RMFILESINUSE	Files in use information. When this message is received, a <a href="#">MsiRMFilesInUse Dialog</a> should be displayed.
INSTALLLOGMODE_OUTOFDISKSPACE	The is insufficient disk space.
INSTALLLOGMODE_ACTIONSTART	The start of new installation actions are logged.
INSTALLLOGMODE_ACTIONDATA	The data record with the installation action is logged.
INSTALLLOGMODE_COMMONDATA	The parameters for user-interface initialization are logged.
INSTALLLOGMODE_PROGRESS	The <i>Progress bar</i>

	<p>information.</p> <p>This message includes information about units so far and total number of units. This message is only sent to an external user interface and is not logged. For more information, see <a href="#">MsiProcessMessage</a>.</p>
INSTALLLOGMODE_INITIALIZE	<p>If this is not a quiet installation, then the <i>basic UI</i> is initialized.</p> <p>If this is a full UI installation, the <i>Full UI</i> is not yet initialized.</p> <p>This message is only sent to an external user interface and is not logged.</p>
INSTALLLOGMODE_TERMINATE	<p>If a full UI is being used, the full UI has ended.</p> <p>If this is not a quiet installation, the basic UI has not ended.</p> <p>This message is only sent to an external user interface and is not logged.</p>
INSTALLLOGMODE_SHOWDIALOG	<p>Sent prior to display of the Full UI dialog.</p> <p>This message is only sent to an external user</p>

	interface and is not logged.
INSTALLLOGMODE_INSTALLSTART	Installation of product begins.  The message contains the product's ProductName and ProductCode.
INSTALLLOGMODE_INSTALLEND	Installation of product ends.  The message contains the product's ProductName, ProductCode, and return value.

*pvContext* [in]

A pointer to an application context that is passed to the callback function.

This parameter can be used for error checking.

*ppuiPrevHandler* [out, optional]

Returns the pointer to the previously set callback function that conforms to the **INSTALLUI\_HANDLER\_RECORD** specification, or NULL if no callback is previously set.

## Return Value

Return code	Description
ERROR_SUCCESS	The function completes successfully.
ERROR_CALL_NOT_IMPLEMENTED	This value indicates that an attempt is made to call this

function from a custom action.  
This function cannot be called from a custom action.

## Remarks

This function cannot be called from [Custom Actions](#).

The external UI handler enabled by calling [MsiSetExternalUIRecord](#) receives messages in the format of a [Record Object](#). The external UI handler enabled by calling [MsiSetExternalUI](#) receives messages in the format of a string. An external UI is always called before the Windows Installer internal UI. An enabled record-based external UI is called before any string-based external UI. If the record-based external UI handler returns 0 (zero), the message is sent to any enabled string-based external UI handler. If the external UI handler returns a non-zero value, the internal Windows Installer UI handler is suppressed and the messages are considered handled.

This function stores the external user interfaces it has set. To replace the current external UI handler with a previous handler, call the function and specify the **INSTALLUI\_HANDLER\_RECORD** as the *puiHandler* parameter and 0 (zero) as the *dwMessageFilter* parameter.

The external user interface handler pointed to by the *puiHandler* parameter does not have full control over the external user interface unless [MsiSetInternalUI](#) is called with the *dwUILevel* parameter set to **INSTALLUILEVEL\_NONE**. If [MsiSetInternalUI](#) is not called, the internal user interface level defaults to **INSTALLUILEVEL\_BASIC**. As a result, any message not handled by the external user interface handler is handled by Windows Installer. The initial "Preparing to install. . ." dialog always appears even if the external user interface handler handles all messages. [MsiSetExternalUI](#) should only be called from an [Bootstrapping](#) application. You cannot call [MsiSetExternalUI](#) from a custom action.

To disable this external UI handler, call [MsiSetExternalUIRecord](#) with a NULL value for the *puiHandler* parameter.

**Windows Installer 2.0 and Windows Installer 3.0:** Not supported. The [MsiSetExternalUIRecord](#) function is available beginning with Windows Installer 3.1.

For more information about using a record-based external handler, see [Monitoring an Installation Using MsiSetExternalUIRecord](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.1 on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiSetExternalUIRecord</b> (ANSI)

## See Also

[Interface and Logging Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSetInternalUI Function

The [MsiSetInternalUI](#) function enables the installer's internal user interface. Then this user interface is used for all subsequent calls to user-interface-generating installer functions in this process. For more information, see [User Interface Levels](#).

## Syntax

```
C++INSTALLUILEVEL MsiSetInternalUI(  
    __in    INSTALLUILEVEL dwUILevel,  
    __inout HWND *phwnd  
);
```

## Parameters

*dwUILevel* [in]

Specifies the level of complexity of the user interface. This parameter can be one of the following values.

Value	Meaning
INSTALLUILEVEL_FULL	Authored user interface with wizards, progress, and errors.
INSTALLUILEVEL_REDUCED	Authored user interface with wizard dialog boxes suppressed.
INSTALLUILEVEL_BASIC	Simple progress and error handling.
INSTALLUILEVEL_DEFAULT	The installer chooses an appropriate user interface level.
INSTALLUILEVEL_NONE	Completely silent installation.

INSTALLUILEVEL_ENDDIALOG	If combined with any above value, the installer displays a modal dialog box at the end of a successful installation or if there has been an error. No dialog box is displayed if the user cancels.
INSTALLUILEVEL_PROGRESSONLY	If combined with the INSTALLUILEVEL_BASIC value, the installer shows simple progress dialog boxes but does not display any modal dialog boxes or error dialog boxes.
INSTALLUILEVEL_NOCHANGE	No change in the UI level. However, if <i>phWnd</i> is not Null, the parent window can change.
INSTALLUILEVEL_HIDECANCEL	If combined with the INSTALLUILEVEL_BASIC value, the installer shows simple progress dialog boxes but does not display a <b>Cancel</b> button on the dialog. This prevents users from canceling the install.
INSTALLUILEVEL_SOURCERESONLY	If this value is combined with the INSTALLUILEVEL_NONE value, the installer displays only the dialog boxes used for source resolution. No other dialog boxes are shown. This value has no effect if the UI level is not

INSTALLUILEVEL\_NONE.

It is used with an external user interface designed to handle all of the UI except for source resolution. In this case, the installer handles source resolution.

*phWnd* [in, out]

Pointer to a window. This window becomes the owner of any user interface created. A pointer to the previous owner of the user interface is returned. If this parameter is null, the owner of the user interface does not change.

## Return Value

The previous user interface level is returned. If an invalid *dwUILevel* is passed, then `INSTALLUILEVEL_NOCHANGE` is returned.

## Remarks

The [MsiSetInternalUI](#) function is useful when the installer must display a user interface. For example, if a feature is installed, but the source is a compact disc that must be inserted, the installer prompts the user for the compact disc. Depending on the nature of the installation, the application might also display progress indicators or query the user for information.

When `Msi.dll` is loaded, the user interface level is set to `DEFAULT` and the user interface owner is set to 0 (that is, the initial user interface owner is the desktop).

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows

<b>Version</b>	Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[Interface and Logging Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSourceListAddMediaDisk Function

The **MsiSourceListAddMediaDisk** function adds or updates a disk of the media source of a registered product or patch. If the disk specified already exists, it is updated with the new values. If the disk specified does not exist, a new disk entry is created with the new values.

## Syntax

```
C++UINT MsiSourceListAddMediaDisk(  
    __in      LPCTSTR szProductCodeOrPatchCode,  
    __in_opt  LPCTSTR szUserSid,  
    __in      MSIINSTALLCONTEXT dwContext,  
    __in      DWORD dwOptions,  
    __in      DWORD dwDiskId,  
    __in      LPCTSTR szVolumeLabel,  
    __in_opt  LPCTSTR szDiskPrompt  
);
```

## Parameters

*szProductCodeOrPatchCode* [in]

The **ProductCode** or patch GUID of the product or patch. Use a null-terminated string. If the string is longer than 39 characters, the function fails and returns `ERROR_INVALID_PARAMETER`. This parameter cannot be `NULL`.

*szUserSid* [in, optional]

This parameter can be a string SID that specifies the user account that contains the product or patch. The SID is not validated or resolved. An incorrect SID can return `ERROR_UNKNOWN_PRODUCT` or `ERROR_UNKNOWN_PATCH`.

Type of SID	Meaning
NULL	NULL denotes the currently logged on user. When referencing the current user account, <i>szUserSID</i> can be NULL and <i>dwContext</i> can be

	MSIINSTALLCONTEXT_USERMANAGED or MSIINSTALLCONTEXT_USERUNMANAGED.
User SID	Specifies enumeration for a particular user in the system. An example of user SID is "S-1-3-64-2415071341-1358098788-3127455600-2561".

**Note** The special SID string s-1-5-18 (system) cannot be used to enumerate products or patches installed as per-machine. Setting the SID value to s-1-5-18 returns ERROR\_INVALID\_PARAMETER. When *dwContext* is set to MSIINSTALLCONTEXT\_MACHINE only, *szUserSid* must be null.

**Note** The special SID string s-1-1-0 (everyone) should not be used. Setting the SID value to s-1-1-0 fails and returns ERROR\_INVALID\_PARAM .

*dwContext* [in]

This parameter specifies the context of the product or patch instance. This parameter can contain one of the following values.

Type of context	Meaning
MSIINSTALLCONTEXT_USERMANAGED	The product or patch instance exists in the per-user-managed context.
MSIINSTALLCONTEXT_USERUNMANAGED	The product or patch instance exists in the per-user-unmanaged context.
MSIINSTALLCONTEXT_MACHINE	The product or patch instance exists in the per-machine context.

*dwOptions* [in]

The *dwOptions* value specifies the meaning of *szProductCodeOrPatchCode*.

Flag	Meaning
MSICODE_PRODUCT	<i>szProductCodeOrPatchCode</i> is a product code GUID.
MSICODE_PATCH	<i>szProductCodeOrPatchCode</i> is a patch code GUID.

*dwDiskId* [in]

This parameter provides the ID of the disk being added or updated.

*szVolumeLabel* [in]

The *szVolumeLabel* provides the label of the disk being added or updated. An update overwrites the existing volume label in the registry. To change the disk prompt only, get the existing volume label from the registry and provide it in this call along with the new disk prompt. Passing a null or empty string for *szVolumeLabel* registers an empty string (0 bytes in length) as the volume label.

*szDiskPrompt* [in, optional]

On entry to **MsiSourceListAddMediaDisk**, *szDiskPrompt* provides the disk prompt of the disk being added or updated. An update overwrites the registered disk prompt. To change the volume label only, get the existing disk prompt that is registered and provide it when calling **MsiSourceListAddMediaDisk** along with the new volume label. Passing NULL or an empty string registers an empty string (0 bytes in length) as the disk prompt.

## Return Value

The **MsiSourceListAddMediaDisk** function returns the following values.

Value	Meaning
ERROR_ACCESS_DENIED	The user does not have the

	ability to read the specified media source or the specified product or patch. This does not indicate whether a media source, product or patch was found.
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INSTALL_SERVICE_FAILURE	The Windows Installer service could not be accessed.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_SUCCESS	The value was successfully reordered.
ERROR_UNKNOWN_PATCH	The patch was not found.
ERROR_UNKNOWN_PRODUCT	The product was not found.
ERROR_FUNCTION_FAILED	Unexpected internal failure.

## Remarks

Administrators can modify the installation of a product or patch instance that exists under the machine context or under their own per-user context (managed or unmanaged.) They can modify the installation of a product or patch instance that exists under any user's per-user-managed context. Administrators cannot modify another user's installation of a product or patch instance that exists under that other user's per-user-unmanaged context.

Non-administrators cannot modify the installation of a product or patch instance that exists under another user's per-user context (managed or unmanaged.) They can modify the installation of a product or patch instance that exists under their own per-user-unmanaged context. They

can modify the installation of a product or patch instance under the machine context or their own per-user-managed context only if they are enabled to browse for a product or patch source. Users can be enabled to browse for sources by setting policy. For more information, see [DisableBrowse](#), [AllowLockdownBrowse](#), [AllowLockDownMedia](#) and [AlwaysInstallElevated](#) policies.

**Windows Installer 2.0:** Not supported. The **MsiSourceListAddMediaDisk** function is available beginning with Windows Installer 3.0.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiSourceListAddMediaDiskW</b> (Unicode) and <b>MsiSourceListAddMediaDiskA</b> (ANSI)

## See Also

### [ProductCode](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSourceListAddSource Function

The **MsiSourceListAddSource** function adds to the list of valid network sources that contain the specified type of sources for a product or patch in a specified user context.

The number of sources in the **SOURCELIST** property is unlimited.

## Syntax

```
C++UINT MsiSourceListAddSource(  
    __in LPCTSTR szProduct,  
    __in LPCTSTR szUserName,  
    __in DWORD dwReserved,  
    __in LPCTSTR szSource  
);
```

## Parameters

*szProduct* [in]

The **ProductCode** of the product to modify.

*szUserName* [in]

The user name for a per-user installation. On Windows 2000 or Windows XP, the user name should always be in the format of DOMAIN\USERNAME (or MACHINENAME\USERNAME for a local user).

An empty string or null for a per-machine installation.

*dwReserved* [in]

Reserved for future use. This value must be set to 0.

*szSource* [in]

Pointer to the string specifying the source.

## Return Value

Return code	Description
ERROR_ACCESS_DENIED	The user does not have the

	ability to add a source.
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_BAD_USERNAME	Could not resolve the user name.
ERROR_FUNCTION_FAILED	The function did not succeed.
ERROR_INSTALL_SERVICE_FAILURE	Could not access installer service.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_SUCCESS	The source was added.
ERROR_UNKNOWN_PRODUCT	The specified product is unknown.

## Remarks

An administrator can modify per-machine installations, their own per-user non-managed installations, and the per-user managed installations for any user. A non-administrator can only modify per-machine installations and their own (managed or non-managed)per-user installations. Users can be enabled to browse for sources by setting policy. For more information, see the [DisableBrowse](#), [AllowLockdownBrowse](#), and [AlwaysInstallElevated](#) policies.

Note that this function merely adds the new source to the list of valid sources. If another source was used to install the product, the new source is not used until the current source is unavailable.

It is the responsibility of the caller to ensure that the provided source is a valid source image for the product.

If the user name is an empty string or NULL, the function operates on the

per-machine installation of the product. In this case, if the product is installed only in the per-user state, the function returns `ERROR_UNKNOWN_PRODUCT`.

If the user name is not an empty string or `NULL`, it specifies the name of the user whose product installation is modified. If the user name is the current user name, the function first attempts to modify a non-managed installation of the product. If no non-managed installation of the product can be found, the function then tries to modify a managed per-user installation of the product. If no managed or unmanaged per-user installations of the product can be found, the function returns `ERROR_UNKNOWN_PRODUCT`, even if the product is installed per-machine.

This function can not modify a non-managed installation for any user besides the current user. If the user name is not an empty string or `NULL`, but is not the current user, the function only checks for a managed per-user installation of the product for the specified user. If the product is not installed as managed per-user for the specified user, the function returns `ERROR_UNKNOWN_PRODUCT`, even if the product is installed per-machine.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiSourceListAddSourceW</b> (Unicode) and <b>MsiSourceListAddSourceA</b> (ANSI)

---

## See Also

[AllowLockdownBrowse](#)

[AlwaysInstallElevated](#)

[DisableBrowse](#)

[Source Resiliency](#)

[Installation Context](#)

[ProductCode](#)

[SOURCELIST](#)

[LookupAccountName](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSourceListAddSourceEx Function

The **MsiSourceListAddSourceEx** function adds or reorders the set of sources of a patch or product in a specified context. It can also create a source list for a patch that does not exist in the specified context.

## Syntax

```
C++UINT MsiSourceListAddSourceEx(  
    __in      LPCTSTR szProductCodeOrPatchCode,  
    __in_opt  LPCTSTR szUserSid,  
    __in      MSIINSTALLCONTEXT dwContext,  
    __in      DWORD dwOptions,  
    __in      LPCTSTR szSource,  
    __in_opt  DWORD dwIndex  
);
```

## Parameters

*szProductCodeOrPatchCode* [in]

The **ProductCode** or patch GUID of the product or patch. Use a null-terminated string. If the string is longer than 39 characters, the function fails and returns `ERROR_INVALID_PARAMETER`. This parameter cannot be `NULL`.

*szUserSid* [in, optional]

This parameter can be a string SID that specifies the user account that contains the product or patch. The SID is not validated or resolved. An incorrect SID can return `ERROR_UNKNOWN_PRODUCT` or `ERROR_UNKNOWN_PATCH`. When referencing a machine context, *szUserSID* must be `NULL` and *dwContext* must be `MSIINSTALLCONTEXT_MACHINE`.

Type of SID	Meaning
NULL	NULL denotes the currently logged on user. When referencing the current user account, <i>szUserSID</i>

	can be NULL and <i>dwContext</i> can be MSIINSTALLCONTEXT_USERMANAGED or MSIINSTALLCONTEXT_USERUNMANAGED.
User SID	Specifies enumeration for a particular user in the system. An example of a user SID is "S-1-3-64-2415071341-1358098788-3127455600-2561".

**Note** The special SID string s-1-5-18 (system) cannot be used to enumerate products or patches installed as per-machine. Setting the SID value to s-1-5-18 returns ERROR\_INVALID\_PARAMETER.

**Note** The special SID string s-1-1-0 (everyone) should not be used. Setting the SID value to s-1-1-0 fails and returns ERROR\_INVALID\_PARAM .

*dwContext* [in]

This parameter specifies the context of the product or patch instance. This parameter can contain one of the following values.

Type of context	Meaning
MSIINSTALLCONTEXT_USERMANAGED	The product or patch instance exists in the per-user-managed context.
MSIINSTALLCONTEXT_USERUNMANAGED	The product or patch instance exists in the per-user-unmanaged context.
MSIINSTALLCONTEXT_MACHINE	The product or patch instance exists in the per-machine context.

*dwOptions* [in]

The *dwOptions* value determines the interpretation of the *szProductCodeOrPatchCode* value and the type of sources to clear. This parameter must be a combination of one of the following MSISOURCETYPE\_\* constants and one of the following MSICODE\_\* constants.

Flag	Meaning
MSISOURCETYPE_NETWORK	The source is a network type.
MSISOURCETYPE_URL	The source is a URL type.
MSICODE_PRODUCT	<i>szProductCodeOrPatchCode</i> is a product code.
MSICODE_PATCH	<i>szProductCodeOrPatchCode</i> is a patch code.

*szSource* [in]

Source to add or move. This parameter is expected to contain only the path without the filename. The filename is already registered as "PackageName" and can be manipulated through [MsiSourceListSetInfo](#). This argument is required.

*dwIndex* [in, optional]

This parameter provides the new index for the source. All sources are indexed in the source list from 1 to *N*, where *N* is the count of sources in the list. Every source in the list has a unique index.

If **MsiSourceListAddSourceEx** is called with a new source and *dwIndex* set to 0 (zero), the new source is appended to the existing list. If *dwIndex* is set to 0 and the source already exists in the list, no update is done on the list.

If **MsiSourceListAddSourceEx** is called with a new source and *dwIndex* set to a non-zero value less than count (*N*), the new source is placed at the specified index and the other sources are re-indexed. If the source already exists, it is moved to the specified index and the other sources are re-indexed.

If **MsiSourceListAddSourceEx** is called with a new source and *dwIndex* set to a non-zero value greater than the count of sources (*N*), the new source is appended to the existing list. If the source already exists, it is moved to the end of the list and the other sources are re-indexed.

## Return Value

The **MsiSourceListAddSourceEx** function returns the following values.

Value	Meaning
ERROR_ACCESS_DENIED	The user does not have the ability to add or move a source. Does not indicate whether the product or patch was found.
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INSTALL_SERVICE_FAILURE	Could not access the Windows Installer service.
ERROR_SUCCESS	The source was inserted or updated.
ERROR_UNKNOWN_PRODUCT	The specified product is unknown.
ERROR_UNKNOWN_PATCH	The specified patch is unknown.
ERROR_FUNCTION_FAILED	Unexpected internal failure.

## Remarks

Administrators can modify the installation of a product or patch instance that exists under the machine context or under their own per-user context (managed or unmanaged.) They can modify the installation of a product

or patch instance that exists under any user's per-user-managed context. Administrators cannot modify another user's installation of a product or patch instance that exists under that other user's per-user-unmanaged context.

Non-administrators cannot modify the installation of a product or patch instance that exists under another user's per-user context (managed or unmanaged.) They can modify the installation of a product or patch instance that exists under their own per-user-unmanaged context. They can modify the installation of a product or patch instance under the machine context or their own per-user-managed context only if they are enabled to browse for a product or patch source. Users can be enabled to browse for sources by setting policy. For more information, see the [DisableBrowse](#), [AllowLockdownBrowse](#), and [AlwaysInstallElevated](#) policies.

**Windows Installer 2.0:** Not supported. The **MsiSourceListAddSourceEx** function is available beginning with Windows Installer 3.0.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiSourceListAddSourceExW</b> (Unicode) and <b>MsiSourceListAddSourceExA</b> (ANSI)

## See Also

[AllowLockdownBrowse](#)

[AlwaysInstallElevated](#)

[DisableBrowse](#)

[ProductCode](#)

[Source Resiliency](#)

[MsiSourceListSetInfo](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSourceListClearAll Function

The **MsiSourceListClearAll** function removes all network sources from the source list of a patch or product in a specified context. For more information, see [Source Resiliency](#).

## Syntax

```
C++UINT MsiSourceListClearAll(  
    __in LPCTSTR szProduct,  
    __in LPCTSTR szUserName,  
    __in DWORD dwReserved  
);
```

## Parameters

*szProduct* [in]

The **ProductCode** of the product to modify.

*szUserName* [in]

The user name for a per-user installation. The user name should always be in the format of DOMAIN\USERNAME (or MACHINENAME\USERNAME for a local user).

An empty string or null for a per-machine installation.

*dwReserved* [in]

Reserved for future use. This value must be set to 0.

## Return Value

The **MsiSourceListClearAll** function returns the following values.

Value	Meaning
ERROR_ACCESS_DENIED	The user does not have the ability to clear the source list for this product.
ERROR_BAD_CONFIGURATION	The configuration data is

	corrupt.
ERROR_BAD_USERNAME	Could not resolve the user name.
ERROR_FUNCTION_FAILED	The function did not succeed.
ERROR_INSTALL_SERVICE_FAILURE	Could not access installer service.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_SUCCESS	The function succeeded.
ERROR_UNKNOWN_PRODUCT	The specified product is unknown.

## Remarks

An administrator can modify per-machine installations, their own per-user non-managed installations, and the per-user managed installations for any user. A non-administrator can only modify per-machine installations and their own (managed or non-managed) per-user installations. Users can be enabled to browse for sources by setting policy. For more information, see the [DisableBrowse](#), [AllowLockdownBrowse](#), and [AlwaysInstallElevated](#) policies.

If a network source is the current source for the product, this function forces the installer to search the source list for a valid source the next time a source is needed. If the current source is media or a URL source, it is still valid after this call and the source list is not searched unless [MsiSourceListForceResolution](#) is also called.

If the user name is an empty string or NULL, the function operates on the per-machine installation of the product. In this case, if the product is installed as per-user only, the function returns ERROR\_UNKNOWN\_PRODUCT.

If the user name is not an empty string or NULL, it specifies the name of the user whose product installation is modified. If the user name is the current user name, the function first attempts to modify a non-managed installation of the product. If no non-managed installation of the product can be found, the function then tries to modify a managed per-user installation of the product. If no managed or unmanaged per-user installations of the product can be found, the function returns `ERROR_UNKNOWN_PRODUCT`, even if the product is installed per-machine.

This function cannot modify a non-managed installation for any user besides the current user. If the user name is not an empty string or NULL, but is not the current user, the function only checks for a managed per-user installation of the product for the specified user. If the product is not installed as managed per-user for the specified user, the function returns `ERROR_UNKNOWN_PRODUCT`, even if the product is installed per-machine.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiSourceListClearAllW</b> (Unicode) and <b>MsiSourceListClearAllA</b> (ANSI)

## See Also

[AllowLockdownBrowse](#)

[AlwaysInstallElevated](#)

[DisableBrowse](#)

[Installation Context](#)

[ProductCode](#)

[Source Resiliency](#)

[LookupAccountName](#)

[MsiSourceListForceResolution](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSourceListClearAllEx Function

The **MsiSourceListClearAllEx** function removes all the existing sources of a given source type for the specified product or patch instance. The patch registration is also removed if the sole source of the patch gets removed and if the patch is not installed as a new patch by any client in the same context. Specifying that **MsiSourceListClearAllEx** remove the current source for this product or patch forces the installer to search the source list for a source the next time a source is required.

## Syntax

```
C++UINT MsiSourceListClearAllEx(  
    __in      LPCTSTR szProductCodeOrPatchCode,  
    __in_opt  LPCTSTR szUserSid,  
    __in      MSIINSTALLCONTEXT dwContext,  
    __in      DWORD dwOptions  
);
```

## Parameters

*szProductCodeOrPatchCode* [in]

The **ProductCode** or patch GUID of the product or patch. Use a null-terminated string. If the string is longer than 39 characters, the function fails and returns `ERROR_INVALID_PARAMETER`. This parameter cannot be `NULL`.

*szUserSid* [in, optional]

This parameter can be a string SID that specifies the user account that contains the product or patch. The SID is not validated or resolved. An incorrect SID can return `ERROR_UNKNOWN_PRODUCT` or `ERROR_UNKNOWN_PATCH`. When referencing a machine context, *szUserSID* must be `NULL` and *dwContext* must be `MSIINSTALLCONTEXT_MACHINE`. Using the machine SID ("S-1-5-18") returns `ERROR_INVALID_PARAMETER`. When referencing the current user account, *szUserSID* can be `NULL` and *dwContext* can be `MSIINSTALLCONTEXT_USERMANAGED` or `MSIINSTALLCONTEXT_USERUNMANAGED`.

*dwContext* [in]

This parameter specifies the context of the product or patch instance. This parameter can contain one of the following values.

Type of context	Meaning
MSIINSTALLCONTEXT_USERMANAGED	The product or patch instance exists in the per-user-managed context.
MSIINSTALLCONTEXT_USERUNMANAGED	The product or patch instance exists in the per-user-unmanaged context.
MSIINSTALLCONTEXT_MACHINE	The product or patch instance exists in the per-machine context.

*dwOptions* [in]

The *dwOptions* value determines the interpretation of the *szProductCodeOrPatchCode* value and the type of sources to clear. This parameter must be a combination of one of the following *MSISOURCETYPE\_\** constants and one of the following *MSICODE\_\** constants.

Flag	Meaning
MSISOURCETYPE_MEDIA	The source is media.
MSISOURCETYPE_NETWORK	The source is a network type.
MSISOURCETYPE_URL	The source is a URL type.
MSICODE_PATCH	<i>szProductCodeOrPatchCode</i> is a

	patch code.
MSICODE_PRODUCT	<i>szProductCodeOrPatchCode</i> is a product code.

## Return Value

Value	Meaning
ERROR_ACCESS_DENIED	The user does not have the ability to add or move a source. Does not indicate whether the product or patch was found.
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INSTALL_SERVICE_FAILURE	Cannot access the Windows Installer service.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_SUCCESS	All sources of the specified type were removed.
ERROR_UNKNOWN_PRODUCT	The specified product is unknown.
ERROR_UNKNOWN_PATCH	The specified patch is unknown.
ERROR_FUNCTION_FAILED	Unexpected internal failure.

## Remarks

Administrators can modify the installation of a product or patch instance

that exists under the machine context or under their own per-user context (managed or unmanaged.) They can modify the installation of a product or patch instance that exists under any user's per-user-managed context. Administrators cannot modify another user's installation of a product or patch instance that exists under that other user's per-user-unmanaged context.

Non-administrators cannot modify the installation of a product or patch instance that exists under another user's per-user context (managed or unmanaged.) They can modify the installation of a product or patch instance that exists under their own per-user-unmanaged context. They can modify the installation of a product or patch instance under the machine context or their own per-user-managed context only if they are enabled to browse for a product or patch source. Users can be enabled to browse for sources by setting policy, for more information, see [DisableBrowse](#), [AllowLockdownBrowse](#), and [AlwaysInstallElevated](#) policies.

**Windows Installer 2.0:** Not supported. The **MsiSourceListClearAllEx** function is available beginning with Windows Installer 3.0.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI</b>	<b>MsiSourceListClearAllExW</b> (Unicode) and

**names**

**MsiSourceListClearAllExA (ANSI)**

## See Also

[AllowLockdownBrowse](#)

[AlwaysInstallElevated](#)

[DisableBrowse](#)

**[ProductCode](#)**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSourceListClearSource Function

The **MsiSourceListClearSource** function removes an existing source for a product or patch in a specified context. The patch registration is also removed if the sole source of the patch gets removed and if the patch is not installed by any client in the same context. Specifying that **MsiSourceListClearSource** remove the current source for this product or patch forces the installer to search the source list for a source the next time a source is required.

## Syntax

```
C++UINT MsiSourceListClearSource(  
    __in      LPCTSTR szProductCodeOrPatchCode,  
    __in_opt  LPCTSTR szUserSid,  
    __in      MSIINSTALLCONTEXT dwContext,  
    __in      DWORD dwOptions,  
    __in      LPCTSTR szSource  
);
```

## Parameters

*szProductCodeOrPatchCode* [in]

The **ProductCode** or patch GUID of the product or patch. Use a null-terminated string. If the string is longer than 39 characters, the function fails and returns `ERROR_INVALID_PARAMETER`. This parameter cannot be `NULL`.

*szUserSid* [in, optional]

This parameter can be a string SID that specifies the user account that contains the product or patch. The SID is not validated or resolved. An incorrect SID can return `ERROR_UNKNOWN_PRODUCT` or `ERROR_UNKNOWN_PATCH`. When referencing a machine context, *szUserSID* must be `NULL` and *dwContext* must be `MSIINSTALLCONTEXT_MACHINE`.

Type of SID	Meaning
NULL	NULL denotes the currently logged on user. When

	referencing the current user account, <i>szUserSID</i> can be NULL and <i>dwContext</i> can be MSIINSTALLCONTEXT_USERMANAGED or MSIINSTALLCONTEXT_USERUNMANAGED.
User SID	Specifies enumeration for a particular user in the system. An example of a user SID is "S-1-3-64-2415071341-1358098788-3127455600-2561".

**Note** The special SID string s-1-5-18 (system) cannot be used to enumerate products or patches installed as per-machine. Setting the SID value to s-1-5-18 returns ERROR\_INVALID\_PARAMETER.

**Note** The special SID string s-1-1-0 (everyone) should not be used. Setting the SID value to s-1-1-0 fails and returns ERROR\_INVALID\_PARAM .

*dwContext* [in]

This parameter specifies the context of the product or patch instance. This parameter can contain one of the following values.

Type of context	Meaning
MSIINSTALLCONTEXT_USERMANAGED	The product or patch instance exists in the per-user-managed context.
MSIINSTALLCONTEXT_USERUNMANAGED	The product or patch instance exists in the per-user-unmanaged context.
MSIINSTALLCONTEXT_MACHINE	The product or patch instance exists in the per-machine context.

*dwOptions* [in]

The *dwOptions* value determines the interpretation of the *szProductCodeOrPatchCode* value and the type of sources to clear. This parameter must be a combination of one of the following MSISOURCETYPE\_\* constants and one of the following MSICODE\_\* constants.

Flag	Meaning
MSISOURCETYPE_NETWORK	The source is a network type.
MSISOURCETYPE_URL	The source is a URL type.
MSICODE_PRODUCT	<i>szProductCodeOrPatchCode</i> is a product code.
MSICODE_PATCH	<i>szProductCodeOrPatchCode</i> is a patch code.

*szSource* [in]

Source to remove. This parameter is expected to contain only the path without the filename. The filename is already registered as "PackageName" and can be manipulated through [MsiSourceListSetInfo](#). This argument is required.

## Return Value

The **MsiSourceListClearSource** function returns the following values.

Value	Meaning
ERROR_ACCESS_DENIED	The user does not have the ability to remove a source. Does not indicate whether the product or patch was found.
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.

ERROR_INSTALL_SERVICE_FAILURE	Could not access the Windows Installer service
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_SUCCESS	The source was removed or not found.
ERROR_UNKNOWN_PATCH	The specified patch is unknown.
ERROR_UNKNOWN_PRODUCT	The specified product is unknown.
ERROR_FUNCTION_FAILED	Unexpected internal failure.

## Remarks

Administrators can modify the installation of a product or patch instance that exists under the machine context or under their own per-user context (managed or unmanaged.) They can modify the installation of a product or patch instance that exists under any user's per-user-managed context. Administrators cannot modify another user's installation of a product or patch instance that exists under that other user's per-user-unmanaged context.

Non-administrators cannot modify the installation of a product or patch instance that exists under another user's per-user context (managed or unmanaged.) They can modify the installation of a product or patch instance that exists under their own per-user-unmanaged context. They can modify the installation of a product or patch instance under the machine context or their own per-user-managed context only if they are enabled to browse for a product or patch source. Users can be enabled to browse for sources by setting policy. For more information, see the [DisableBrowse](#), [AllowLockdownBrowse](#), and [AlwaysInstallElevated](#) policies.

**Windows Installer 2.0:** Not supported. The **MsiSourceListClearSource** function is available beginning with Windows Installer 3.0.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiSourceListClearSourceW</b> (Unicode) and <b>MsiSourceListClearSourceA</b> (ANSI)

## See Also

[AllowLockdownBrowse](#)

[AlwaysInstallElevated](#)

[DisableBrowse](#)

[ProductCode](#)

[Source Resiliency](#)

[MsiSourceListSetInfo](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSourceListClearMediaDisk Function

The **MsiSourceListClearMediaDisk** function provides the ability to remove an existing registered disk under the media source for a product or patch in a specific context.

## Syntax

```
C++UINT MsiSourceListClearMediaDisk(  
    __in      LPCTSTR szProductCodeOrPatchCode,  
    __in_opt  LPCTSTR szUserSid,  
    __in      MSIINSTALLCONTEXT dwContext,  
    __in      DWORD dwOptions,  
    __in      DWORD dwDiskID  
);
```

## Parameters

*szProductCodeOrPatchCode* [in]

The **ProductCode** or patch GUID of the product or patch. Use a null-terminated string. If the string is longer than 39 characters, the function fails and returns `ERROR_INVALID_PARAMETER`. This parameter cannot be `NULL`.

*szUserSid* [in, optional]

This parameter can be a string SID that specifies the user account that contains the product or patch. The SID is not validated or resolved. An incorrect SID can return `ERROR_UNKNOWN_PRODUCT` or `ERROR_UNKNOWN_PATCH`.

Type of SID	Meaning
NULL	NULL denotes the currently logged on user. When referencing the current user account, <i>szUserSID</i> can be NULL and <i>dwContext</i> can be <code>MSIINSTALLCONTEXT_USERMANAGED</code> or <code>MSIINSTALLCONTEXT_USERUNMANAGED</code> .

User SID	Specifies enumeration for a particular user in the system. An example of user SID is "S-1-3-64-2415071341-1358098788-3127455600-2561".
----------	--

**Note** The special SID string s-1-5-18 (system) cannot be used to enumerate products or patches installed as per-machine. Setting the SID value to s-1-5-18 returns `ERROR_INVALID_PARAMETER`. When *dwContext* is set to `MSIINSTALLCONTEXT_MACHINE` only, *szUserSid* must be null.

**Note** The special SID string s-1-1-0 (everyone) should not be used. Setting the SID value to s-1-1-0 fails and returns `ERROR_INVALID_PARAM`.

*dwContext* [in]

This parameter specifies the context of the product or patch instance. This parameter can contain one of the following values.

Type of context	Meaning
<code>MSIINSTALLCONTEXT_USERMANAGED</code>	The product or patch instance exists in the per-user-managed context.
<code>MSIINSTALLCONTEXT_USERUNMANAGED</code>	The product or patch instance exists in the per-user-unmanaged context.
<code>MSIINSTALLCONTEXT_MACHINE</code>	The product or patch instance exists in the per-machine context.

*dwOptions* [in]

The *dwOptions* value specifies the meaning of *szProductCodeOrPatchCode*.

Flag	Meaning
MSICODE_PRODUCT	<i>szProductCodeOrPatchCode</i> is a product code GUID.
MSICODE_PATCH	<i>szProductCodeOrPatchCode</i> is a patch code GUID.

*dwDiskID* [in]

This parameter provides the ID of the disk being removed.

## Return Value

The **MsiSourceListClearMediaDisk** function returns the following values.

Value	Meaning
ERROR_ACCESS_DENIED	The user does not have the ability to read the specified media source or the specified product or patch. This does not indicate whether a media source, product or patch was found.
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INSTALL_SERVICE_FAILURE	The Windows Installer service could not be accessed.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_SUCCESS	The value was successfully removed or not found.

ERROR_UNKNOWN_PATCH	The patch was not found.
ERROR_UNKNOWN_PRODUCT	The product was not found.
ERROR_FUNCTION_FAILED	Unexpected internal failure.

## Remarks

Administrators can modify the installation of a product or patch instance that exists under the machine context or under their own per-user context (managed or unmanaged.) They can modify the installation of a product or patch instance that exists under any user's per-user-managed context. Administrators cannot modify another user's installation of a product or patch instance that exists under that other user's per-user-unmanaged context.

Non-administrators cannot modify the installation of a product or patch instance that exists under another user's per-user context (managed or unmanaged.) They can modify the installation of a product or patch instance that exists under their own per-user-unmanaged context. They can modify the installation of a product or patch instance under the machine context or their own per-user-managed context only if they are enabled to browse for a product or patch source. Users can be enabled to browse for sources by setting policy. For more information, see the [DisableBrowse](#), [AllowLockdownBrowse](#), and [AlwaysInstallElevated](#) policies.

**Windows Installer 2.0:** Not supported. The **MsiSourceListClearMediaDisk** function is available beginning with Windows Installer 3.0.

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or

<b>Version</b>	Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiSourceListClearMediaDiskW</b> (Unicode) and <b>MsiSourceListClearMediaDiskA</b> (ANSI)

## See Also

### [ProductCode](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSourceListEnumSources Function

The **MsiSourceListEnumSources** function enumerates the sources in the source list of a specified patch or product.

## Syntax

```
C++UINT MsiSourceListEnumSources(  
    __in          LPCTSTR szProductCodeOrPatchCode,  
    __in_opt     LPCTSTR szUserSid,  
    __in         MSIINSTALLCONTEXT dwContext,  
    __in         DWORD dwOptions,  
    __in         DWORD dwIndex,  
    __in_opt     LPTSTR szSource,  
    __inout_opt  LPDWORD pcchSource  
);
```

## Parameters

*szProductCodeOrPatchCode* [in]

The **ProductCode** or patch GUID of the product or patch. Use a null-terminated string. If the string is longer than 39 characters, the function fails and returns `ERROR_INVALID_PARAMETER`. This parameter cannot be `NULL`.

*szUserSid* [in, optional]

A string SID that specifies the user account that contains the product or patch. The SID is not validated or resolved. An incorrect SID can return `ERROR_UNKNOWN_PRODUCT` or `ERROR_UNKNOWN_PATCH`. When referencing a machine context, *szUserSID* must be `NULL` and *dwContext* must be `MSIINSTALLCONTEXT_MACHINE`.

Type of SID	Meaning
NULL	A NULL indicates the current user who is logged on. When referencing the current user account, <i>szUserSID</i> can be <code>NULL</code> and <i>dwContext</i> can be

	MSIINSTALLCONTEXT_USERMANAGED or MSIINSTALLCONTEXT_USERUNMANAGED.
User SID	An enumeration for a specific user in the system. An example of a user SID is "S-1-3-64-2415071341-1358098788-3127455600-2561".
s-1-1-0	The special SID string s-1-1-0 (everyone) specifies enumeration across all users in the system.

**Note** The special SID string s-1-5-18 (system) cannot be used to enumerate products or patches installed as per-machine. Setting the SID value to s-1-5-18 returns ERROR\_INVALID\_PARAMETER.

*dwContext* [in]

The context of the product or patch instance. This parameter can contain one of the following values.

Type of context	Meaning
MSIINSTALLCONTEXT_USERMANAGED	The product or patch instance exists in the per-user-managed context.
MSIINSTALLCONTEXT_USERUNMANAGED	The product or patch instance exists in the per-user-unmanaged context.
MSIINSTALLCONTEXT_MACHINE	The product or patch instance exists in the per-machine context.

*dwOptions* [in]

The *dwOptions* value determines the interpretation of the *szProductCodeOrPatchCode* value and the type of sources to clear. This parameter must be a combination of one of the following **MSISOURCETYPE\_\*** constants and one of the following **MSICODE\_\*** constants.

<b>Flag</b>	<b>Meaning</b>
<b>MSISOURCETYPE_NETWORK</b>	The source is a network type.
<b>MSISOURCETYPE_URL</b>	The source is a URL type.
<b>MSICODE_PRODUCT</b>	<i>szProductCodeOrPatchCode</i> is a product code.
<b>MSICODE_PATCH</b>	<i>szProductCodeOrPatchCode</i> is a patch code.

*dwIndex* [in]

The index of the source to retrieve. This parameter must be 0 (zero) for the first call to the **MsiSourceListEnumSources** function, and then incremented for subsequent calls until the function returns **ERROR\_NO\_MORE\_ITEMS**. The index should be incremented only if the previous call returned **ERROR\_SUCCESS**.

*szSource* [in, optional]

A pointer to a buffer that receives the path to the source that is being enumerated. This buffer should be large enough to contain the received value. If the buffer is too small, the function returns **ERROR\_MORE\_DATA** and sets *\*pcchSource* to the number of **TCHAR** in the value, not including the terminating NULL character.

If *szSource* is set to NULL and *pcchSource* is set to a valid pointer, the function returns **ERROR\_SUCCESS** and sets *\*pcchSource* to the number of **TCHAR** in the value, not including the terminating NULL character. The function can then be called again to retrieve the value, with *szSource* buffer large enough to contain *\*pcchSource* + 1 characters.

If *szSource* and *pcchSource* are both set to NULL, the function returns `ERROR_SUCCESS` if the value exists, without retrieving the value.

*pcchSource* [in, out, optional]

A pointer to a variable that specifies the number of **TCHAR** in the *szSource* buffer. When the function returns, this parameter is set to the size of the requested value whether or not the function copies the value into the specified buffer. The size is returned as the number of **TCHAR** in the requested value, not including the terminating null character.

This parameter can be set to NULL only if *szSource* is also NULL, otherwise the function returns `ERROR_INVALID_PARAMETER`.

## Return Value

The `MsiSourceListEnumSources` function returns the following values.

Value	Meaning
<code>ERROR_ACCESS_DENIED</code>	The user does not have the ability to read the specified source list. This does not indicate whether a product or patch is found.
<code>ERROR_BAD_CONFIGURATION</code>	The configuration data is corrupt.
<code>ERROR_INVALID_PARAMETER</code>	An invalid parameter is passed to the function.
<code>ERROR_MORE_DATA</code>	The provided buffer is not sufficient to contain the requested data.
<code>ERROR_NO_MORE_ITEMS</code>	There are no more sources in the specified list to enumerate.
<code>ERROR_SUCCESS</code>	A source is enumerated successfully.
<code>ERROR_UNKNOWN_PATCH</code>	The patch specified is not installed on the computer in the specified contexts.

ERROR_UNKNOWN_PRODUCT	The product specified is not installed on the computer in the specified contexts.
ERROR_FUNCTION_FAILED	Unexpected internal failure.

## Remarks

When making multiple calls to **MsiSourceListEnumSources** to enumerate all sources for a single product instance, each call must be made from the same thread.

An administrator can enumerate per-user unmanaged and managed installations for themselves, per-machine installations, and per-user managed installations for any user. An administrator cannot enumerate per-user unmanaged installations for other users. Non-administrators can only enumerate their own per-user unmanaged and managed installations and per-machine installations.

**Windows Installer 2.0:** Not supported. The **MsiSourceListEnumSources** function is available beginning with Windows Installer 3.0.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib

<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiSourceListEnumSourcesW</b> (Unicode) and <b>MsiSourceListEnumSourcesA</b> (ANSI)

## See Also

[ProductCode](#)

[Installation Context](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSourceListEnumMediaDisks Function

The **MsiSourceListEnumMediaDisks** function enumerates the list of disks registered for the media source for a patch or product.

## Syntax

```
C++UINT MsiSourceListEnumMediaDisks(  
    __in          LPCTSTR szProductCodeOrPatchCode,  
    __in_opt     LPCTSTR szUserSID,  
    __in         MSIINSTALLCONTEXT dwContext,  
    __in         DWORD dwOptions,  
    __in         DWORD dwIndex,  
    __out_opt    LPWORD pdwDiskId,  
    __out_opt    LPTSTR szVolumeLabel,  
    __inout_opt LPDWORD pcchVolumeLabel,  
    __out_opt    LPTSTR szDiskPrompt,  
    __inout_opt LPDWORD pcchDiskPrompt  
);
```

## Parameters

*szProductCodeOrPatchCode* [in]

The **ProductCode** or patch GUID of the product or patch. Use a null-terminated string. If the string is longer than 39 characters, the function fails and returns `ERROR_INVALID_PARAMETER`. This parameter cannot be `NULL`.

*szUserSID* [in, optional]

A string SID that specifies the user account that contains the product or patch. The SID is not validated or resolved. An incorrect SID can return `ERROR_UNKNOWN_PRODUCT` or `ERROR_UNKNOWN_PATCH`. When referencing a machine context, *szUserSID* must be `NULL` and *dwContext* must be `MSIINSTALLCONTEXT_MACHINE`.

Type of SID	Meaning
-------------	---------

NULL	A NULL denotes the currently logged on user. When referencing the current user account, <i>szUserSID</i> can be NULL and <i>dwContext</i> can be MSIINSTALLCONTEXT_USERMANAGED or MSIINSTALLCONTEXT_USERUNMANAGED.
User SID	An enumeration for a specific user in the system. An example of a user SID is "S-1-3-64-2415071341-1358098788-3127455600-2561".
s-1-1-0	The special SID string s-1-1-0 (everyone) specifies enumeration across all users in the system.

**Note** The special SID string s-1-5-18 (system) cannot be used to enumerate products or patches installed as per-machine. Setting the SID value to s-1-5-18 returns ERROR\_INVALID\_PARAMETER.

*dwContext* [in]

This parameter specifies the context of the product or patch instance. This parameter can contain one of the following values.

Type of context	Meaning
MSIINSTALLCONTEXT_USERMANAGED	The product or patch instance exists in the per-user-managed context.
MSIINSTALLCONTEXT_USERUNMANAGED	The product or patch instance exists in the per-user-unmanaged context.
MSIINSTALLCONTEXT_MACHINE	The product or patch instance exists in the per-

	machine context.
--	------------------

*dwOptions* [in]

The *dwOptions* value that specifies the meaning of *szProductCodeOrPatchCode*.

Flag	Meaning
MSICODE_PRODUCT	<i>szProductCodeOrPatchCode</i> is a product code GUID.
MSICODE_PATCH	<i>szProductCodeOrPatchCode</i> is a patch code GUID.

*dwIndex* [in]

The index of the source to retrieve. This parameter must be 0 (zero) for the first call to the **MsiSourceListEnumMediaDisks** function, and then incremented for subsequent calls until the function returns **ERROR\_NO\_MORE\_ITEMS**.

*pdwDiskId* [out, optional]

On entry to **MsiSourceListEnumMediaDisks** this parameter provides a pointer to a **DWORD** to receive the ID of the disk that is being enumerated. This parameter is optional.

*szVolumeLabel* [out, optional]

An output buffer that receives the volume label of the disk that is being enumerated. This buffer should be large enough to contain the information. If the buffer is too small, the function returns **ERROR\_MORE\_DATA** and sets *\*pcchVolumeLabel* to the number of **TCHAR** in the value, not including the terminating NULL character.

If *szVolumeLabel* and *pcchVolumeLabel* are both set to NULL, the function returns **ERROR\_SUCCESS** if the value exists, without retrieving the value.

*pcchVolumeLabel* [in, out, optional]

A pointer to a variable that specifies the number of **TCHAR** in the *szVolumeLabel* buffer. When the function returns, this parameter is

the number of **TCHAR** in the received value, not including the terminating null character.

This parameter can be set to NULL only if *szVolumeLabel* is also NULL, otherwise the function returns **ERROR\_INVALID\_PARAMETER**.

*szDiskPrompt* [out, optional]

An output buffer that receives the disk prompt of the disk that is being enumerated. This buffer should be large enough to contain the information. If the buffer is too small, the function returns **ERROR\_MORE\_DATA** and sets *\*pcchDiskPrompt* to the number of **TCHAR** in the value, not including the terminating NULL character.

If the *szDiskPrompt* is set to NULL and *pcchDiskPrompt* is set to a valid pointer, the function returns **ERROR\_SUCCESS** and sets *\*pcchDiskPrompt* to the number of **TCHAR** in the value, not including the terminating NULL character. The function can then be called again to retrieve the value, with *szDiskPrompt* buffer large enough to contain *\*pcchDiskPrompt* + 1 characters.

If *szDiskPrompt* and *pcchDiskPrompt* are both set to NULL, the function returns **ERROR\_SUCCESS** if the value exists, without retrieving the value.

*pcchDiskPrompt* [in, out, optional]

A pointer to a variable that specifies the number of **TCHAR** in the *szDiskPrompt* buffer. When the function returns, this parameter is set to the size of the requested value whether or not the function copies the value into the specified buffer. The size is returned as the number of **TCHAR** in the requested value, not including the terminating null character.

This parameter can be set to NULL only if *szDiskPrompt* is also NULL, otherwise the function returns **ERROR\_INVALID\_PARAMETER**.

## Return Value

The **MsiSourceListEnumMediaDisks** function returns the following values.



<b>Value</b>	<b>Meaning</b>
ERROR_ACCESS_DENIED	The user does not have the ability to read the specified media source or the specified product or patch. This does not indicate whether a media source, product, or patch is found.
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INVALID_PARAMETER	An invalid parameter is passed to the function.
ERROR_NO_MORE_ITEMS	There are no more disks registered for this product or patch.
ERROR_SUCCESS	The value is enumerated successfully.
ERROR_UNKNOWN_PATCH	The patch is not found.
ERROR_UNKNOWN_PRODUCT	The product is not found.
ERROR_MORE_DATA	The buffer that is provided is too small to contain the requested information.
ERROR_FUNCTION_FAILED	Unexpected internal failure.

## Remarks

When making multiple calls to **MsiSourceListEnumMediaDisks** to enumerate all the sources for a single product instance, each call must be made from the same thread.

An administrator can enumerate per-user unmanaged and managed installations for themselves, per-machine installations, and per-user managed installations for any user. An administrator cannot enumerate per-user unmanaged installations for other users. Non-administrators can only enumerate their own per-user unmanaged and managed

installations and per-machine installations.

**Windows Installer 2.0:** Not supported. The **MsiSourceListEnumMediaDisks** function is available beginning with Windows Installer 3.0.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiSourceListEnumMediaDisksW</b> (Unicode) and <b>MsiSourceListEnumMediaDisksA</b> (ANSI)

## See Also

### [ProductCode](#)

### [Installation Context](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSourceListForceResolution Function

The **MsiSourceListForceResolution** function forces the installer to search the source list for a valid product source the next time a source is required. For example, when the installer performs an installation or reinstallation, or when it requires the path for a component that is set to run from source.

## Syntax

```
C++UINT MsiSourceListForceResolution(  
    __in LPCTSTR szProduct,  
    __in LPCTSTR szUserName,  
    __in DWORD dwReserved  
);
```

## Parameters

*szProduct* [in]

The **ProductCode** of the product to modify.

*szUserName* [in]

The user name for a per-user installation. The user name should always be in the format of DOMAIN\USERNAME (or MACHINENAME\USERNAME for a local user).

An empty string or null for a per-machine installation.

*dwReserved* [in]

Reserved for future use. This value must be set to 0.

## Return Value

The **MsiSourceListForceResolution** function returns the following values.

Value	Meaning

ERROR_ACCESS_DENIED	The caller does not have sufficient access to force resolution for the product.
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_BAD_USER_NAME	The specified user is not a valid user.
ERROR_FUNCTION_FAILED	The function could not complete.
ERROR_INSTALL_SERVICE_FAILURE	The installation service could not be accessed.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_SUCCESS	The function succeeded.
ERROR_UNKNOWN_PRODUCT	The specified product is unknown.

## Remarks

An administrator can modify per-machine installations, their own per-user non-managed installations, and the per-user managed installations for any user. A non-administrator can only modify per-machine installations and their own (managed or non-managed) per-user installations.

If the user name is an empty string or NULL, the function operates on the per-machine installation of the product. In this case, if the product is installed as per-user only, the function returns ERROR\_UNKNOWN\_PRODUCT.

If the user name is not an empty string or NULL, it specifies the name of the user whose product installation is modified. If the user name is the

current user name, the function first attempts to modify a non-managed installation of the product. If no non-managed installation of the product can be found, the function then tries to modify a managed per-user installation of the product. If no managed or unmanaged per-user installations of the product can be found, the function returns `ERROR_UNKNOWN_PRODUCT`, even if the product is installed per-machine.

This function can not modify a non-managed installation for any user besides the current user. If the user name is not an empty string or `NULL`, but is not the current user, the function only checks for a managed per-user installation of the product for the specified user. If the product is not installed as managed per-user for the specified user, the function returns `ERROR_UNKNOWN_PRODUCT`, even if the product is installed per-machine.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiSourceListForceResolutionW</b> (Unicode) and <b>MsiSourceListForceResolutionA</b> (ANSI)

## See Also

[Source Resiliency](#)  
[Installation Context](#)  
**LookupAccountName**

## **MsiGetComponentPath**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSourceListForceResolutionEx Function

The **MsiSourceListForceResolutionEx** function removes the registration of the property called "LastUsedSource". This function does not affect the registered source list. Whenever the installer requires the source to reinstall a product or patch, it first tries the source registered as "LastUsedSource". If that fails, or if that registration is missing, the installer searches the other registered sources until it finds a valid source or until the list of sources is exhausted. Clearing the "LastUsedSource" registration forces the installer to do a source resolution against the registered sources the next time it requires the source.

## Syntax

```
C++UINT MsiSourceListForceResolutionEx(  
    __in      LPCTSTR szProductCodeOrPatchCode,  
    __in_opt  LPCTSTR szUserSid,  
    __in      MSIINSTALLCONTEXT dwContext,  
    __in      DWORD dwOptions  
);
```

## Parameters

*szProductCodeOrPatchCode* [in]

The **ProductCode** or patch GUID of the product or patch. Use a null-terminated string. If the string is longer than 39 characters, the function fails and returns `ERROR_INVALID_PARAMETER`. This parameter cannot be `NULL`.

*szUserSid* [in, optional]

This parameter can be a string SID that specifies the user account that contains the product or patch. The SID is not validated or resolved. An incorrect SID can return `ERROR_UNKNOWN_PRODUCT` or `ERROR_UNKNOWN_PATCH`. When referencing a machine context, *szUserSID* must be `NULL` and *dwContext* must be `MSIINSTALLCONTEXT_MACHINE`. Using the

machine SID ("S-1-5-18") returns ERROR\_INVALID\_PARAMETER. When referencing the current user account, *szUserSID* can be NULL and *dwContext* can be MSIINSTALLCONTEXT\_USERMANAGED or MSIINSTALLCONTEXT\_USERUNMANAGED.

*dwContext* [in]

This parameter specifies the context of the product or patch instance. This parameter can contain one of the following values.

Type of context	Meaning
MSIINSTALLCONTEXT_USERMANAGED	The product or patch instance exists in the per-user-managed context.
MSIINSTALLCONTEXT_USERUNMANAGED	The product or patch instance exists in the per-user-unmanaged context.
MSIINSTALLCONTEXT_MACHINE	The product or patch instance exists in the per-machine context.

*dwOptions* [in]

The *dwOptions* value determines the interpretation of the *szProductCodeOrPatchCode* value .

Flag	Meaning
MSICODE_PRODUCT	<i>szProductCodeOrPatchCode</i> is a product code.
MSICODE_PATCH	<i>szProductCodeOrPatchCode</i> is a patch code.

## Return Value

The **MsiSourceListForceResolutionEx** function returns the following values.

Value	Meaning
ERROR_ACCESS_DENIED	The user does not have the ability to modify the specified source list. Does not indicate whether the product or patch was found.
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INSTALL_SERVICE_FAILURE	Could not access the Windows Installer service
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_SUCCESS	The "LastUsedSource" registration was cleared.
ERROR_UNKNOWN_PATCH	The patch was not found.
ERROR_UNKNOWN_PRODUCT	The specified product or patch was not found.
ERROR_FUNCTION_FAILED	Unexpected internal failure.

## Remarks

Administrators can modify the installation of a product or patch instance that exists under the machine context or under their own per-user context

(managed or unmanaged.) They can modify the installation of a product or patch instance that exists under any user's per-user-managed context. Administrators cannot modify another user's installation of a product or patch instance that exists under that other user's per-user-unmanaged context.

Non-administrators cannot modify the installation of a product or patch instance that exists under another user's per-user context (managed or unmanaged.) They can modify the installation of a product or patch instance that exists under their own per-user-unmanaged context. They can modify the installation of a product or patch instance under the machine context or their own per-user-managed context only if they are enabled to browse for a product or patch source. Users can be enabled to browse for sources by setting policy, for more information, see [DisableBrowse](#), [AllowLockdownBrowse](#), and [AlwaysInstallElevated](#) policies.

**Windows Installer 2.0:** Not supported. The **MsiSourceListForceResolutionEx** function is available beginning with Windows Installer 3.0.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiSourceListForceResolutionExW</b> (Unicode) and <b>MsiSourceListForceResolutionExA</b> (ANSI)

## See Also

[ProductCode](#)

[Source Resiliency](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSourceListGetInfo Function

The **MsiSourceListGetInfo** function retrieves information about the source list for a product or patch in a specific context.

## Syntax

```
C++UINT MsiSourceListGetInfo(  
    __in          LPCTSTR szProductCodeOrPatchCode,  
    __in_opt     LPCTSTR szUserSid,  
    __in         MSIINSTALLCONTEXT dwContext,  
    __in         DWORD dwOptions,  
    __in         LPCTSTR szProperty,  
    __out_opt    LPTSTR szValue,  
    __inout_opt LPDWORD pcchValue  
);
```

## Parameters

*szProductCodeOrPatchCode* [in]

The **ProductCode** or patch GUID of the product or patch. Use a null-terminated string. If the string is longer than 39 characters, the function fails and returns `ERROR_INVALID_PARAMETER`. This parameter cannot be `NULL`.

*szUserSid* [in, optional]

This parameter can be a string security identifier (SID) that specifies the user account that contains the product or patch. The SID is not validated or resolved. An incorrect SID can return `ERROR_UNKNOWN_PRODUCT` or `ERROR_UNKNOWN_PATCH`. When referencing a machine context, *szUserSID* must be `NULL` and *dwContext* must be `MSIINSTALLCONTEXT_MACHINE`.

Type of SID	Meaning
NULL	NULL denotes the currently logged on user. When referencing the current user account, <i>szUserSID</i> can be <code>NULL</code> and <i>dwContext</i> can be <code>MSIINSTALLCONTEXT_USERMANAGED</code> or

	MSIINSTALLCONTEXT_USERUNMANAGED.
User SID	Specifies enumeration for a specific user in the system. An example of a user SID is "S-1-3-64-2415071341-1358098788-3127455600-2561".

**Note** The special SID string s-1-5-18 (system) cannot be used to enumerate products or patches installed as per-machine. Setting the SID value to s-1-5-18 returns ERROR\_INVALID\_PARAMETER.

**Note** The special SID string s-1-1-0 (everyone) should not be used. Setting the SID value to s-1-1-0 fails and returns ERROR\_INVALID\_PARAM.

*dwContext* [in]

This parameter specifies the context of the product or patch instance. This parameter can contain one of the following values.

Type of context	Meaning
MSIINSTALLCONTEXT_USERMANAGED	The product or patch instance exists in the per-user-managed context.
MSIINSTALLCONTEXT_USERUNMANAGED	The product or patch instance exists in the per-user-unmanaged context.
MSIINSTALLCONTEXT_MACHINE	The product or patch instance exists in the per-machine context.

*dwOptions* [in]

The *dwOptions* value specifies the meaning of *szProductCodeOrPatchCode*.

Flag	Meaning
MSICODE_PRODUCT	<i>szProductCodeOrPatchCode</i> is a product code GUID.
MSICODE_PATCH	<i>szProductCodeOrPatchCode</i> is a patch code GUID.

*szProperty* [in]

A null-terminated string that specifies the property value to retrieve. The *szProperty* parameter can be one of the following values.

Name	Meaning
INSTALLPROPERTY_MEDIAPACKAGEPATH "MediaPackagePath"	The path relative to the root of the installation media.
INSTALLPROPERTY_DISKPROMPT "DiskPrompt"	The prompt template that is used when prompting the user for installation media.
INSTALLPROPERTY_LASTUSEDSOURCE "LastUsedSource"	The most recently used source location for the product.
INSTALLPROPERTY_LASTUSEDTYPE "LastUsedType"	An "n" if the last-used source is a network location. A "u" if the last used source is a URL location. An "m" if the last used source is media. An empty

	string ("" ) if there is no last used source.
INSTALLPROPERTY_PACKAGENAME "PackageName"	The name of the Windows Installer package or patch package on the source.

*szValue* [out, optional]

An output buffer that receives the information. This buffer should be large enough to contain the information. If the buffer is too small, the function returns `ERROR_MORE_DATA` and sets *pcchValue* to the number of **TCHAR** in the value, not including the terminating NULL character.

If the *szValue* is set to NULL and *pcchValue* is set to a valid pointer, the function returns `ERROR_SUCCESS` and sets *pcchValue* to the number of **TCHAR** in the value, not including the terminating NULL character. The function can then be called again to retrieve the value, with *szValue* buffer large enough to contain *pcchValue* + 1 characters.

If *szValue* and *pcchValue* are both set to NULL, the function returns `ERROR_SUCCESS` if the value exists, without retrieving the value.

*pcchValue* [in, out, optional]

A pointer to a variable that specifies the number of **TCHAR** in the *szValue* buffer. When the function returns, this parameter is set to the size of the requested value whether or not the function copies the value into the specified buffer. The size is returned as the number of **TCHAR** in the requested value, not including the terminating null character.

This parameter can be set to NULL only if *szValue* is also NULL, otherwise the function returns `ERROR_INVALID_PARAMETER`.

## Return Value

The **MsiSourceListGetInfo** function returns the following values.

<b>Value</b>	<b>Meaning</b>
ERROR_ACCESS_DENIED	The user does not have the ability to read the specified source list. This does not indicate whether a product or patch is found.
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INVALID_PARAMETER	An invalid parameter is passed to the function.
ERROR_MORE_DATA	The provided buffer is not sufficient to contain the requested data.
ERROR_SUCCESS	The property is retrieved successfully.
ERROR_UNKNOWN_PATCH	The patch is not found.
ERROR_UNKNOWN_PRODUCT	The product is not found.
ERROR_UNKNOWN_PROPERTY	The source property is not found.
ERROR_FUNCTION_FAILED	An unexpected internal failure.

## Remarks

Administrators can modify the installation of a product or patch instance that exists under the machine context or under their own per-user context (managed or unmanaged.) They can modify the installation of a product or patch instance that exists under any user's per-user-managed context. Administrators cannot modify another user's installation of a product or patch instance that exists under that other user's per-user-unmanaged context.

Non-administrators cannot modify the installation of a product or patch instance that exists under another user's per-user context (managed or unmanaged.) They can modify the installation of a product or patch

instance that exists under their own per-user-unmanaged context. They can modify the installation of a product or patch instance under the machine context or their own per-user-managed context only if they are enabled to browse for a product or patch source. Users can be enabled to browse for sources by setting policy. For more information, see [DisableBrowse](#), [AllowLockdownBrowse](#), and [AlwaysInstallElevated](#) policies.

**Windows Installer 2.0:** Not supported. The **MsiSourceListGetInfo** function is available beginning with Windows Installer 3.0.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiSourceListGetInfoW</b> (Unicode) and <b>MsiSourceListGetInfoA</b> (ANSI)

## See Also

### [MsiSourceListSetInfoProductCode](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSourceListSetInfo Function

The **MsiSourceListSetInfo** function sets information about the source list for a product or patch in a specific context.

## Syntax

```
C++UINT MsiSourceListSetInfo(  
    __in      LPCTSTR szProductCodeOrPatchCode,  
    __in_opt LPCTSTR szUserSid,  
    __in      MSIINSTALLCONTEXT dwContext,  
    __in      DWORD dwOptions,  
    __in      LPCTSTR szProperty,  
    __in      LPCTSTR szValue  
);
```

## Parameters

*szProductCodeOrPatchCode* [in]

The **ProductCode** or patch GUID of the product or patch. Use a null-terminated string. If the string is longer than 39 characters, the function fails and returns `ERROR_INVALID_PARAMETER`. This parameter cannot be `NULL`.

*szUserSid* [in, optional]

This parameter can be a string SID that specifies the user account that contains the product or patch. The SID is not validated or resolved. An incorrect SID can return `ERROR_UNKNOWN_PRODUCT` or `ERROR_UNKNOWN_PATCH`. When referencing a machine context, *szUserSID* must be `NULL` and *dwContext* must be `MSIINSTALLCONTEXT_MACHINE`.

Type of SID	Meaning
NULL	NULL denotes the currently logged on user. When referencing the current user account, <i>szUserSID</i> can be <code>NULL</code> and <i>dwContext</i> can be <code>MSIINSTALLCONTEXT_USERMANAGED</code> or <code>MSIINSTALLCONTEXT_USERUNMANAGED</code> .

User SID	Specifies enumeration for a particular user in the system. An example of a user SID is "S-1-3-64-2415071341-1358098788-3127455600-2561".
----------	--

**Note** The special SID string s-1-5-18 (system) cannot be used to enumerate products or patches installed as per-machine. Setting the SID value to s-1-5-18 returns ERROR\_INVALID\_PARAMETER.

**Note** The special SID string s-1-1-0 (everyone) should not be used. Setting the SID value to s-1-1-0 fails and returns ERROR\_INVALID\_PARAM.

*dwContext* [in]

This parameter specifies the context of the product or patch instance. This parameter can contain one of the following values.

Type of context	Meaning
MSIINSTALLCONTEXT_USERMANAGED	The product or patch instance exists in the per-user-managed context.
MSIINSTALLCONTEXT_USERUNMANAGED	The product or patch instance exists in the per-user-unmanaged context.
MSIINSTALLCONTEXT_MACHINE	The product or patch instance exists in the per-machine context.

*dwOptions* [in]

The *dwOptions* value specifies the meaning of *szProductCodeOrPatchCode*.

If the property being set is "LastUsedSource", this parameter also specifies the type of source as network or URL. In this case, the *dwOptions* parameter must be a combination of one of the following MSISOURCETYPE\_\* constants and one of the following MSICODE\_\* constants.

Flag	Meaning
MSISOURCETYPE_NETWORK	The source is a network type.
MSISOURCETYPE_URL	The source is a URL type.
MSICODE_PRODUCT	<i>szProductCodeOrPatchCode</i> is a product code GUID.
MSICODE_PATCH	<i>szProductCodeOrPatchCode</i> is a patch code GUID.

*szProperty* [in]

The parameter *szProperty* indicates the property value to set. Not all properties that can be retrieved through [MsiSourceListGetInfo](#) can be set via a call to [MsiSourceListSetInfo](#). The *szProperty* value can be one of the following values.

Name	Meaning
INSTALLPROPERTY_MEDIAPACKAGEPATH "MediaPackagePath"	The path relative to the of the installation media.
INSTALLPROPERTY_DISKPROMPT "DiskPrompt"	The prompt template used when prompting the user for installation media.
INSTALLPROPERTY_LASTUSEDSOURCE "LastUsedSource"	The most recently used source location for the product. If the source is registered, the function <a href="#">MsiSourceListAddSource</a> to register it. On success

	registration, the function sets the source as the LastUsedSource.
INSTALLPROPERTY_PACKAGENAME "PackageName"	The name of the Windows Installer package or patch package on the source.

*szValue* [in]

The new value of the property. No validation of the new value is performed. This value cannot be NULL. It can be an empty string.

## Return Value

The **MsiSourceListSetInfo** function returns the following values.

Value	Meaning
ERROR_ACCESS_DENIED	The user does not have the ability to set the source list for the specified product.
ERROR_BAD_CONFIGURATION	The configuration data is corrupt.
ERROR_INSTALL_SERVICE_FAILURE	The Windows Installer service could not be accessed.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_SUCCESS	The property was set.
ERROR_UNKNOWN_PATCH	The patch was not found.
ERROR_UNKNOWN_PRODUCT	The product was not found.
ERROR_UNKNOWN_PROPERTY	The source property was not found.

ERROR_FUNCTION_FAILED	Unexpected internal failure.
-----------------------	------------------------------

## Remarks

Administrators can modify the installation of a product or patch instance that exists under the machine context or under their own per-user context (managed or unmanaged.) They can modify the installation of a product or patch instance that exists under any user's per-user-managed context. Administrators cannot modify another user's installation of a product or patch instance that exists under that other user's per-user-unmanaged context.

Non-administrators cannot modify the installation of a product or patch instance that exists under another user's per-user context (managed or unmanaged.) They can modify the installation of a product or patch instance that exists under their own per-user-unmanaged context. They can modify the installation of a product or patch instance under the machine context or their own per-user-managed context only if they are enabled to browse for a product or patch source. Users can be enabled to browse for sources by setting policy. For more information, see the [DisableBrowse](#), [AllowLockdownBrowse](#), and [AlwaysInstallElevated](#) policies.

An exception to the above rule is setting "LastUsedSource" to one of the registered sources. If the source is already registered, a non-administrator can set "LastUsedSource" to their own installations (managed or non-managed) and per-machine installations, irrespective of policies.

**Windows Installer 2.0:** Not supported. The **MsiSourceListSetInfo** function is available beginning with Windows Installer 3.0.

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or
--

<b>Version</b>	Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiSourceListSetInfoW</b> (Unicode) and <b>MsiSourceListSetInfoA</b> (ANSI)

## See Also

### [ProductCode](#)

[Installation Context](#)

### [MsiSourceListGetInfo](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiUseFeature Function

The **MsiUseFeature** function increments the usage count for a particular feature and indicates the installation state for that feature. This function should be used to indicate an application's intent to use a feature.

## Syntax

```
C++INSTALLSTATE MsiUseFeature(  
    __in LPCTSTR szProduct,  
    __in LPCTSTR szFeature  
);
```

## Parameters

*szProduct* [in]

Specifies the product code for the product that owns the feature to be used.

*szFeature* [in]

Identifies the feature to be used.

## Return Value

Value	Meaning
INSTALLSTATE_ABSENT	The feature is not installed.
INSTALLSTATE_ADVERTISED	The feature is advertised
INSTALLSTATE_BADCONFIG	The configuration data is corrupt.
INSTALLSTATE_INVALIDARG	Invalid function argument.
INSTALLSTATE_LOCAL	The feature is locally installed and available for use.
INSTALLSTATE_SOURCE	The feature is installed from the source and available for use.

INSTALLSTATE_UNKNOWN	The feature is not published.
----------------------	-------------------------------

## Remarks

The [MsiUseFeature](#) function should only be used on features known to be published. INSTALLSTATE\_UNKNOWN indicates that the program is trying to use a feature that is not published. The application should determine whether the feature is published before calling [MsiUseFeature](#) by calling [MsiQueryFeatureState](#) or [MsiEnumFeatures](#). The application should make these calls while it initializes. An application should only use features that are known to be published.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiUseFeatureW</b> (Unicode) and <b>MsiUseFeatureA</b> (ANSI)

## See Also

[Application-Only Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiUseFeatureEx Function

The **MsiUseFeatureEx** function increments the usage count for a particular feature and indicates the installation state for that feature. This function should be used to indicate an application's intent to use a feature.

## Syntax

```
C++INSTALLSTATE MsiUseFeatureEx(  
    __in LPCTSTR szProduct,  
    __in LPCTSTR szFeature,  
    __in DWORD dwInstallMode,  
    __in DWORD dwReserved  
);
```

## Parameters

*szProduct* [in]

Specifies the product code for the product that owns the feature to be used.

*szFeature* [in]

Identifies the feature to be used.

*dwInstallMode* [in]

This parameter can have the following value.

Value	Meaning
INSTALLMODE_NODETECTION	Return value indicates the installation state.

*dwReserved* [in]

Reserved for future use. This value must be set to 0.

## Return Value



Value	Meaning
INSTALLSTATE_ABSENT	The feature is not installed.
INSTALLSTATE_ADVERTISED	The feature is advertised
INSTALLSTATE_LOCAL	The feature is locally installed and available for use.
INSTALLSTATE_SOURCE	The feature is installed from the source and available for use.
INSTALLSTATE_UNKNOWN	The feature is not published.

## Remarks

The **MsiUseFeatureEx** function should only be used on features known to be published. `INSTALLSTATE_UNKNOWN` indicates that the program is trying to use a feature that is not published. The application should determine whether the feature is published before calling **MsiUseFeature** by calling **MsiQueryFeatureState** or **MsiEnumFeatures**. The application should make these calls while it initializes. An application should only use features that are known to be published.

## Requirements

	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Version</b>	
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib

<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiUseFeatureExW</b> (Unicode) and <b>MsiUseFeatureExA</b> (ANSI)

## See Also

[Application-Only Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiVerifyPackage Function

The **MsiVerifyPackage** function verifies that the given file is an installation package.

## Syntax

```
C++UINT MsiVerifyPackage(  
    __in LPCTSTR szPackagePath  
);
```

## Parameters

*szPackagePath* [in]

Specifies the path and file name of the package.

## Return Value

Value	Meaning
ERROR_INSTALL_PACKAGE_INVALID	The file is not a valid package.
ERROR_INSTALL_PACKAGE_OPEN_FAILED	The file could not be opened.
ERROR_INVALID_PARAMETER	The parameter is not valid.
ERROR_SUCCESS	The file is a package.

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or

<b>Version</b>	Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000. See the <a href="#">Windows Installer Run-Time Requirements</a> for information about the minimum Windows service pack that is required by a Windows Installer version.
<b>Header</b>	Msi.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiVerifyPackageW</b> (Unicode) and <b>MsiVerifyPackageA</b> (ANSI)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Initialization Errors

This table describes initialization error codes and values.

Error code	Value	Error
ERROR_SUCCESS	0	Initialization complete
ERROR_ALREADY_INITIALIZED	1247	The installer is already initialized
ERROR_INSTALL_SERVICE_FAILURE	1601	Install server could not be accessed
ERROR_BAD_CONFIGURATION	1610	Configuration data is corrupt
ERROR_INSTALL_PACKAGE_VERSION	1613	Installer version does not support database format
ERROR_INSTALL_ALREADY_RUNNING	1618	An installation is already in progress
ERROR_INSTALL_PACKAGE_OPEN_FAILED	1619	Database could not be opened
ERROR_INSTALL_PACKAGE_INVALID	1620	Incompatible database
ERROR_INSTALL_UI_FAILURE	1621	Could not initialize handler interface

ERROR_INSTALL_LOG_FAILURE	1622	Could not open log file in requested mode
ERROR_INSTALL_LANGUAGE_UNSUPPORTED	1623	No acceptable language could be found
ERROR_INSTALL_TRANSFORM_FAILURE	1624	Database transform failed to merge
ERROR_INSTALL_PACKAGE_REJECTED	1625	This installation is forbidden by system policy. Contact your system administrator.
ERROR_INSTALL_PLATFORM_UNSUPPORTED	1633	The platform specified by the <b>Template Summary</b> property is not supported.
ERROR_PATCH_PACKAGE_OPEN_FAILED	1635	Patch package could not be opened
ERROR_PATCH_PACKAGE_INVALID	1636	Patch package invalid.
ERROR_PATCH_PACKAGE_UNSUPPORTED	1637	Patch package unsupported
ERROR_INVALID_COMMAND_LINE	1639	Invalid command line

		syntax
ERROR_INSTALL_REMOTE_DISALLOWED	1640	Installation from a Terminal Services client session not permitted for current user.
ERROR_PATCH_TARGET_NOT_FOUND	1642	The installer cannot install the upgrade patch because the program being upgraded may be missing or the upgrade patch updates a different version of the program. Verify that the program to be upgraded exists on your computer and that you have the correct upgrade patch.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Displayed Error Messages

An [installer function](#) may display an error message dialog box returning any of the following errors. An error box is displayed only if the user interface level is at the full, reduced, or basic level. For more information, see [User Interface Levels](#).

The installer functions only display error messages that are in the following table. For errors not in this list, the caller of the function is responsible for handling the display of error messages.

<b>Error code</b>	<b>Error</b>
ERROR_INSTALL_SUSPEND	The install was suspended.
ERROR_INSTALL_USEREXIT	The user exited the install.
ERROR_INSTALL_FAILURE	Install failed.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# InitializeEmbeddedUI Callback Function

The **InitializeEmbeddedUI** function prototype defines a user-defined initialization function exported by the embedded user interface DLL that is defined in the [MsiEmbeddedUI table](#).

## Syntax

```
C++UINT CALLBACK InitializeEmbeddedUI(  
    __in     MSIHANDLE hInstall,  
    __in     LPCWSTR wzResourcePath,  
    __inout  LPDWORD pdwInternalUILevel  
);
```

## Parameters

*hInstall* [in]

A handle to the installer that performs the installation. This handle is valid only for the duration of the **InitializeEmbeddedUI** function.

*wzResourcePath* [in]

Pointer to a null-terminated string that contains the full path to the directory that contains all the files from the MsiEmbeddedUI table. The user interface DLL can use this path to load resources stored in the Windows Installer package.

*pdwInternalUILevel* [in, out]

Pointer to a location that contains the internal UI level.

On entry to the **InitializeEmbeddedUI** function, this parameter receives a value that indicates the current UI level for the installation. The value is a combination of INSTALLUILEVEL flags that notifies the UI handler whether the installation is at the full, reduced, or basic [user interface levels](#).

The **InitializeEmbeddedUI** function should modify the value to one or more of the following flags. The installer resets the internal user interface level after the function returns.

---

Value	Meaning
INSTALLUILEVEL_FULL	The embedded UI will handle some messages by using the internal Windows Installer UI set at the full user interface level.
INSTALLUILEVEL_REDUCED	The embedded UI will handle some messages by using the internal Windows Installer UI set at the reduced user interface level.
INSTALLUILEVEL_BASIC	The dialog boxes of the internal Windows Installer basic user interface are displayed together with dialog boxes of the embedded UI. The INSTALLUILEVEL_BASIC value also disables the <b>Cancel</b> button on the basic internal UI dialog boxes.
INSTALLUILEVEL_NONE	The embedded UI will handle all messages.
INSTALLUILEVEL_SOURCERESONLY	The embedded UI will handle all messages. If this value is combined with the INSTALLUILEVEL_NONE value, the installer displays only the dialog boxes used for source resolution. No other dialog boxes are shown. This value has no effect if the UI level is not INSTALLUILEVEL_NONE.

It is used with an embedded user interface designed to handle all of the UI except for source resolution. In this case, the installer handles source resolution.

## Return Value

The **InitializeEmbeddedUI** function returns the following values.

Return code	Description
ERROR_SUCCESS	The embedded UI DLL initialized successfully. Messages are sent to this DLL.
INSTALLUILEVEL_NONE	The embedded UI DLL did not initialize, and the installation continues with no user interface.
INSTALLUILEVEL_BASIC	The embedded UI DLL did not initialize, and the installation continues using the internal basic user interface of the Windows Installer.
INSTALLUILEVEL_REDUCED	The embedded UI DLL did not initialize, and the installation continues using the internal reduced user interface of the Windows Installer.
INSTALLUILEVEL_FULL	The embedded UI DLL did not initialize, and the installation continues using the internal full user interface of the Windows Installer.
ERROR_INSTALL_FAILURE	The embedded UI DLL did not initialize. The installation returns

ERROR_INSTALL_FAILURE.
------------------------

## Remarks

The **InitializeEmbeddedUI** function should not increase the internal UI level to a level above the existing internal UI level. If the function attempts to do so, the installer caps the UI level at the existing level and the installer writes a message to the log file.

For an example of the **InitializeEmbeddedUI** function see [Using an Embedded UI](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.5 on Windows Vista, Windows XP, Windows Server 2003, and Windows Server 2008
<b>Header</b>	Msi.h

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# EmbeddedUIHandler Callback Function

The **EmbeddedUIHandler** function prototype defines a callback function exported by the embedded UI DLL that is defined in the [MsiEmbeddedUI table](#). Call the **EmbeddedUIHandler** function once for each message sent to the user interface.

**Windows Installer 4.0 and earlier:** Not supported. Available beginning with Windows Installer 4.5.

## Syntax

```
C++INT CALLBACK EmbeddedUIHandler(  
    UINT uiMessageType,  
    MSIHANDLE hRecord  
);
```

## Parameters

### *uiMessageType*

Specifies a combination of one message box style, one message box icon type, one default button, and one installation message type.

Include one of the following message box styles. If no value is specified, use MB\_OK.

Message box style	Meaning
MB_ABORTRETRYIGNORE	The message box contains the <b>Abort</b> , <b>Retry</b> , and <b>Ignore</b> buttons.
MB_OK	The message box contains the <b>OK</b> button. This is the default.
MB_OKCANCEL	The message box contains the <b>OK</b> and <b>Cancel</b> buttons.

MB_RETRYCANCEL	The message box contains the <b>Retry</b> and <b>Cancel</b> buttons.
MB_YESNO	The message box contains the <b>Yes</b> and <b>No</b> buttons.
MB_YESNOCANCEL	The message box contains the <b>Yes</b> , <b>No</b> , and <b>Cancel</b> buttons.

Include one of the following message box icon types. If no value is specified, use no icon.

Message box icon type	Meaning
MB_ICONEXCLAMATION, MB_ICONWARNING	An exclamation point appears in the message box.
MB_ICONINFORMATION, MB_ICONASTERISK	The information sign appears in the message box.
MB_ICONQUESTION	A question mark appears in the message box.
MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND	A stop sign appears in the message box.

Include one of the following default buttons. If no value is specified, use MB\_DEFBUTTON1.

Default button	Meaning
MB_DEFBUTTON1	The first button is the default button.
MB_DEFBUTTON2	The second button is the default button.
MB_DEFBUTTON3	The third button is the default button.

MB\_DEFBUTTON4

The fourth button is the default button.

Include one of the following installation message types. There is no default installation message type; an installation message type is always specified.

<b>Installation message type</b>	<b>Meaning</b>
INSTALLMESSAGE_FATALEXIT	Premature termination.
INSTALLMESSAGE_ERROR	Formatted error message.
INSTALLMESSAGE_WARNING	Formatted warning message.
INSTALLMESSAGE_USER	User request message.
INSTALLMESSAGE_INFO	Informative message for log.
INSTALLMESSAGE_FILESIUSE	List of files currently in use that must be closed before being replaced.
INSTALLMESSAGE_RESOLVESOURCE	Request to determine a valid source location.
INSTALLMESSAGE_RMFILESINUSE	List of files currently in use that must be closed before being replaced. For more information about this message, see <a href="#">Using Restart Manager with an External UI</a> .
INSTALLMESSAGE_OUTOFDISKSPACE	Insufficient disk space message.
INSTALLMESSAGE_ACTIONSTART	Start of action message.

	This message includes the action name and description.
INSTALLMESSAGE_ACTIONDATA	Formatted data associated with the individual action item.
INSTALLMESSAGE_PROGRESS	Progress gauge information. This message includes information on units so far and total number of units.
INSTALLMESSAGE_COMMONDATA	Formatted dialog information for the user interface.
INSTALLMESSAGE_INITIALIZE	Sent prior to UI initialization, with no string data.
INSTALLMESSAGE_TERMINATE	Sent after UI termination, with no string data.
INSTALLMESSAGE_SHOWDIALOG	Sent prior to the display of an authored dialog box or wizard.
INSTALLMESSAGE_INSTALLSTART	Sent prior to installation of product.
INSTALLMESSAGE_INSTALLEND	Sent after installation of product.

*hRecord*

Specifies a handle to a record that contains message data. For

information about record objects, see the [Record Processing Functions](#).

## Return Value

The **EmbeddedUIHandler** function can return the following values.

Return code	Description
-1	An error was found in the message handler.
0	No action was taken.

The following return values map to the buttons specified by the message box style:

- IDOK
- IDCANCEL
- IDABORT
- IDRETRY
- IDIGNORE
- IDYES
- IDNO

## Remarks

For an example of the **EmbeddedUIHandler** function see [Using an Embedded UI](#).

## Requirements

Version
Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.5 on Windows Server 2008, Windows Vista, Windows Server 2003, and Windows XP

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ShutdownEmbeddedUI Callback Function

The **ShutdownEmbeddedUI** function prototype defines a user-defined shutdown function exported by the embedded UI DLL that is defined in the [MsiEmbeddedUI table](#). This function should be called at the end of the installation. After calling this function, the installer sends no additional messages to the embedded UI DLL and you can unload the DLL.

**Windows Installer 4.0 and earlier:** Not supported. Available beginning with Windows Installer 4.5.

## Syntax

```
C++DWORD CALLBACK ShutdownEmbeddedUI(void);
```

## Parameters

This callback has no parameters.

## Return Value

The return value from this function prototype is 0 in all cases. If a user-defined function returns a non-zero value, the installer writes a message to the log file.

## Remarks

For an example of the **ShutdownEmbeddedUI** function see [Using an Embedded UI](#).

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.5 on Windows Vista, Windows XP, Windows

<b>Version</b>	Server 2003, and Windows Server 2008
<b>Header</b>	Msi.h

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# INSTALLUI\_HANDLER Callback Function

The [INSTALLUI\\_HANDLER](#) function prototype defines a callback function that the installer calls for progress notification and error messages. For more information on the usage of this function prototype, a sample code snippet is available in [Handling Progress Messages Using MsiSetExternalUI](#).

## Syntax

```
C++INT CALLBACK InstalluiHandler(  
    LPVOID pvContext,  
    UINT iMessageType,  
    LPCTSTR szMessage  
);
```

## Parameters

### *pvContext*

Pointer to an application context passed to the [MsiSetExternalUI](#) function. This parameter can be used for error checking.

### *iMessageType*

Specifies a combination of one message box style, one message box icon type, one default button, and one installation message type. This parameter must be one of the following.

Message box StylesFlag	Meaning
MB_ABORTRETRYIGNORE	The message box contains the <b>Abort</b> , <b>Retry</b> , and <b>Ignore</b> buttons.
MB_OK	The message box contains the <b>OK</b> button. This is the default.
MB_OKCANCEL	The message box contains the <b>OK</b> and <b>Cancel</b> buttons.

MB_RETRYCANCEL	The message box contains the <b>Retry</b> and <b>Cancel</b> buttons.
MB_YESNO	The message box contains the <b>Yes</b> and <b>No</b> buttons.
MB_YESNOCANCEL	The message box contains the <b>Yes</b> , <b>No</b> , and <b>Cancel</b> buttons.

<b>Message box IconTypesFlag</b>	<b>Meaning</b>
----------------------------------	----------------

MB_ICONEXCLAMATION, MB_ICONWARNING	An exclamation point appears in the message box.
MB_ICONINFORMATION, MB_ICONASTERISK	The information sign appears in the message box.
MB_ICONQUESTION	A question mark appears in the message box.
MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND	A stop sign appears in the message box.

<b>Default ButtonsFlag</b>	<b>Meaning</b>
----------------------------	----------------

MB_DEFBUTTON1	The first button is the default button.
MB_DEFBUTTON2	The second button is the default button.
MB_DEFBUTTON3	The third button is the default button.
MB_DEFBUTTON4	The fourth button is the default button.

<b>Install message TypesFlag</b>	<b>Meaning</b>
----------------------------------	----------------

--	--

INSTALLMESSAGE_FATALEXIT	Premature termination
INSTALLMESSAGE_ERROR	Formatted error message
INSTALLMESSAGE_WARNING	Formatted warning message
INSTALLMESSAGE_USER	User request message.
INSTALLMESSAGE_INFO	Informative message for log
INSTALLMESSAGE_FILESINUSE	List of files currently in use that must be closed before being replaced.
INSTALLMESSAGE_RESOLVESOURCE	Request to determine a valid source location
INSTALLMESSAGE_RMFILESINUSE	List of files currently in use that must be closed before being replaced. Available beginning with Windows Installer 4.0. For more information about this message see <a href="#">Using Restart Manager with an External UI</a> .
INSTALLMESSAGE_OUTOFDISKSPACE	Insufficient disk space message
INSTALLMESSAGE_ACTIONSTART	Start of action message. This message includes the action name and description.
INSTALLMESSAGE_ACTIONDATA	Formatted data associated with the individual action item.

INSTALLMESSAGE_PROGRESS	Progress gauge information. This message includes information on units so far and total number of units.
INSTALLMESSAGE_COMMONDATA	Formatted dialog information for user interface.
INSTALLMESSAGE_INITIALIZE	Sent prior to UI initialization, no string data
INSTALLMESSAGE_TERMINATE	Sent after UI termination, no string data
INSTALLMESSAGE_SHOWDIALOG	Sent prior to display of authored dialog or wizard
INSTALLMESSAGE_INSTALLSTART	Sent prior to installation of product.
INSTALLMESSAGE_INSTALLEND	Sent after installation of product.

The following defaults should be used if any of the preceding messages are missing: MB\_OK, no icon, and MB\_DEFBUTTON1. There is no default installation message type; a message type is always specified.

*szMessage*

Specifies the message text.

## **Return Value**

-1

An error was found in the message handler. Windows Installer ignores this returned value.

0

No action was taken.

The following return values map to the buttons specified by the message box style:

IDOK  
IDCANCEL  
IDABORT  
IDRETRY  
IDIGNORE  
IDYES  
IDNO

## Remarks

For more information on returning values from an external user interface handler, see the [Returning Values from an External User Interface Handler](#) topic.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msi.h
<b>Unicode and ANSI names</b>	INSTALLUI_HANDLERW (Unicode) and INSTALLUI_HANDLERA (ANSI)

## See Also

[MsiSetExternalUI](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# INSTALLUI\_HANDLER\_RECORD Callback Function

The `INSTALLUI_HANDLER_RECORD` function prototype defines a callback function that the installer calls for progress notification and error messages. Call the [MsiSetExternalUIRecord](#) function to enable a record-base external user-interface (UI) handler.

**Windows Installer 3.0 and Windows Installer 2.0:** Not supported.  
Available beginning with Windows Installer version 3.1 and later.

## Syntax

```
C++INT CALLBACK InstalluiHandlerRecord(  
    LPVOID pvContext,  
    UINT iMessageType,  
    MSIHANDLE hRecord  
);
```

## Parameters

### *pvContext*

Pointer to an application context passed to the [MsiSetExternalUIRecord](#) function. This parameter can be used for error checking.

### *iMessageType*

Specifies a combination of one message box style, one message box icon type, one default button, and one installation message type. This parameter must be one of the following.

Message box StylesFlag	Meaning
MB_ABORTRETRYIGNORE	The message box contains the <b>Abort</b> , <b>Retry</b> , and <b>Ignore</b> buttons.
MB_OK	The message box contains the <b>OK</b> button. This is the default.

MB_OKCANCEL	The message box contains the <b>OK</b> and <b>Cancel</b> buttons.
MB_RETRYCANCEL	The message box contains the <b>Retry</b> and <b>Cancel</b> buttons.
MB_YESNO	The message box contains the <b>Yes</b> and <b>No</b> buttons.
MB_YESNOCANCEL	The message box contains the <b>Yes</b> , <b>No</b> , and <b>Cancel</b> buttons.

<b>Message box IconTypesFlag</b>	<b>Meaning</b>
MB_ICONEXCLAMATION, MB_ICONWARNING	An exclamation point appears in the message box.
MB_ICONINFORMATION, MB_ICONASTERISK	The information sign appears in the message box.
MB_ICONQUESTION	A question mark appears in the message box.
MB_ICONSTOP, MB_ICONERROR, MB_ICONHAND	A stop sign appears in the message box.

<b>Default ButtonsFlag</b>	<b>Meaning</b>
MB_DEFBUTTON1	The first button is the default button.
MB_DEFBUTTON2	The second button is the default button.
MB_DEFBUTTON3	The third button is the default button.
MB_DEFBUTTON4	The fourth button is the default button.

Install message TypesFlag	Meaning
INSTALLMESSAGE_FATALEXIT	Premature termination
INSTALLMESSAGE_ERROR	Formatted error message
INSTALLMESSAGE_WARNING	Formatted warning message
INSTALLMESSAGE_USER	User request message.
INSTALLMESSAGE_INFO	Informative message for log
INSTALLMESSAGE_FILESinUSE	List of files currently in use that must be closed before being replaced
INSTALLMESSAGE_RESOLVESOURCE	Request to determine a valid source location
INSTALLMESSAGE_RMFILESinUSE	List of files currently in use that must be closed before being replaced. Available beginning with Windows Installer version 4.0. For more information about this message see <a href="#">Using Restart Manager with an External UI</a> .
INSTALLMESSAGE_OUTOFDISKSPACE	Insufficient disk space message
INSTALLMESSAGE_ACTIONSTART	Start of action message. This message includes the action name and

	description.
INSTALLMESSAGE_ACTIONDATA	Formatted data associated with the individual action item.
INSTALLMESSAGE_PROGRESS	Progress gauge information. This message includes information on units so far and total number of units.
INSTALLMESSAGE_COMMONDATA	Formatted dialog information for user interface.
INSTALLMESSAGE_INITIALIZE	Sent prior to UI initialization, no string data
INSTALLMESSAGE_TERMINATE	Sent after UI termination, no string data
INSTALLMESSAGE_SHOWDIALOG	Sent prior to display of authored dialog or wizard
INSTALLMESSAGE_INSTALLSTART	Sent prior to installation of product.
INSTALLMESSAGE_INSTALLEND	Sent after installation of product.

The following defaults should be used if any of the preceding messages are missing: MB\_OK, no icon, and MB\_DEFBUTTON1. There is no default installation message type; a message type is always specified.

*hRecord*

Specifies a handle to the record object. For information about record objects, see the [Record Processing Functions](#).

## Return Value

-1

An error was found in the message handler. Windows Installer ignores this returned value.

0

No action was taken.

The following return values map to the buttons specified by the message box style:

IDOK  
IDCANCEL  
IDABORT  
IDRETRY  
IDIGNORE  
IDYES  
IDNO

## Remarks

This type of external UI handler should be used when it is known what type of errors or messages the caller is interested in, and wants to avoid the overhead of parsing the string message that is sent to an external UI handler of `INSTALLUI_HANDLER` type, but retrieve the data of interest from fields of `hRecord`.

For more information on returning values from an external user interface handler, see the [Returning Values from an External User Interface Handler](#) topic. The `hRecord` object sent to the record-based external UI handler is owned by Windows Installer and is valid only for the callback's lifetime. The callback should extract from the record any data it needs and it should not close that handle.

Any attempt by a record-based external UI handler to alter the data in the `hRecord` object will be ignored by Windows Installer.

For more information about using a record-based external handler, see

## Monitoring an Installation Using MsiSetExternalUIRecord.

### Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.1 on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msi.h

### See Also

#### [MsiSetExternalUI](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer Structures

The Windows Installer uses the following structures.

- [MSIFILEHASHINFO](#)
- [MSIPATCHSEQUENCEINFO](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MSIFILEHASHINFO Structure

The **MSIFILEHASHINFO** structure contains the file hash information returned by [MsiGetFileHash](#) and used in the [MsiFileHash](#) table.

## Syntax

```
C++
typedef struct _MSIFILEHASHINFO {
    ULONG dwFileHashInfoSize;
    ULONG dwData[4];
}MSIFILEHASHINFO, *PMSIFILEHASHINFO;
```

## Members

### dwFileHashInfoSize

Specifies the size, in bytes, of this data structure. Set this member to `sizeof(MSIFILEHASHINFO)` before calling the **MsiGetFileHash** function.

### dwData

The entire 128-bit file hash is contained in four 32-bit fields. The first field corresponds to the HashPart1 column of the MsiHashFile table, the second field corresponds to the HashPart2 column, the third field corresponds to the HashPart3 column, and the fourth field corresponds to the HashPart4 column.

## Remarks

The file hash entered into the fields of the MsiFileHash table must be obtained by calling **MsiGetFileHash** or the [FileHash method](#). Do not use other methods to generate a file hash.

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or

<b>Version</b>	Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msi.h

## See Also

[MsiGetFileHash](#)

[MsiFileHash table](#)

[Default File Versioning](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MSIPATCHSEQUENCEINFO

## Structure

The **MSIPATCHSEQUENCEINFO** structure is used by the [MsiDeterminePatchSequence](#) and [MsiDetermineApplicablePatches](#) functions.

### Syntax

```
C++
typedef struct _MSIPATCHSEQUENCEINFO {
    LPCTSTR          szPatchData;
    MSIPATCHDATATYPE ePatchDataType;
    DWORD           dwOrder;
    UINT            uStatus;
}MSIPATCHSEQUENCEINFO, *PMSIPATCHSEQUENCEINFO;
```

### Members

#### szPatchData

Pointer to the path of a patch file, an XML blob, or an XML file.

#### ePatchDataType

Qualifies **szPatchData** as a patch file, an XML blob, or an XML file.

Value	Meaning
MSIPATCH_DATATYPE_PATCHFILE 0	The <b>szPatchData</b> member refers to a path of a patch file.
MSIPATCH_DATATYPE_XMLPATH 1	The <b>szPatchData</b> member refers to a path of a XML file.
MSIPATCH_DATATYPE_XMLBLOB 2	The <b>szPatchData</b> member refers to an XML blob.

#### dwOrder

Set to an integer that indicates the sequence of the patch in the order of application. The sequence starts with 0. If a patch is not applicable to the specified .msi file, or if the function fails, **dwOrder** is set to -1.

### **uStatus**

Set to ERROR\_SUCCESS or the corresponding Win32 error code.

## **Remarks**

**Windows Installer 2.0:** Not supported. The **MSIPATCHSEQUENCEINFO** structure is available beginning with Windows Installer 3.0.

## **Requirements**

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer 3.0 or later on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msi.h
<b>Unicode and ANSI names</b>	<b>MSIPATCHSEQUENCEINFOW</b> (Unicode) and <b>MSIPATCHSEQUENCEINFOA</b> (ANSI)

## **See Also**

[MsiDetermineApplicablePatches](#)  
[MsiDeterminePatchSequence](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer Database

This material is primarily for developers who want to write tools that create new installer packages and who need detailed information about the installer's relational database. Complete installer packages are adopted by application developers as an efficient means to manage the installation of their applications. Knowledge of the installation database is also useful to application developers and administrators who want to modify existing installer packages and installation processes using custom actions. For general information on Windows Installer and component management technology, see [About Windows Installer](#).

The installer database contains all the necessary information for the installation of an application. Therefore developers of tools for the creation of install packages and developers creating install packages without using such tools need to understand the installer database. The major areas of functionality of the installer database are presented in the following topics:

[About the Installer Database](#)

[Using the Installer Database](#)

[Installer Database Reference](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# About the Installer Database

The Windows Installer database consists of many interrelated tables that together comprise a relational database of the information necessary to install a group of applications. Because the database is relational, the tables are linked through the data in the primary and foreign key values. This provides an efficient method for changing the installation process and means that users can more easily customize a large application or group of applications. The database tables reflect the general layout of the entire group of applications, including:

- Available features
- Components
- Relationship between features and components
- Necessary registry settings
- User interface for the application

To create an installation database, you must populate the tables with all the information about the applications and the installation process. Manually authoring all these tables becomes a large task even for a moderate size installation; therefore some third-party tools are available to assist with building the installer database. The following sections describe groups of related database tables.

[Core Tables Group](#)

[File Tables Group](#)

[Registry Tables Group](#)

[System Tables Group](#)

[Locator Tables Group](#)

[Program Information Tables Group](#)

[Installation Procedure Tables Group](#)

[Entity Relationship Diagram Legend](#)

[Text Archive Files](#)

For a complete list of all tables in an installation database, see [Database Tables](#).

## **See Also**

[Released Versions, Tools, and Redistributables](#)

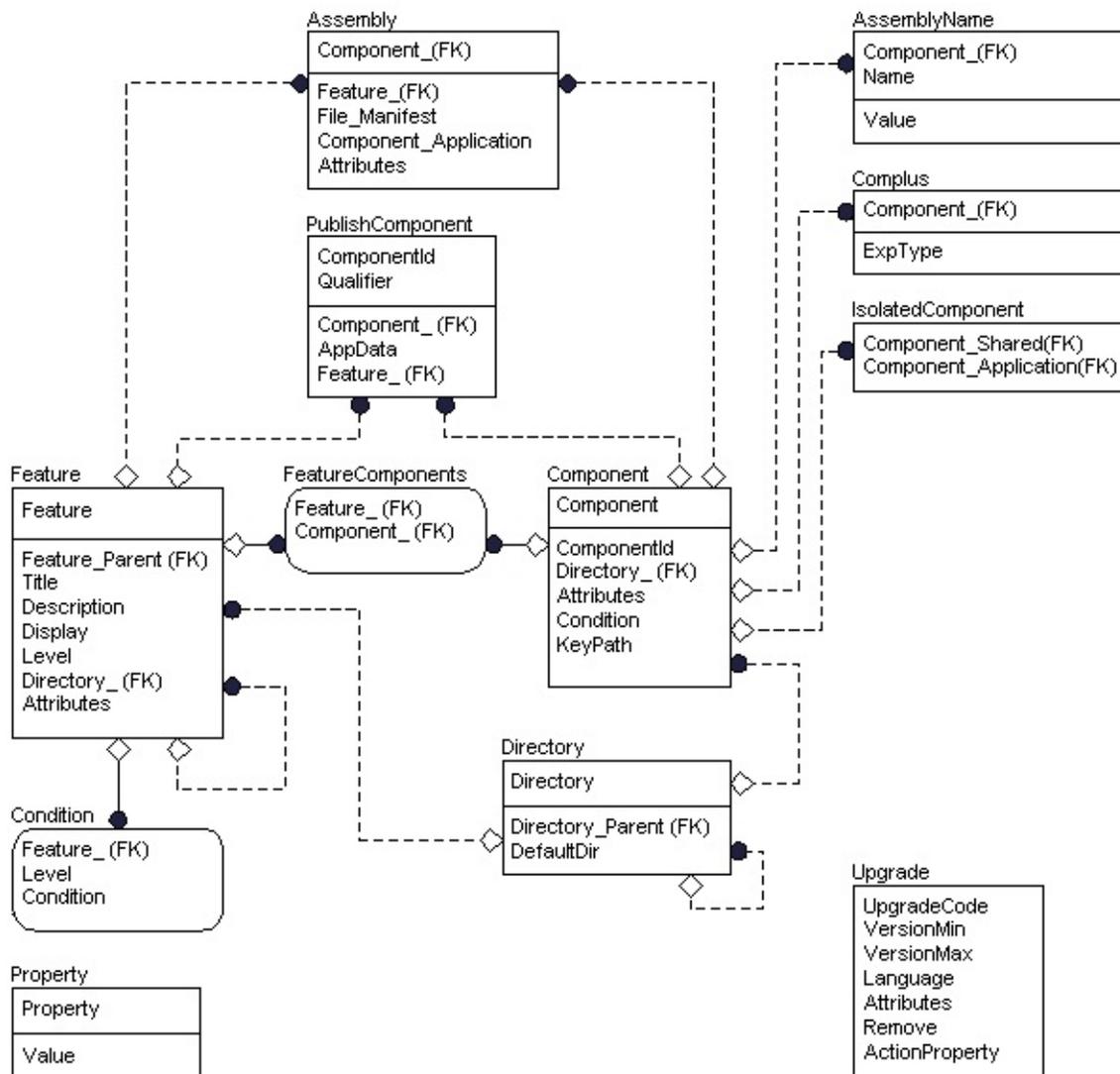
[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Core Tables Group

For more information about the following diagram, see the [entity relationship diagram legend](#).



The core group consists of tables describing the fundamental features and components of the application and the installer package. Developers of install packages should therefore consider how to populate these tables first because the organization of much of the database will become

apparent from the content of this group.

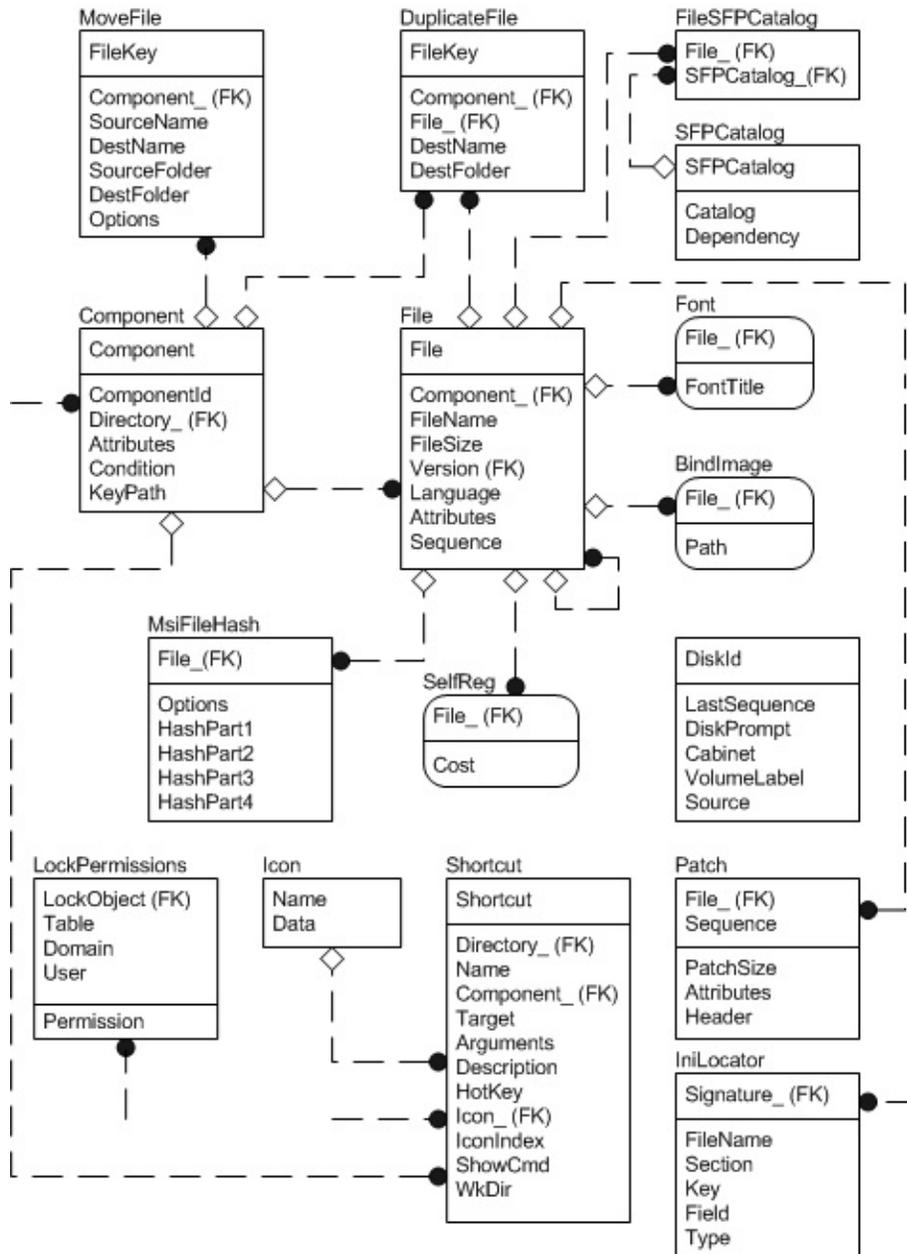
- The [Feature table](#) lists all features belonging to the application.
- The [Condition table](#) contains the conditional expressions that determine whether or not a particular feature will be installed.
- The [FeatureComponents table](#) describes which components belong to each feature.
- The [Component table](#) lists all components belonging to the installation.
- The [Directory table](#) lists the directories that are needed during the installation. Because each component must be associated with one and only one directory, the Component table is closely related to this table and has an external key to the Directory table.
- The [PublishComponent table](#) lists the features and components that are published for use by other applications. [Components and Features](#) are the two types of feature advertisement.
- The [MsiAssembly table](#) specifies Windows Installer settings for .NET Framework common language runtime assemblies and Win32 assemblies.
- The [MsiAssemblyName table](#) specifies the schema for the elements of a strong assembly cache name for a common language runtime or Win32 assembly.
- The [Complus table](#) contains information needed to install COM+ applications.
- The [IsolatedComponent table](#) associates the component specified in the Component\_Application column (commonly an .exe) with the component specified in the Component\_Shared column (commonly a shared DLL).
- The [Upgrade table](#) contains information required during [major upgrades](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# File Tables Group



For more information about this diagram, see the [entity relationship diagram legend](#).

An installer package developer should consider populating the file table

group of tables after breaking the application into [components and features](#) and after populating the [core tables group](#). The file table group contains all of the files belonging to the installation and most of these files are listed in the [File table](#). The [Directory table](#) is not shown in the figure but is closely related to the file table group. The Directory table gives the directory structure of the installation.

The file group of tables contains all of the tables that are related to files.

- The [File table](#) lists files belonging to the installation. Files that are not listed in the File table include disk files, which are listed in Media table. Because every file belongs to a component, the File table has an external key into the Component table.
- The [RemoveFile table](#) contains a list of files to be removed by the [RemoveFiles action](#).
- The [Font table](#) lists font files to be registered with the system.
- The [SelfReg table](#) lists module files of the installation that are self-registered.
- The [Media table](#) lists the source media and disks belonging to the installation.
- The [BindImage table](#) lists files that are bound to DLLs imported by executables.
- The [MoveFile table](#) specifies which files are moved during the installation.
- The [DuplicateFile table](#) specifies which files are duplicated during the installation.
- The [IniFile table](#) lists the .ini files and the information that the application needs to set in the file.
- The [RemoveIniFile table](#) contains the information an application needs to delete from a .ini file.
- The [Environment table](#) is used to set the values of environment variables.
- The [Icon table](#) provides icon information which is copied to a file as a

part of product advertisement.

- The [FileSFPCatalog table](#) associates specified files with system file protection catalog files.

**Windows Vista, Windows Server 2003, Windows XP, and Windows 2000:** Not supported.

- The [SFPCatalog table](#) contains system file protection catalogs.

**Windows Vista, Windows Server 2003, Windows XP, and Windows 2000:** Not supported.

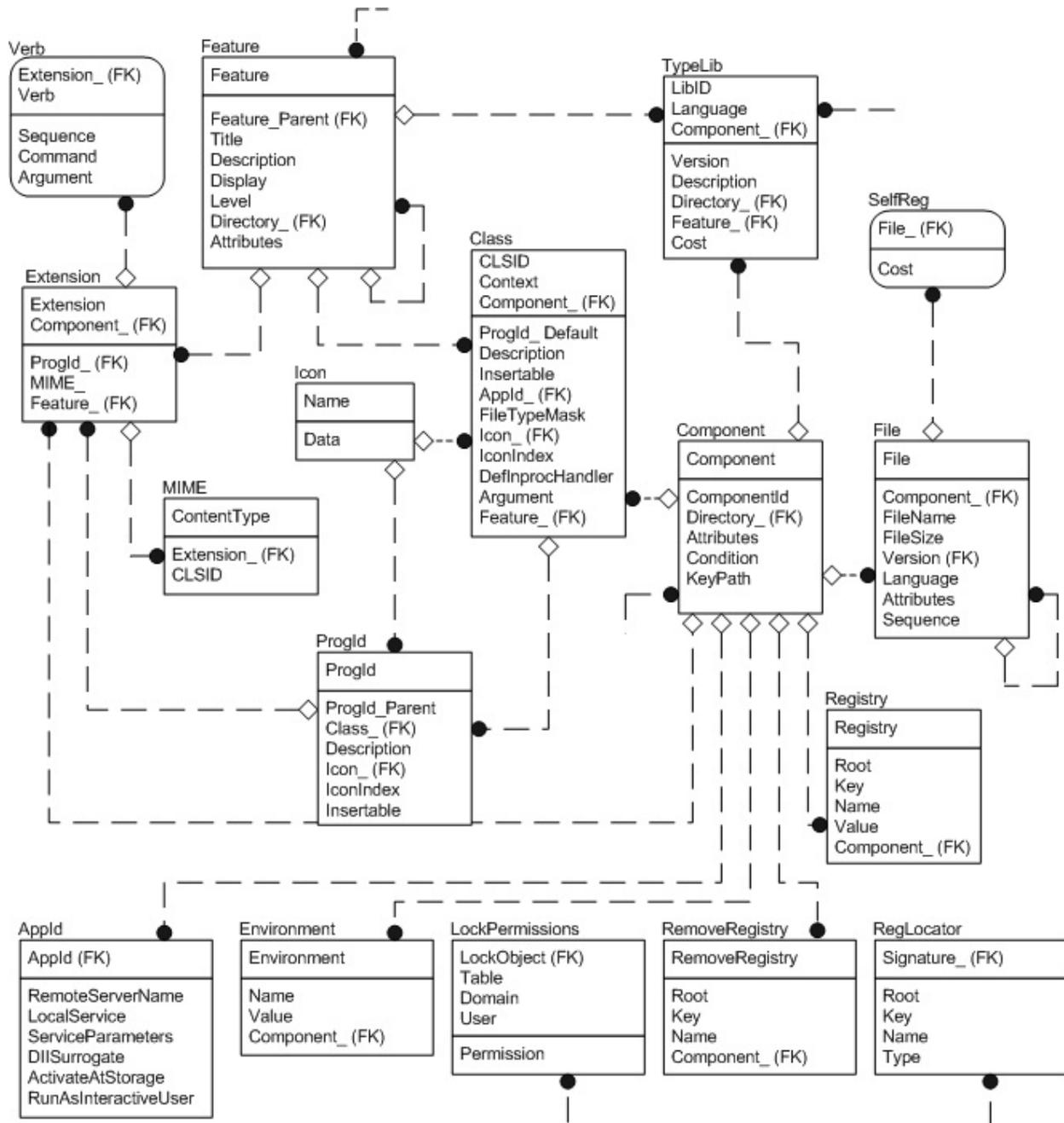
- The [MsiFileHash table](#) is used to store a 128-bit hash of a source file provided by the Windows Installer package.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Registry Tables Group



For more information about this diagram, see the [entity relationship diagram legend](#).

The installer has specific tables for the different types of registry entries. When populating the registry tables group it is important to try to minimize the number of entries put into the [Registry table](#) and maximize the use of the other, specific, registry tables. This is because the installer cannot distinguish between different types of registry entries in the Registry table and cannot use the internal logic necessary to take full advantage of all of the installer features, such as [advertising](#). Authoring COM and shell-related registry entries in this way also provides a more logical organization and can help minimize erroneous registration of COM server information.

The figure shows the registry entry group of tables as well as the [Component table](#), [Feature table](#), and [File table](#). Although the latter are not directly involved with populating the registry, they are included in the figure because they are essential to the logic of the registry entry group.

The registry entry group contain the following tables of specific registry entries.

- The [Extension table](#) contains all of the filename extensions your application uses along with their associated features and components.
- The [Verb table](#) associates command-verb information with the file name extensions listed in the [Extension table](#). This provides an indirect link between the Verb and Feature table that is needed for feature advertisement.
- The [TypeLib table](#) provides information that the installer places in the registry for the registration of type libraries. Type library entries are not written at the time of advertisement. The installer writes the type library entries at the time the components associated with the library are installed.
- The [MIME table](#) associates a MIME context type with a CLSID or a file name extension. This provides a path between the MIME and Feature Table that is needed for feature advertisement.
- The [SelfReg table](#) provides information needed to self-register modules. Self-registration is provided by the installer only for

backward compatibility and it is not recommended as a method for populating the registry, however if there are any modules in your application that must register themselves, use the SelfReg table.

- The [Class table](#) is used to register Class IDs and other information for COM objects. This table contains COM server-related information that must be generated as a part of the product advertisement.
- The [ProgId table](#) associates program IDs with class IDs.
- The [AppId table](#) is used to register common security and configuration settings for DCOM objects.
- The [Environment table](#) is used to set the values of environment variables, and in Windows 2000, the Environment table writes to the registry as well.
- The [Registry table](#) holds any other information that the application needs to put into the system registry. This would include default settings, user information or data, or COM registration not supported by the above tables.
- The [RemoveRegistry table](#) contains the registry information the application needs to delete from the system registry at installation time.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# System Tables Group

The tables of the system tables group track the tables and columns of the installation database.

- The [\\_Tables table](#) tracks all the tables in the database. This includes tables that you may have created for your own custom actions. Query this table to find out if a table exists.
- The [\\_Columns table](#) tracks columns in the installation database. Temporary columns are currently not tracked by this table. Query this table to find out if a given column exists.
- The [\\_Streams table](#) lists embedded OLE data streams.
- The [\\_Storages table](#) lists embedded OLE data storages.
- The [\\_Validation table](#). The [\\_Validation](#) table tracks the types and allowed ranges of every column in the database. The [\\_Validation](#) table is used during the database validation process to ensure that all columns are accounted for and have the correct values. This table is not shipped with the installer database.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Locator Tables Group

The Locator Tables group is used to locate files and applications. To search for a file, first determine the file signature and then locate the file. The Locator tables are used to search the registry, installer configuration data, directory tree, or .ini files for the unique signature of a file. The file signature can then be checked in the [Signature table](#) to ascertain that a particular file is really the file being sought and not another file with the same name. If a record in a locator table does not contain a key into the Signature table, then the record refers to a directory and not a file.

The component controlling a file is found in the [File table](#) through the external key to the [Component table](#). The installer resolves the location of a file through the Component table because every file belongs to one component. The location of a component is found through an external key in the Component table to the [Directory table](#).

The location of an application is found by searching for files that make up the application. The installer also provides two tables for searching for previous versions of an application: the [AppSearch table](#) and the [CCPSearch table](#).

The following tables make up the Locator tables group and are used to determine the file signature.

- The [RegLocator table](#) holds the information needed to search for a file or directory in the registry.
- The [IniLocator table](#) holds the information needed to search for a .ini file. The .ini file must be present in the default Microsoft Windows directory.
- The [CompLocator table](#) holds the information needed to search for a file or a directory using the installer's configuration data.
- The [DrLocator table](#) holds the information needed to search for a file or directory in the directory tree.
- The [AppSearch table](#) contains the properties that must be set to the search result of a corresponding file signature.

- The [CCPSearch table](#) contains the list of file signatures, at least one of which needs to be present on a user's computer for the Compliance Checking Program (CCP).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Program Information Tables Group

The Program Information Tables group contain important information that used throughout the installation.

- The [Property table](#) provides a means to specify all of the [properties](#) of an installation.
- The [Binary table](#) holds the binary data for items such as bitmaps, animations, and icons. The binary table is also used to store data for custom actions. This table can contain a bitmap for a billboard, the icon for your program, or the executable form of a custom action.
- The [Error table](#) is used to look up error message formatting templates when processing errors with an error code set. This is for the usual case where there is no formatting template set. The installer has its own error processing mechanism. Errors are passed as records.
- [Shortcut table](#) the shortcut table holds the information the application needs to create Shortcuts on the user's computer.
- The [ReserveCost table](#) contains the disk space necessary for each component to work properly.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installation Procedure Tables Group

The tables in the Installation Procedure group control tasks performed during the installation by [standard actions](#) and [custom actions](#).

Some of the tables in this group control a high level action by providing a sequence of actions. Each of the following sequence tables controls a portion of a high level action.

- [InstallUISequence table](#)
- [InstallExecuteSequence table](#)
- [AdminUISequence table](#)
- [AdminExecuteSequence table](#)
- [AdvtUISequence table](#)
- [AdvtExecuteSequence table](#)

There may be situations in which an installation needs to do something that is not possible using only [standard actions](#). To provide the greatest degree of flexibility, the installer provides setup authors the ability to create their own custom actions. If you have any custom actions, you should register them with the installer by populating the CustomAction Table.

The [CustomAction table](#) provides the means of integrating custom code and data into the installation process. The code that is executed can be a stream contained within the database, a recently installed file, or an existing executable.

The following tables extend the installer's capabilities to manipulate files and folders during the installation.

- The [RemoveFile table](#) contains a list of files that are removed during the installation.
- The [RemoveIniFile table](#) contains the information an application needs to remove from .ini files.
- The [RemoveRegistry table](#) contains the information which is deleted from the system registry when the corresponding component is

selected to be installed.

- The [CreateFolder table](#) lists the folders that must be created during the installation. Although the installer creates folders as they are needed, these are removed as soon as they are empty. Folders list in the CreateFolder table are not deleted until the component is uninstalled.
- The [MoveFile table](#) contains a list of files to be moved or copied from a specified source directory on the user's computer to a destination directory. It is not necessary to use the MoveFile table to describe the files associated with the components you are installing.

To set up necessary conditions that must be met to initiate the installation, populate the LaunchCondition table.

The [LaunchCondition table](#) contains a list of conditions, all of which must be satisfied for the action to succeed.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Entity Relationship Diagram Legend

A black circle leading to an open diamond indicates a one-to-many relationship between the primary key of the first table and the foreign key of the second table. For an example of this look at the figure for the [Registry Tables Group](#). Extension is a primary key of the Extension table and a foreign key of the Verb table. One Extension can therefore be registered to have several Verbs, but a particular Verb can only be associated to one Extension.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Text Archive Files

The Windows Installer database tables can be exported to ASCII text files by using [MsiDatabaseExport](#) or the [Export](#) method of the [Database](#) object. The information in these text archive files can then be imported back into a Windows Installer database using [MsiDatabaseImport](#) or the [Import](#) method of the [Database](#) object.

Tools such as [msidb.exe](#) are capable of exporting and importing text archive files. See [Export Files](#) and [Import Files](#) for [Windows Installer Scripting Examples](#) that can export and import text archive files from a database.

**Note** Text archive files are not intended as a means to edit the installation database. You should use a Windows Installer table editing tool, such as [Orca](#) or a third-party tool, to create and modify an installation package.

Text archive files can be used for the following purposes.

- Text archive files can be used with version control systems.
- To remove wasted storage space and reduce the final size of .msi files. See [Reducing the Size of an .msi File](#).
- To add localization information to an installation database. For more information, see [Code Page Handling of Imported and Exported Tables](#).
- To determine the code page of a database. See [Determining an Installation Database's Code Page](#).
- To set the code page of a database. See [Setting the code page of a database](#).
- To increase the limit of a database column. Export the table using [MsiDatabaseExport](#), edit the exported .idt file, and then import the table using [MsiDatabaseImport](#). Authors cannot change the column data types, nullability, or localization attributes of any columns in standard tables. See also [Authoring a Large Package](#).

The following pages describe text archive pages and their formats.

- [Archive File Format](#)
- [ASCII Data in Text Archive Files](#)
- [\\_ForceCodepage](#)
- [\\_SummaryInformation](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Archive File Format

A [text archive file](#) for a Windows Installer database carries an .idt file name extension. When an entire database is exported to archive files, each table in the database has a separate .idt file. If a table contains a stream column, each stream in the table is represented by a file with an .ibd file name extension. The .ibd files are stored in a folder with the same name as the table.

## .idt File Format

The .idt file of an exported database table that contains only ASCII characters has the following basic format.

- The first row contains the table column names separated by tabs.
- The second row contains the column definitions separated by tabs.
- If the file contain only ASCII data, the third row is table name and primary key column names separated by tabs.
- The remaining rows in the file represent rows in the table, with columns separated by tabs.

**Note** If the file contains non-ASCII data, the third row is the numeric code page followed by the table name and primary key column names separated by tabs. An .idt file that contains non-ASCII information should be saved in the ASCII format. For example, a text archive file can contain the column and table names encoded as UTF-8, but the archive file itself should be ASCII. See the section [ASCII Data in Text Archive Files](#).

**Note** The special [\\_ForceCodepage](#) and [\\_SummaryInformation](#) .idt files use extended formats. See the [\\_ForceCodepage](#) and [\\_SummaryInformation](#) sections for descriptions of their formats.

## Column Definitions

Column definitions are indicated by characters.

- The first character indicates the column type. A lowercase letter

indicates a non-nullable column and an uppercase letter indicates that the column can contain null values.

Character	Meaning
s, S	String Column
l, L	Localizable String Column
v, V	Binary Column
i, I	Integer Column

- The second character indicates the column data size.  
**Note** The Windows Installer does not actually use the specified column size to limit the size of the string that can be entered into a string column field. However, some authoring tools do use the specified column size to limit the size of a valid string. It is recommended that strings entered into any column meet the specified size requirement.

Column Definition	Meaning
s255	Non-Nullable String Column 255 long
L50	Nullable Localizable String Column 50 long
i2, I2	Short Integer Column
i4, I4	Long Integer Column

## Control Character Translation

Exporting a table to a text archive file translates the control characters to avoid conflicts with file delimiters. While writing into the .idt file, the control characters are translated as follows.

Control Character	Translation in .idt	Meaning

NULL	21	Null
BS	27	Back Space
HT	16	Tab
LF	25	Line Feed
FF	24	Form Feed
CR	17	Carriage Return

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ASCII Data in Text Archive Files

When a table that contains only ASCII characters is exported to a [text archive file](#), the .idt file adheres to the basic [archive file format](#). If the table contains non-ASCII information, the format of the archive file is extended to include code page information.

## Text archive files that contain only ASCII characters

When a table that contains only ASCII characters is exported to an archive file, the .idt file is in the basic archive file format. Each stream in the table is exported as a file with an .ibd file name extension. The .ibd files are stored in a folder with the same name as the table. For example, consider the export of the following [Binary](#) table.

Name	Data
Books	Books.ibd
Cars	Cars.ibd

The directory structure after exporting this table is as follows. The information in the database table is exported to Binary.idt. The two streams of binary data are exported to Book.ibd and Cars.ibd saved in the folder named Binary.

```
Binary.idt
[Binary]
    Books.ibd
    Cars.ibd
```

The Binary.idt archive file is in the basic archive file format and would look as follows.

```
Name    Data
s72     v0
Binary  Name
Books   Books.ibd
Cars    Cars.ibd
```

## Text archive files that contain non- ASCII characters

If the file contains non-ASCII data, the basic archive file format of the .idt file is extended to include code page information. The third row in the .idt table is the numeric code page followed by the table name and primary key column names separated by tabs.

**Note** An .idt file that contains non-ASCII information should be saved in the ASCII format. For example, a text archive file can contain the column and table names encoded as UTF-8, but the archive file itself should be ASCII.

The following ActionText table localized into French would contain non-ASCII information. The numeric code page used for French strings is 1252.

Action	Description	Template
ADVERTISE	Publication d'informations sur l'application	

The exported archive file, ActionText.idt, would be as follows.

```
Action  Description      Template
s72     L0      L0
1252    ActionText      Action
Advertise      Publication d'informations sur l'application
```

**Note** If a text archive file contains non-ASCII data, the archive file includes code page information. Archive files with code page information can only be imported back into a database of that exact code page or into a language neutral database. In the case of a language neutral database, the code page is set to the code page of the archive file. For more information about how Windows Installer handles code pages see the section [Code Page Handling \(Windows Installer\)](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# **`_ForceCodepage`**

The `_ForceCodepage` table is a special table used for changing the code page of an installation package. You can determine or set the code page by exporting or importing a [text archive file](#) named `_ForceCodepage.idt`.

The format of the `_ForceCodepage` table is 2 blank lines followed by a third line that indicates the numeric code page. For example, a `_ForceCodepage` table `.idt` file for a Japanese database would look as follows. The numeric code page in this case is 932.

```
<- this line blank
<- this line blank
932      _ForceCodepage
```

For more information on how to use `_ForceCodepage` to get or set the code page of a database see the following topics.

- [Code Page Handling \(Windows Installer\)](#)
- [Determining an Installation Database's Code Page](#)
- [Setting the Code Page of a Database](#)

The `_ForceCodepage` table is a special pseudo table used for changing the code page of an installation package. Using the `_ForceCodepage` table unconditionally sets the database to the code page without performing any validation as to whether the data currently in the database can be translated to the new code page. It is always recommended that changing the code page of a database start with a language neutral database.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# **\_SummaryInformation**

The `_SummaryInformation` table is a special table used with the [Summary Information Stream](#). You can get or set the Summary Information Stream of a Windows Installer database by exporting or importing a [text archive file](#) named `_SummaryInformation.idt`.

The `.idt` file of an exported `_SummaryInformation` table has the following format.

- The first row contains the table column names separated by tabs: `PropertyId` and `Value`. See the [Summary Information Stream Property Set](#) topic for a list of the properties and their ids (PID).
- The second row contains the column definitions separated by tabs. The column definitions are specified in the same way as in the basic `.idt archive file format`. The `PropertyId` column can be a non-nullable short integer. The `Value` column can be a non-nullable localizable string 255 characters long.
- The third row is the table name and the primary key column name separated by tabs: `_SummaryInformation` and `PropertyId`.
- The remaining rows in the file represent the PID and associated value, separated by tabs. Date and time in `_SummaryInformation` are in the format: `YYYY/MM/DD hh::mm::ss`. For example, `1999/03/22 15:25:45`.

The following is an example of the Summary Information Stream of a database in `.idt` format.

```
PropertyId      Value
i2              1255
_SummaryInformation  PropertyId
1              1252
2              Installation Database
3              Internal Quick Test
4              Microsoft Corporation
5              Installer,MSI,Database
```

```
6      Installer Internal Release Quick Test
7      Intel;1033
9      {00000002-0001-0000-0000-624474736554}
12     1999/06/21
14     110
15     1
18     Windows Installer
```

When you use [MsiDatabaseImport](#) or the [Import](#) method of the [Database](#) object to import a text archive table named `_SummaryInformation` into an installer database, you write the "05SummaryInformation" stream in the database.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using the Installer Database

The [Installer Database](#) enables the developer of an installation package to control the transfer of an application from a source to a target image. The layout of both the source and target images of the application are specified in the [Directory](#) table and the actions that install the application are specified in six sequence tables:

- [InstallUISequence](#) table
- [InstallExecuteSequence](#) table
- [AdminUISequence](#) table
- [AdminExecuteSequence](#) table
- [AdvtUISequence](#) table
- [AdvtExecuteSequence](#) table

The following sections describe how to use the Installer Database:

- [Using the Directory Table](#)
- [Using a Sequence Table](#)
- [Obtaining a Database Handle](#)
- [Committing Databases](#)
- [Importing and Exporting](#)
- [Merging Databases](#)
- [Naming Custom Tables, Properties, and Actions](#)
- [OLE Limitations on Streams](#)
- [Working with Queries](#)
- [Adding Binary Data to a Table Using SQL](#)
- [Working with Records](#)
- [Working with Folder Locations](#)
- [Specifying the Order of Self Registration](#)
- [Conditioning Actions to Run During Removal](#)

- [Calling Database Functions from Programs](#)
- [Conditional Statement Syntax](#)
- [Column Definition Format](#)
- [Determining Whether a Column is a Primary or External Key](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using the Directory Table

The [Directory table](#) specifies the layout of an installation. When the directories are resolved during the [CostFinalize action](#), the keys in the Directory table become [properties](#) set to directory paths. Note that the installer sets a number of standard [properties](#) to system folder paths. See the [Property Reference](#) for a list of the properties that are set to system folders.

The best way to specify the target location for a directory is by authoring the Directory table in your installation package to provide the correct location as discussed in this section. If it is necessary to change the directory location at the time of the installation see also the section: [Changing the Target Location for a Directory](#)

The following is an example of a Directory table.

Directory	Directory_Parent	DefaultDir
TARGETDIR		SourceDir
EXEDIR	TARGETDIR	App
DLLDIR	EXEDIR	Bin
DesktopFolder	TARGETDIR	Desktop

Each row of the Directory table indicates a directory both at the source and the target. For example, assume the installation package resides at `\\applications\source\`. Because the `Directory_Parent` field of the first row is Null, this record indicates the root directories for both the source and the target. For the source, the value of this directory is given by the `DefaultDir` field. The [SourceDir](#) property defaults to the location of the installation package. Thus, unless the **SourceDir** property is overridden, the root source directory is `\\applications\source\`.

The `Directory` field of the first record indicates the location of the root target directory. In this case, the value of the [TARGETDIR](#) property indicates this directory. Typically, the value of the **TARGETDIR** property is set at the command line or through a user interface. In this case, assume the **TARGETDIR** property is set to `C:\Program Files\Target\`.

For the second record, the Directory\_Parent field is not Null. Therefore, this record indicates a non-root directory for both the source and the target. For a non-root source directory, the source directory indicated by the record described in the Directory\_Parent field is the parent directory. For the second record, the Directory\_Parent field is TARGETDIR. As shown earlier, the source directory indicated by the TARGETDIR record resolved to \\applications\source\. Thus, the source directory indicated by the second record is \\applications\source\App\.

A similar process works for the target directory. The value of the parent directory for the target directory described in the second record is the target directory resolved by the Directory\_Parent field. Again, the Directory\_Parent field contains the value TARGETDIR. This indicates the first record that resolves to a target directory of C:\Program Files\Target\. The Directory field contains an author-defined property called EXEDIR. If this property is set, then its value gives the full path of the directory. Thus, if this property is set to C:\Data\Common\, the value of the target directory indicated by the second record is C:\Data\Common\. If it is not set, the target directory takes the name given by the DefaultDir field. In this case, the target directory is C:\Program Files\Target\App\.

The same process works for the third record. If EXEDIR and DLLDIR are not set, the target directory is C:\Program Files\Target\App\Bin, and the source directory is \\applications\source\App\Bin\.

The fourth record uses the **DesktopFolder** property. If the location of the user's desktop is C:\Winnt\Profiles\User\Desktop\, the target directory resolves to C:\Winnt\Profiles\User\Desktop\. The source directory resolves to \\applications\source\Desktop\.

There are two additional syntax features that can be used in the DefaultDir column of the Directory table. For a non-root source directory, a period (.) entered in the DefaultDir column indicates that the directory should be located in its parent directory without a subdirectory. To specify different source and target directory paths, separate the target and source paths in the DefaultDir column with a colon as follows: [targetpath]:[sourcepath]. These features can be used together to add levels to either the source or target paths for a single directory. See the following example of a Directory table.

Directory	Directory_Parent	DefaultDir
-----------	------------------	------------

TARGETDIR		SourceDir
MyAppDir	TARGETDIR	MyApp
BinDir	MyAppDir	Bin
Binx86Dir	BinDir	.:x86
BinAlphaDir	BinDir	.:Alpha

The source and target paths resolve for the MyAppDir, BinDir, Binx86Dir, and BinAlphaDir rows as follows.

<b>Record</b>	<b>Target paths</b>	<b>Source paths</b>
MyAppDir:	[TARGETDIR]MyApp	[SourceDir]MyApp
BinDir:	[TARGETDIR]MyApp\Bin	[SourceDir]MyApp\Bin
Binx86Dir:	[TARGETDIR]MyApp\Bin	[SourceDir]MyApp\Bin\x86
BinAlphaDir:	[TARGETDIR]MyApp\Bin	[SourceDir]MyApp\Bin\Alpha

**Note** The Alpha platform is not supported by the Windows Installer.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using a Sequence Table

The authoring of the sequence tables is an essential part of developing an installer package because these tables specify the order of execution for the [standard actions](#) that control the installation process and display the user interface dialog boxes.

There are three modes of installation and two types of sequence tables for each mode.

The three separate installation modes currently supported by the installer are:

- Simple Installation
- Administrative Installation
- Advertisement Installation

The sequence tables each have three fields: Action, Condition, and Sequence. The Action field names either a standard or custom action or a user defined dialog box or sequence the installer executes. The Condition field allows the author to specify a logical expression that controls whether an action or user-defined dialog is executed or displayed. If the Condition field is blank or contains an expression that evaluates to True, the action or dialog is executed or displayed. The action or dialog is skipped if the expression evaluates to False. The Sequence field specifies the order of execution of each action or user-defined dialog in the table.

Each of these installation modes processes the user interface sequence tables and the execute sequence tables. The user interface sequence tables are only processed if the installer was initialized with the user interface display level set to Reduced or Full. See the [MsiSetInternalUI](#) reference for more information about user interface display levels.

The user interface sequence tables typically contains standard actions related to collecting system information that are displayed to the user through the user interface. The user interface is displayed by recording the foreign keys to the names of dialog boxes in the [dialog table](#) in the Action field of the user interface sequence table. The user then has the opportunity to modify or accept the system information and begin the

installation, which occurs when the execute sequence table is processed.

During a simple installation, the [INSTALL](#) top-level action is executed which in turn processes the [InstallUISequence table](#) and the [InstallExecuteSequence table](#).

An Administrative Installation is typically initiated by a network administrator to assign and install applications for individual users and groups of users. During this type of installation, the [ADMIN](#) top-level action is executed which processes the [AdminUISequence table](#) and the [AdminExecuteSequence table](#).

To [advertise](#) an application or feature, the installer must be initiated with the [ADVERTISE](#) action. During this type of installation the [AdvtExecuteSequence table](#) is processed.

When authoring any sequence table, it is good practice to use the sequence number for standard actions from the suggested sequences in the topics below. For standard actions which have no standard position in the sequence table such as [ForceReboot](#), [ValidateProductID](#), and [InstallExecute](#), use a sequence number that is a multiple of ten to identify the action as a standard action. For custom actions, use a sequence number that is not a multiple of ten to differentiate it from standard actions in the sequence table.

For suggested action sequences for each sequence table, see the following topics:

- [Suggested InstallUISequence](#)
- [Suggested InstallExecuteSequence](#)
- [Suggested AdminUISequence](#)
- [Suggested AdminExecuteSequence](#)
- [Suggested AdvtUISequence](#)
- [Suggested AdvtExecuteSequence](#)

For a detailed description of sequence tables and how standard actions are executed, see the [sequence table detailed example](#).

### **Windows Installer 3.0 and later:**

Beginning with Windows Installer 3.0, a patch package can contain

the [MsiPatchSequence table](#). This table contains all the information the installer requires to determine the sequence of the application of a small update patch relative to all other patches. For more information, see [Patching and Upgrades](#).

## Note

[Merge Modules](#) may contain [Merge Module Database Tables](#) that modify the action sequence tables of the target .msi file. Merging the module into a database can modify the information in the sequence table, but does not add these tables to the .msi file. For more information, see [Authoring Merge Module Sequence Tables](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Sequence Table Detailed Example

Here is an example of a sequence table.

Action	Condition	Sequence
LaunchConditions		
AppSearch		200
CCPSearch	CCP_TEST	300
CCPDialog	NOT_CCP_SUCCESS	400
MyCustomConfig	NOT <b>Installed</b>	500
CostInitialize		600
FileCost		700
CostFinalize		800
InstallDialog	NOT <b>Installed</b>	900
MaintenanceDialog	<b>Installed</b> AND NOT <b>Resume</b>	1000
ActionDialog		1100
RegisterProduct		1200
InstallValidate		1300
InstallFiles		1400
MyCustomAction	\$MyComponent > 2	1500
InstallFinalize		1600

The following actions in this sequence table are defined by the installer and are examples of standard actions:

LaunchConditionsAppSearch  
CCPSearch  
CostInitialize  
FileCost  
CostFinalize

RegisterProduct  
InstallFiles  
InstallFiles  
InstallValidate

The following actions were defined by the table's author and are examples of [custom actions](#) and must be listed in the [CustomAction table](#):

MyCustomConfig  
MyCustomAction

The remaining entries in the Action field are foreign keys into the [Dialog table](#). They specify the names of dialog boxes that will be displayed if the condition field evaluates to True.

CCPDialog  
InstallDialog  
MaintenanceDialog  
ActionDialog

The Condition column causes the installer to skip the action if the property or expression in this field is False. The [Installed](#) property and the [RESUME](#) property are examples of properties that are set by the installer. The [Installed](#) property is set to true if the product is already installed and the [RESUME](#) property is set if resuming a suspended installation. The CCP\_TEST and the NOT\_CCP\_SUCCESS properties are examples of properties that can be set at the command line by the user installing the application.

All actions run in sequence with the following conditional steps:

- The CPPSearch is run only if CCP\_TEST is set.
- CCPDialog is run only if NOT\_CCP\_SUCCESS is set.
- MaintenanceDialog is run only if this product is already installed and if this is not an installation that is being resumed after being suspended.
- MyCustomAction is run only if the expression in the Condition column is True. The expression \$MyComponent > 2 refers to the action state of the component called MyComponent. This condition

indicates that MyCustomAction should only be run if MyComponent is set to be installed. For more information on Action states and Selection states, see the [FeatureRequestState](#) property, the [Feature table](#), and the [InstallFiles](#) action.

## See Also

[Using Properties](#)  
[Conditional Statement Syntax](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Suggested InstallUISequence

Action	Condition	Sequence
FatalErrorDlg		-3
UserExitDlg		-2
ExitDlg		-1
LaunchConditions		100
PrepareDlg		140
AppSearch		400
CCPSearch	NOT <b>Installed</b>	500
RMCCPSearch	NOT <b>Installed</b>	600
CostInitialize		800
FileCost		900
CostFinalize		1000
WelcomeDlg	NOT <b>Installed</b>	1230
ResumeDlg	<b>Installed</b> AND ( <b>RESUME</b> OR <b>Preselected</b> )	1240
MaintenanceWelcomeDlg	<b>Installed</b> AND NOT <b>RESUME</b> AND NOT <b>Preselected</b>	1250
ProgressDlg		1280
ExecuteAction		1300

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Suggested InstallExecuteSequence

Action	Condition	Sequence
LaunchConditions		100
AppSearch		400
CCPSearch	NOT <b>Installed</b>	500
RMCCPSearch	NOT <b>Installed</b>	600
ValidateProductID		700
CostInitialize		800
FileCost		900
CostFinalize		1000
SetODBCFolders		1100
InstallValidate		1400
InstallInitialize		1500
AllocateRegistrySpace	NOT <b>Installed</b>	1550
ProcessComponents		1600
UnpublishComponents		1700
UnpublishFeatures		1800
StopServices	<b>VersionNT</b>	1900
DeleteServices	<b>VersionNT</b>	2000
UnregisterComPlus		2100
SelfUnregModules		2200
UnregisterTypeLibraries		2300
RemoveODBC		2400
UnregisterFonts		2500
RemoveRegistryValues		2600

UnregisterClassInfo	2700
UnregisterExtensionInfo	2800
UnregisterProgIdInfo	2900
UnregisterMIMEInfo	3000
RemoveIniValues	3100
RemoveShortcuts	3200
RemoveEnvironmentStrings	3300
RemoveDuplicateFiles	3400
RemoveFiles	3500
RemoveFolders	3600
CreateFolders	3700
MoveFiles	3800
InstallFiles	4000
PatchFiles	4090
DuplicateFiles	4210
BindImage	4300
CreateShortcuts	4500
RegisterClassInfo	4600
RegisterExtensionInfo	4700
RegisterProgIdInfo	4800
RegisterMIMEInfo	4900
WriteRegistryValues	5000
WriteIniValues	5100
WriteEnvironmentStrings	5200
RegisterFonts	5300
InstallODBC	5400

RegisterTypeLibraries		5500
SelfRegModules		5600
RegisterComPlus		5700
InstallServices	<b>VersionNT</b>	5800
StartServices	<b>VersionNT</b>	5900
RegisterUser		6000
RegisterProduct		6100
PublishComponents		6200
PublishFeatures		6300
PublishProduct		6400
InstallFinalize		6600

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Suggested AdminUISequence

Action	Condition	Sequence
FatalErrorDlg		-3
UserExitDlg		-2
ExitDlg		-1
PrepareDlg		140
CostInitialize		800
FileCost		900
CostFinalize		1000
AdminWelcomeDlg		1230
ProgressDlg		1280
ExecuteAction		1300

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Suggested AdminExecuteSequence

Action	Condition	Sequence
CostInitialize		800
FileCost		900
CostFinalize		1000
InstallValidate		1400
InstallInitialize		1500
InstallAdminPackage		3900
InstallFiles		4000
InstallFinalize		6600

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Suggested AdvtUISequence

Do not author the AdvtUISequence table. The installer does not use this table. The AdvtUISequence table should not exist in the installation database or it should be left empty. The name of the application to be advertised should be passed in to the installer during initialization.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Suggested AdvtExecuteSequence

Action	Condition	Sequence
CostInitialize		800
CostFinalize		1000
InstallValidate		1400
InstallInitialize		1500
CreateShortcuts		4500
RegisterClassInfo		4600
RegisterExtensionInfo		4700
RegisterProgIdInfo		4800
RegisterMIMEInfo		4900
PublishComponents		6200
PublishFeatures		6300
PublishProduct		6400
InstallFinalize		6600

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Obtaining a Database Handle

Before working with a database you must first obtain a handle to it.

## ▶ To access information about an installer database

1. Obtain a handle to the database in one of two ways:
  - If an installation is in progress, get a handle to the active database by calling the **MsiGetActiveDatabase** function.
  - If an installation is not in progress, open any specified database by calling the **MsiOpenDatabase** function.
2. After the database has been opened, you can call functions to obtain information about the database or to manipulate the database.
  - Create a **View** object and specify a SQL query of the open database by calling the **MsiDatabaseOpenView** function.
  - Obtain a record that contains all primary keys of a specified table in the open database by calling the **MsiDatabaseGetPrimaryKeys** function.
  - Check the current state of an open database by calling the **MsiGetDatabaseState** function. With the **MsiGetDatabaseState** function, you can determine the read/write status for a database or if the handle is valid.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Committing Databases

Changes made to the installation database are not written to the database until you call **MsiDatabaseCommit**.

▶ **To ensure changes made in a database are finalized**

1. Check to see whether a table will be written when you call **MsiDatabaseCommit** by calling **MsiDatabasesTablePersistent**.
2. Call the **MsiDatabaseCommit** function to finalize changes to the database.

Changes made in a database are accumulated and are not reflected in the actual database until you call **MsiDatabaseCommit**. Temporary columns or rows are not committed to the database. When a database is closed, all changes made since the last **MsiDatabaseCommit** are automatically rolled back.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Importing and Exporting

You can use the installer to import a text archive into an active database as well as to export a database file to a text archive. This can be useful for text-based source control systems.

To import a text archive into an active database, call the [MsiDatabaseImport](#) function.

To export a database file to a text archive, call the [MsiDatabaseExport](#) function.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Merging Databases

You can use the installer to add the information in one database into another database by performing a merge. [Merges and Transforms](#) operate on an entire database, and a merge combines two databases into one. Merges are useful to development teams because they allow the installation database of large application to be divided into smaller parts and then recombined into the complete installation database later.

## ▶ To merge several component databases into a single complete database

1. Develop the partial component databases separately.
2. Merge each component database into the main product database by calling the [MsiDatabaseMerge](#) function.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Naming Custom Tables, Properties, and Actions

Package authors should ensure that the names of any custom tables, properties, or actions defined in their package will not conflict with the names of standard tables, properties, or actions that are native to the Windows Installer.

Package authors should adhere to the following guidelines for custom table, property, or action names.

- Make names for custom tables, properties, or actions that have a prefix that identifies your application.
- Never use a name with Msi as the prefix. Starting with Windows Installer 2.0, all new native Windows Installer properties, actions, and tables are given names prefixed by Msi. This prefix is not case sensitive, for example MsiNewInternalTable. Because the prefix is not case sensitive, authors should avoid all cases of the Msi prefix.

For more information, see [Table Names](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Table Names

Table names are limited to 31 characters. If a table name contains a period (a practice that is permitted but not recommended), the name must be enclosed within grave accent marks ` (ASCII value 96), for example, `filename.ext`. This escape syntax prevents clashes between the table name and reserved words, such as SQL keywords. For more information, see [SQL Syntax](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# OLE Limitations on Streams

Developers of installation databases need to be aware of two limitations on the handling of streams by the Win32 OLE structured storage implementation. These limitations can affect installer functions indirectly through transforms and other data that may be stored in the database as a stream.

There are two relevant limitations:

- Binary data is stored with an index name created by concatenating the table name and the values of the record's primary keys using a period delimiter. OLE limits stream names to 32 characters (31 + null terminator). Windows Installer uses a compression algorithm that can expand the limit to 62 characters depending upon the character. Note that double-byte characters count as 2.
- Although you can have multiple streams open at one time, you cannot open a stream a second time until the first reference is closed. This means you cannot select the same binary data stream to be open in multiple records simultaneously. Attempts to read the binary data from the second record fail. Also you cannot rename the primary keys of a record while a binary data stream in that record is open.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Working with Queries

Because the installer uses a relational database, there are functions for making Structured Query Language (SQL) queries to the database. The following procedure describes how to use SQL to query a database.

## ► To query a database with SQL

1. Open the **View** object, with the appropriate SQL statement, by calling the **MsiDatabaseOpenView** function.

A **View** object is the logical table created by applying a query to a set of tables. SQL queries must adhere to the [SQL syntax](#) provided by the installer. This SQL statement can contain parameter markers that are not specified until the **View** object runs.

2. Run the **View** object by calling the **MsiViewExecute** function.
3. Retrieve the next record from a **View** object by calling the **MsiViewFetch** function.
4. Modify the **View** object by calling the **MsiViewModify** function. You can also validate data with **MsiViewModify** by passing the appropriate flags. If **MsiViewModify** returns `ERROR_INVALID_DATA` from a validation request, the underlying data is corrupt.
5. Obtain detailed error information on the **View** object by calling the **MsiViewGetError** function.
6. Close the **View** object by calling the **MsiViewClose** function.

For more information, see [Examples of Database Queries Using SQL and Script](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Examples of Database Queries Using SQL and Script

An example of using script-driven database queries is provided in the Windows [Installer Software Development Kit](#) (SDK) as the utility `WiRunSQL.vbs`. This utility handles database queries using the Windows Installer version of SQL described in the section [SQL Syntax](#).

## Delete a record from a table

The following command line deletes the record having the primary key `RED` from the [Feature](#) table of the `Test.msi` database.

```
Cscript WiRunSQL.vbs Test.msi "DELETE FROM `Feature` WHERE `Feature`.`Feature`='RED'"
```

## Add a table to a database

The following command line adds the [Directory](#) table to the `Test.msi` database.

```
CScript WiRunSQL.vbs Test.msi "CREATE TABLE `Directory` (`Directory` CHAR(72) NOT NULL, `Directory_Parent` CHAR(72), `DefaultDir` CHAR(255) NOT NULL LOCALIZABLE PRIMARY KEY `Directory`)"
```

## Remove a table from a database

The following command line removes the `Feature` table from the `Test.msi` database.

```
Cscript WiRunSQL.vbs Test.msi "DROP TABLE `Feature`"
```

## Add a new column to a table

The following command line adds the `Test` column to the [CustomAction](#) table of the `Test.msi` database.

```
CScript WiRunSQL.vbs Test.msi "ALTER TABLE `CustomAction` ADD `Test` INTEGER"
```

## Insert a new record into a table

The following command line inserts a new record into the `Feature` table of the `Test.msi` database.

```
Cscript WiRunSQL.vbs Test.msi "INSERT INTO `Feature`
(`Feature`.`Feature`,`Feature`.`Feature_Parent`,`Feature`.`Title`,`Featu
`Feature`.`Display`,`Feature`.`Level`,`Feature`.`Directory`,`Feature`.`
VALUES ('Tennis','Sport','Tennis','Tournament',25,3,'SPORTDIR',2)"
```

This inserts the following record into the Feature table of Test.msi.

Feature Table

Feature	Feature_Parent	Title	Description	Display	Level	Directory_	A
Tennis	Sport	Tennis	Tournament	25	3	SPORTDIR	2

Note that binary data cannot be inserted into a table directly using the INSERT INTO or UPDATE SQL queries. For information see [Adding Binary Data to a Table Using SQL](#).

### Modify an existing record in a table

The following command line changes the existing value in the Title field to "Performances." The updated record has "Arts" as its primary key and is in the Feature table of the Test.msi database.

```
Cscript WiRunSQL.vbs Test.msi "UPDATE `Feature` SET
`Feature`.`Title`='Performances' WHERE `Feature`.`Feature`='Arts'"
```

### Select a group of records

The following command line selects the name and type of all controls that belong to the ErrorDialog in the Test.msi database.

```
CScript WiRunSQL.vbs Test.msi "SELECT `Control`,`Type` FROM
`Control` WHERE `Dialog_`='ErrorDialog' "
```

### Hold a table in memory

The following command line locks the [Component](#) table of the Test.msi database in memory.

```
CScript WiRunSQL.vbs Test.msi "ALTER TABLE `Component`
HOLD"
```

### Free a table in memory

The following command line frees the Component table of the Test.msi database from memory.

**CScript WiRunSQL.vbs Test.msi "ALTER TABLE `Component`  
FREE"**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Working with Records

The installer supplies functions that manipulate the records in an installation database. These functions can be used in conjunction with the functions described in [Working with Queries](#) to make actual changes in a database.

The following functions create or remove records:

- To create a new record for a database, call the **MsiCreateRecord** function.
- To clear data from a record, set each field to null by calling the **MsiRecordClearData** function.

The following functions fill specified fields of records:

- To set a record to an integer, call the **MsiRecordSetInteger** function.
- To set a record to a string, call the **MsiRecordSetString** function.
- To insert an entire file into a stream field, call the **MsiRecordSetStream** function.

The following functions read values from specified fields of records:

- To read an integer value from a field, call the **MsiRecordGetInteger** function.
- To retrieve a string value, call the **MsiRecordGetString** function.
- To obtain a stream, call the **MsiRecordReadStream** function.
- To determine if a particular field of a record is null, call the **MsiRecordIsNull** function.

The following functions are informational record functions:

- To get the number of fields a record contains, call the **MsiRecordGetFieldCount** function.
- To get the size of a field, call the **MsiRecordDataSize** function. The

return value of **MsiRecordDataSize** is sensitive to the field type.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Adding Binary Data to a Table Using SQL

Binary data cannot be inserted into a table directly using the INSERT INTO or UPDATE SQL queries. In order to add binary data to a table, you must first use the parameter marker (?) in the query as a placeholder for the binary value. The execution of the query should include a record that contains the binary data in one of its fields.

A marker is a parameter reference to a value supplied by a record submitted with the query. It is represented in the SQL statement by a question mark (?).

The following sample code adds binary data to a table.

```
#include <windows.h>
#include <Msiquery.h>
#include <tchar.h>
#pragma comment(lib, "msi.lib")

int main()
{

PMSIHANDLE hDatabase = 0;
PMSIHANDLE hView = 0;
PMSIHANDLE hRec = 0;

if (ERROR_SUCCESS == MsiOpenDatabase(_T("c:\\temp\\testdb.msi"),
{
    //
    // Open view on Binary table so that we can add a record
    //

    if (ERROR_SUCCESS == MsiDatabaseOpenView(hDatabase,
    {

        //
        // Create record with binary data in 1st field
        //
```

```

        hRec = MsiCreateRecord(1);
        if (ERROR_SUCCESS == MsiRecordSetStream(hRec)
        {
            //
            // Execute view with record containing binary data
            //

            if (ERROR_SUCCESS == MsiViewExecute(
                && ERROR_SUCCESS == MsiViewClose(
                && ERROR_SUCCESS == MsiDatabaseCommit(
                {
                    //
                    // New binary data successfully added
                    //
                }
            }
        }
    }
}

return 0;
}

```

The following sample script adds binary data to a table.

```

Dim Installer
Dim Database
Dim View
Dim Record

Set Installer = CreateObject("WindowsInstaller.Installer")

Set Record = Installer.CreateRecord(1)
Record.SetStream 1, "c:\temp\data.bin"

Set Database = Installer.OpenDatabase("c:\temp\testdb.msi",
Set View = Database.OpenView("INSERT INTO `Binary` (`Name`,
View.Execute Record
Database.Commit ' save changes

```



## See Also

[Working with Queries](#)

[SQL Syntax](#)

[Examples of Database Queries Using SQL and Script](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Working with Folder Locations

Use the following functions to find the source or target of a specified folder in the [Directory table](#):

- Obtain the path of the specified source folder by calling the [MsiGetSourcePath](#) function.
- Obtain the path of the specified target folder by calling the [MsiGetTargetPath](#) function.
- Change the target location for a folder with the [MsiSetTargetPath](#) function.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Specifying the Order of Self Registration

Note that you cannot specify the order in which the installer registers or unregisters self-registering DLLs by using the [SelfRegModules](#) and [SelfUnRegModules](#) actions. These actions register all the modules listed in the [SelfReg table](#). The installer does not self-register .exe files.

To specify the order in which the installer registers or unregisters modules, you must use two [custom actions](#) for each module. One custom action for DllRegisterServer and a second for DllUnregisterServer. These custom actions must then be authored in the [InstallExecuteSequence table](#) at the point in the sequence wherever the DLL is to be registered or unregistered.

The following example illustrates how to author the database to schedule the self-registration of a DLL at a particular point in the action sequence.

## File Table (partial)

File	Component_	FileName	Sequence
mydll	myComponent	Mydll.dll	13

## Component Table (partial)

Component	ComponentId	Directory_	KeyPath
myComponent	{a GUID}	myFolder	mydll

## Directory Table

Directory	Directory_Parent	DefaultDir
TARGETDIR		SourceDir
myFolder	TARGETDIR	myFolder My Folder

## CustomAction Table

Action	Type	Source	Target
mydllREG	3170	myFolder	"[SystemFolder]msiexec" /y "[#mydll]"
mydllUNREG	3170	myFolder	"[SystemFolder]msiexec" /z "[#mydll]"

## InstallExecuteSequence Table (partial)

Action	Condition	Sequence
SelfUnregModules		2200
mydllUNREG	\$myComponent=2	2201
RemoveFiles		3500
InstallFiles		4000
SelfRegModules		6500
mydllREG	\$myComponent>2	6501

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Conditioning Actions to Run During Removal

There are two ways to author the installation database such that an action is only called when the package is uninstalled:

- If the action is sequenced after the [InstallValidate action](#) in the [InstallExecuteSequence table](#), the package author may specify a condition of REMOVE="ALL" for the action in the Condition column. Note that the **REMOVE** property is not guaranteed to be set to ALL during an uninstall before the installer executes the InstallValidate action. Note that the quote marks around the value ALL are required in this case.
- If the action is sequenced after the [CostFinalize action](#) and any actions that could change the feature state, such as [MigrateFeatureStates action](#), the action can be conditioned on the state of a particular feature or component. See [Conditional Statement Syntax](#). Use this option to call an action during the removal of a particular feature or component, which may occur outside of the complete removal of the application.

Note that the **Installed** property can be used in conditional expressions to determine whether a product is installed per-computer or for the current user. To determine whether the product is installed for a different user, check the **ProductState** property.

Note that older versions of a product may be removed during an upgrade by the [RemoveExistingProducts action](#). The [Upgrade table](#) may also set the **REMOVE** property to ALL in this case. To determine whether a product is being removed by an upgrade, check the **UPGRADINGPRODUCTCODE** property. The installer only sets this property when RemoveExistingProducts removes the product. The installer does not set the property during a normal uninstall, such as removal with Add/Remove programs.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Calling Database Functions from Programs

Before calling any of the following [Database Functions](#) from a program, such as a custom action or an automation process, the installer must first run the [CostInitialize action](#), [FileCost action](#), and [CostFinalize action](#).

The following is a list of database functions used in Windows Installer:

- [MsiGetComponentState](#)
- [MsiGetFeatureCost](#)
- [MsiGetFeatureState](#)
- [MsiGetFeatureValidStates](#)
- [MsiGetSourcePath](#)
- [MsiGetTargetPath](#)
- [MsiSetComponentState](#)
- [MsiSetFeatureState](#)
- [MsiSetInstallLevel](#)
- [MsiSetTargetPath](#)
- [MsiVerifyDiskSpace](#)

Before calling [MsiSetFeatureAttributes](#) from a program, the installer must first run the [CostInitialize action](#). The installer then runs the [CostFinalize action](#) after [MsiSetFeatureAttributes](#).

The following example illustrates the order in which function actions must be called when using [MsiGetTargetPath](#) in a program.

```
#include <windows.h>
#include <Msiquery.h>
#include <tchar.h>
#pragma comment(lib, "msi.lib")

int main()
{
```

```

MSIHANDLE hInstall;
TCHAR *szBuf;
DWORD cch = 0 ;

if(MsiOpenPackage(_T("PathToPackage..."), &hInstall) == ERROR_SUCCESS)
{
    if(MsiDoAction(hInstall, _T("CostInitialize"))==ERROR_SUCCESS &&
        MsiDoAction(hInstall, _T("FileCost"))==ERROR_SUCCESS &&
        MsiDoAction(hInstall, _T("CostFinalize"))==ERROR_SUCCESS)
    {
        if(MsiGetTargetPath(hInstall, _T("FolderName"), _T("FolderName"), &cch) == ERROR_SUCCESS)
        {
            cch++; // add 1 to include null terminator since we are using TCHAR
            szBuf = (TCHAR *) malloc(cch*sizeof(TCHAR));
            if(szBuf)
            {
                if(MsiGetTargetPath(hInstall, _T("FolderName"), _T("FolderName"), szBuf) == ERROR_SUCCESS)
                {
                    // Add code to use szBuf here
                }
                free(szBuf);
            }
        }
    }
    MsiCloseHandle(hInstall);
}

return 0;
}

```

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Conditional Statement Syntax

This section describes the syntax of conditional statements used by the [MsiEvaluateCondition](#) function and the action [sequence tables](#). For more information, see, [Examples of Conditional Statement Syntax](#).

## Summary of Conditional Statement Syntax

This table and the following list summarize the syntax to use in conditional expressions.

Item	Syntax
value	symbol   literal   integer
comparison-operator	<   >   <=   >=   =   <>
term	value   value comparison-operator value   ( expression )
Boolean-factor	term   <b>NOT</b> term
Boolean-term	Boolean-factor   Boolean-factor <b>AND</b> term
expression	Boolean-term   Boolean-term <b>OR</b> expression
symbol	property   %environment-variable   \$component-action   ? component-state   &feature-action   !feature-state

- Symbol names and values are case sensitive.
- Environment variable names are not case sensitive.
- Literal text must be enclosed between quotation marks ("text").  
**Note** Literal text containing quotation marks cannot be used in conditional statements because there is no escape character for quotation marks inside literal text. To do a comparison against literal text containing quotation marks, the literal text should be put in a

property. For example, to verify that the SERVERNAME property does not contain any quotation marks, define a property called QUOTES in the [Property table](#) with a value of " and change the condition to NOT SERVERNAME><QUOTES.

- Nonexistent property values are treated as empty strings.
- Floating point numeric values are not supported.
- Operators and precedence are the same as in the BASIC and SQL languages.
- Arithmetic operators are not supported.
- Parentheses can be used to override operator precedence.
- Operators are not case sensitive.
- For string comparisons, a tilde "~" prefixed to the operator performs a comparison that is not case sensitive.
- Comparison of an integer with a string or property value that cannot be converted to an integer is always **msiEvaluateConditionFalse**, except for the comparison operator "<>", which returns **msiEvaluateConditionTrue**.

## Access Prefixes

The following table shows the prefixes to use to access various system and installer information for use in conditional expressions.

Symbol type	Prefix	Value
Installer property	(none)	Value of property (Property) table.
Environment variable	%	Value of environment variable.
Component table key	\$	Action state of the component.
Component table key	?	Installed state of the component.
Feature table key	&	Action state of the feature.
Feature table key	!	Installed state of the feature.

## Logical Operators

The following table shows the logical operators in conditional expressions, in order of high-to-low precedence.

Operator	Meaning
Not	Prefix unary operator; inverts state of following term.
And	TRUE if both terms are TRUE.
Or	TRUE if either or both terms are TRUE.
Xor	TRUE if either but not both terms are TRUE.
Eqv	TRUE if both terms are TRUE or both terms are FALSE.
Imp	TRUE if left term is FALSE or right term is TRUE.

## Comparative Operators

The following table shows the comparison operators used in conditional expressions. These comparison operators can only occur between two values.

Operator	Meaning
=	TRUE if left value is equal to right value.
<>	TRUE if left value is not equal to right value.
>	TRUE if left value is greater than right value.
>=	TRUE if left value is greater than or equal to right value.
<	TRUE if left value is less than right value.
<=	TRUE if left value is less than or equal to right value.

## Substring Operators

The following table shows the substring operators used in conditional expressions. Substring operators can occur between two string values.

<b>Operator</b>	<b>Meaning</b>
><	TRUE if left string contains the right string.
<<	TRUE if left string starts with the right string.
>>	TRUE if left string ends with the right string.

## Bitwise Numeric Operators

The following table shows the bitwise numeric operators in conditional expressions. These operators can occur between two integer values.

<b>Operator</b>	<b>Meaning</b>
><	Bitwise AND, TRUE if the left and right integers have any bits in common.
<<	True if the high 16-bits of the left integer are equal to the right integer.
>>	True if the low 16-bits of the left integer are equal to the right integer.

## Feature and Component State Values

The following table shows where it is valid to use the feature and component operator symbols.

<b>Operator</b> <b>&lt;state&gt;</b>	<b>Where this syntax is valid</b>
\$component-action	In the <a href="#">Condition</a> table, and in the sequence tables, after the <a href="#">CostFinalize</a> action.
&feature-action	In the Condition table, and in the sequence tables, after the <a href="#">CostFinalize</a> action.

!feature-state	In the Condition table, and in the sequence tables, after the CostFinalize action.
?component-state	In the Condition table, and in the sequence tables, after the CostFinalize action.

The following table shows the feature and component state values used in conditional expressions. These states are not set until **MsiSetInstallLevel** is called, either directly or by the **CostFinalize** action.

State	Value	Meaning
INSTALLSTATE_UNKNOWN	-1	No action to be taken on the feature or component.
INSTALLSTATE_ADVERTISED	1	Advertised feature. This state is not available for components.
INSTALLSTATE_ABSENT	2	Feature or component is not present.
INSTALLSTATE_LOCAL	3	Feature or component on the local computer.
INSTALLSTATE_SOURCE	4	Feature or component run from the source.

For example, the conditional expression "&MyFeature=3" evaluates to True only if MyFeature is changing from its current state to the state of being installed on the local computer, INSTALLSTATE\_LOCAL.

Note that you should not depend upon the condition \$Component1=3 to check whether Component1 is locally installed on the computer. This can fail if Component1 is installed by more than one product. After Component1 has been installed locally by Product1, the installer evaluates the condition \$Component1=3 as False during the installation of Product2. This is because the installer determines the version of the component using the component's key path and marks the component for installation if its version is greater than or equal to the installed component.

Note that the installer will not do direct comparisons of the [Version](#) data type in conditional statements. For example, you cannot use comparative operators to compare versions such as "01.10" and "1.010" in a conditional statement. Instead use a valid method to search for a version, such as described in [Searching for Existing Applications, Files, Registry Entries or .ini File Entries](#), and then set a property.

## See Also

[Using Properties in Conditional Statements](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Examples of Conditional Statement Syntax

The following provides some common instances of conditional statements. For more information, see [Conditional Statement Syntax](#).

## Run action on removal.

For information, see [Conditioning Actions to Run During Removal](#).

## Run action only if the product has not been installed.

```
NOT Installed
```

## Run action only if the product will be installed local. Do not run action on a reinstallation.

```
(&FeatureName=3) AND NOT(!FeatureName=3)
```

The term "&FeatureName=3" means the action is to install the feature local. The term "NOT(!FeatureName=3)" means the feature is not installed local.

## Run action only if the feature will be uninstalled.

```
(&FeatureName=2) AND (!FeatureName=3)
```

This condition only checks for a transition of the feature from an installed state of local to the absent state.

## Run action only if the component was installed local, but is transitioning out of state.

```
(?ComponentName=3) AND ($ComponentName=2 OR $ComponentName=4)
```

The term "?ComponetName=3" means the component is installed local.

The term "\$ComponentName=2" means that the action state on the component is Absent. The term "\$ComponentName=4" means that the action state on the component is run from source. Note that an action state of advertise is not valid for a component.

## **Run action only on the reinstallation of a component.**

```
?ComponentName=$ComponentName
```

## **Run action only when a particular patch is applied.**

```
PATCH AND PATCH >< MEDIASRCPROPNAME
```

For more information, see the Remarks section on the [PATCH](#) property page.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Column Definition Format

**MsiViewGetColumnInfo** and the **ColumnInfo Property** of the **View** object use the following format to describe database column definitions. Each column is described by a string in the corresponding record field returned by the function or property. The definition string consists of a single letter representing the data type followed by the width of the column (in characters when applicable, bytes otherwise). A width of zero designates an unbounded width (for example, long text fields and streams). An uppercase letter indicates that null values are allowed in the column.

Column descriptor	Definition string
s?	String, variable length (?=1-255)
s0	String, variable length
i2	Short integer
i4	Long integer
v0	Binary Stream
g?	Temporary string (?=0-255)
j?	Temporary integer (?=0,1,2,4)
O0	Temporary object

The strings used to describe columns have the following relationship to the SQL query strings used by CREATE and ALTER. For more information, see [SQL Syntax](#).

Returned value	SQL syntax
s0	LONGCHAR
l0	LONGCHAR LOCALIZABLE
s #	CHAR(#)
s #	CHARACTER(#)

l #	CHAR(#) LOCALIZABLE
l #	CHARACTER(#) LOCALIZABLE
i2	SHORT
i2	INT
i2	INTEGER
i4	LONG
v0	OBJECT

If the letter is not capitalized, the SQL statement should be appended with NOT NULL.

Returned value	SQL syntax
s0	LONGCHAR NOT NULL

The installer does not internally limit the length of columns to the value specified by the column definition format. If the data entered into a field exceeds the specified column length, the package fails to pass [package validation](#). To pass validation in this case, either the data or the database schema must be changed. The only means to change the column length of a standard table is by exporting the table using [MsiDatabaseExport](#), editing the exported .idt file, and then importing the table using [MsiDatabaseImport](#). Authors cannot change the column data types, nullability, or localization attributes of any columns in standard tables. Authors can create custom tables with columns having any column attributes.

When using [MsiDatabaseMerge](#) to merge a reference database into a target database, the column names, number of primary keys, and column data types must match. [MsiDatabaseMerge](#) ignores the localization and column length attributes. If a column in the reference database has a length that is 0 or greater than the that column's length in the target

database, **MsiDatabaseMerge** increases the column length in the target database to the length in the reference database.

When using Mergmod.dll version 2.0, the application of a merge module to a .msi file never changes the length of columns or the column types of an existing database table. The application of a merge module can however change the schema of an existing database table if the module adds new columns to a table for which it is valid to add columns. When using a Mergemod.dll version less than version 2.0, the application of a merge module never changes the length of columns and never changes the schema of the target database.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Determining Whether a Column is a Primary or External Key

You can determine whether a specific column is a primary key or external key of the table by consulting the reference page of the table. A list of links to the reference pages of all the installation database tables is located under [Database Tables](#). Each of the reference pages identifies columns that are a primary key or an external key.

Database columns that are external keys take the name of the primary key column with an added underscore character. For example, an external key to the File column of the File table is always named File\_.

To view entity relationship diagrams that illustrate table relationships, see [Core Tables Group](#), [File Tables Group](#), [Registry Tables Group](#), [System Tables Group](#), or [User Interface Schema](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Installer Database Reference

To add installer functionality to your application, you can use the Windows Installer database tables and the installer database functions. The topics in this section describe each of the database functions and database tables.

To view a list of all the installation database tables, see [Database Tables](#).

To view a list of the functions that are used to work with information in installation databases, see [Database Functions](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Database Functions

This material is intended for developers who are writing their own setup programs and developers who want to learn more about the installer database tables. For general information on the installer, see [About Windows Installer](#).

You can use the installer access functions to access the database and the installation process. These functions should only be used by custom installation actions and authoring tools. Some of the installer access functions require SQL query strings for querying the database. Queries must adhere to the installer [SQL syntax](#).

This topic lists the installer database access functions by category.

## General Database Access Functions

Function	Description
<a href="#">MsiDatabaseCommit</a>	Commits changes to a database.
<a href="#">MsiDatabaseGetPrimaryKeys</a>	Returns the names of all the primary key columns.
<a href="#">MsiDatabaseIsTablePersistent</a>	Returns an enumeration describing the state of a table.
<a href="#">MsiDatabaseOpenView</a>	Prepares a database query and creates a view object.
<a href="#">MsiGetActiveDatabase</a>	Returns the active database for the installation.
<a href="#">MsiViewGetColumnInfo</a>	Returns column names or definitions.
<a href="#">MsiOpenDatabase</a>	Opens a database file for data access.
<a href="#">MsiViewClose</a>	Releases the result set for an executed view.
<a href="#">MsiViewExecute</a>	Executes the view query and supplies required parameters.
<a href="#">MsiViewFetch</a>	Fetches the next sequential record from the

	view.
<b>MsiViewGetError</b>	Returns the error that occurred in the <b>MsiViewModify</b> function.
<b>MsiViewModify</b>	Updates a fetched record.

## Database Management Functions

Function	Description
<b>MsiCreateTransformSummaryInfo</b>	Creates summary information for an existing transform.
<b>MsiDatabaseApplyTransform</b>	Applies a transform to a database.
<b>MsiDatabaseExport</b>	Exports a table from an open database to a text archive file.
<b>MsiDatabaseGenerateTransform</b>	Generates a transform file of differences between two databases.
<b>MsiDatabaseImport</b>	Imports an installer text archive table into an open database.
<b>MsiDatabaseMerge</b>	Merges two databases together.
<b>MsiGetDatabaseState</b>	Returns the state of the database.

## Record Processing Functions

Function	Description
<b>MsiCreateRecord</b>	Creates new record object with specified number of fields.
<b>MsiFormatRecord</b>	Formats record field data and properties using a format string.
<b>MsiRecordClearData</b>	Sets all fields in a record to null.

<b>MsiRecordDataSize</b>	Returns the length of a record field.
<b>MsiRecordGetFieldCount</b>	Returns the number of fields in a record.
<b>MsiRecordGetInteger</b>	Returns the integer value from a record field.
<b>MsiRecordGetString</b>	Returns the string value of a record field.
<b>MsiRecordIsNull</b>	Reports whether a record field is null.
<b>MsiRecordReadStream</b>	Reads bytes from a record stream field into a buffer.
<b>MsiRecordSetInteger</b>	Sets a record field to an integer field.
<b>MsiRecordSetStream</b>	Sets a record stream field from a file.
<b>MsiRecordSetString</b>	Copies a string into the designated field.

## Summary Information Property Functions

<b>Function</b>	<b>Description</b>
<b>MsiGetSummaryInformation</b>	Obtains handle to summary information stream of installer database.
<b>MsiSummaryInfoGetProperty</b>	Gets a single property from the summary information.
<b>MsiSummaryInfoGetPropertyCount</b>	Returns number of properties in the summary information stream.
<b>MsiSummaryInfoPersist</b>	Writes changed summary information back to summary information stream.
<b>MsiSummaryInfoSetProperty</b>	Sets a single summary information property.

## Installer State Access Functions

Function	Description
<b>MsiGetLanguage</b>	Returns the numeric language of the current installation.
<b>MsiGetLastErrorRecord</b>	Returns error record last returned for the calling process.
<b>MsiGetMode</b>	Returns one of the Boolean internal installation states.
<b>MsiGetProperty</b>	Gets the value of an installer property.
<b>MsiSetProperty</b>	Sets the value of an installation property.
<b>MsiSetMode</b>	Sets an internal engine Boolean state.

## Installer Action Functions

Function	Description
<b>MsiDoAction</b>	Executes built-in action, custom action, or user-interface wizard action.
<b>MsiEvaluateCondition</b>	Evaluates a conditional expression containing property names and values.
<b>MsiProcessMessage</b>	Sends an error record to the installer for processing.
<b>MsiSequence</b>	Executes an action sequence.

## Installer Location Functions

Function	Description
----------	-------------

<b>MsiGetSourcePath</b>	Returns the full source path for a folder in the Directory table.
<b>MsiGetTargetPath</b>	Returns the full target path for a folder in the Directory table.
<b>MsiSetTargetPath</b>	Sets the full target path for a folder in the Directory table.

## Installer Selection Functions

<b>Function</b>	<b>Description</b>
<b>MsiEnumComponentCosts</b>	Enumerates the disk-space per drive required to install a component.
<b>MsiGetComponentState</b>	Obtains the state of a component.
<b>MsiGetFeatureCost</b>	Returns the disk space required by a feature.
<b>MsiGetFeatureState</b>	Gets the state of a feature.
<b>MsiGetFeatureValidStates</b>	Returns a valid installation state.
<b>MsiSetComponentState</b>	Sets a component to the specified state.
<b>MsiSetFeatureAttributes</b>	Modifies the default attributes of a feature at run time.
<b>MsiSetFeatureState</b>	Sets a feature to a specified state.
<b>MsiSetInstallLevel</b>	Sets the installation level of a full product installation.
<b>MsiVerifyDiskSpace</b>	Checks for sufficient disk space.

## User Interface Functions



Function	Description
<a href="#">MsiEnableUIPreview</a>	Enables preview mode of the user interface.
<a href="#">MsiPreviewBillboard</a>	Displays a billboard with the host control in the displayed dialog box.
<a href="#">MsiPreviewDialog</a>	Displays a dialog box as modeless and inactive.

All functions support both ANSI and Unicode calls. To use these functions, include MsiQuery.h and link with Msi.lib.

## Installation Functions

In addition to the database access functions listed above, you create an installation package for an application by using the installer functions listed in the [Installer Function Reference](#) section.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SQL Syntax

The SQL query strings for Windows Installer are restricted to the following formats.

Action	Query
Select a group of records	SELECT [DISTINCT]{column-list} FROM {table-list} [WHERE {operation-list}] [ORDER BY {column-list}]
Delete records from a table	DELETE FROM {table} [WHERE {operation-list}]
Modify existing records in a table	UPDATE {table-list} SET {column}={constant} [, {column}={constant}][, ...] [WHERE {operation-list}] UPDATE queries only work on nonprimary key columns.
Add records to a table	INSERT INTO {table} ({column-list}) VALUES ({constant-list}) [TEMPORARY] Binary data cannot be inserted into a table directly using the INSERT INTO or UPDATE SQL queries. For more information, see <a href="#">Adding Binary Data to a Table Using SQL</a> .
Add a table	CREATE TABLE {table} ( {column} {column type}) [HOLD] Column types must be specified for each column when adding a table. At least one primary key column must be specified for the creation of a new table. The possible substitutions for {column type} in the above are: CHAR [( {size} )]   CHARACTER [( {size} )]   LONGCHAR   SHORT   INT   INTEGER   LONG   OBJECT [NOT NULL] [TEMPORARY] [LOCALIZABLE] [, column...][, ...] PRIMARY KEY column [, column][, ...].
Remove a table	DROP TABLE {table}
Add a column	ALTER TABLE {table} ADD {column} {column type} The column type must be specified when adding a column. The

	possible substitutions for {column type} in the above are: CHAR [( {size} )]   CHARACTER [( {size} )]   LONGCHAR   SHORT   INT   INTEGER   LONG   OBJECT [NOT NULL] [TEMPORARY] [LOCALIZABLE] [HOLD].
Hold and free temporary tables	ALTER TABLE {table name} HOLD ALTER TABLE {table name} FREE  The user can use the commands HOLD and FREE to control the life span of a temporary table or a temporary column. The hold count on a table is incremented for every SQL HOLD operation on that table and decremented for every SQL FREE operation on the table. When the last hold count is released on a table, all temporary columns become inaccessible. If all columns are temporary, the table becomes inaccessible.

For more information, see [Examples of Database Queries Using SQL and Script](#).

## SQL Grammar

The optional parameters are shown enclosed in brackets [ ]. When several choices are listed, the optional parameters are separated by a vertical bar.

A {constant} is either a string or an integer. A string must be enclosed in single quote marks 'example'. A {constant-list} is a comma-delimited list of one or more constants.

The LOCALIZABLE option sets a column attribute that indicates the column needs to be localized.

A {column} is a columnar reference to a value in a field of a table.

A {marker} is a parameter reference to a value supplied by a record submitted with the query. It is represented in the SQL statement by a question mark ?. For information regarding the use of parameters, see either the [MsiViewExecute](#) function or the [Execute](#) method.

The Windows Installer SQL syntax does not support the escaping of

single-quotes (ASCII value 39) in a string literal. However, you can fetch or create the record, set the field with the **StringData** or **IntegerData** property, and then call the **Modify** method. Alternatively, you can create a record and use the parameter markers (?) described in **Execute** method. You can also do this using the database functions **MsiViewExecute**, **MsiRecordSetInteger**, and **MsiRecordSetString**.

A WHERE {operation-list} clause is optional and is a grouping of operations to be used to filter the selection. The operations must be of the following types:

- {column} = {column}
- {column} = | <> | > | < | >= | <= {constant}
- {column} = | <> | > | < | >= | <= {marker}
- {column} is null
- {column} is not null

For string values, only the = or <> operations are possible. Object value comparisons are limited to IS NULL and IS NOT NULL.

Individual operations can be grouped by AND or OR operators. Ordering can be imposed by use of parentheses ( ).

The ORDER BY clause is optional and causes an initial delay during sorting. Ordering by strings will group identical strings together, but it will not alphabetize the strings.

The DISTINCT clause is optional and does not repeat identical records in the returned result set.

A {table-list} is a comma-delimited list of one or more table names referred to as {table} in the join.

A {column-list} is a comma-delimited list of one or more table columns referred to as {column} selected. Ambiguous columns may be further qualified as {tablename.column}. An asterisk may be used as a column-list in a SELECT query to represent all columns in the referenced tables. When referencing fields by column position, select the columns by name instead of using the asterisk. An asterisk cannot be used as a column-list in an INSERT INTO query.

To escape table names and column names that clash with SQL keywords, enclose the name between two grave accent marks `` (ASCII value 96). If a column name must be escaped and is qualified as {tablename.column}, then the table and the column must be escaped individually as {`tablename`.`column`}. It is recommended that all table names and column names be escaped in this fashion to avoid clashes with reserved words and gain significant performance.

Table names are limited to 31 characters. For more information, see [Table Names](#). Table and column names are case-sensitive. SQL keywords are not case-sensitive.

The maximum number of expressions in a WHERE clause of a SQL query is limited to 32.

Only inner joins are supported and are specified by a comparison of columns from different tables. Circular joins are not supported. A circular join is a SQL query that links three or more tables together into a circuit. For example, the following is a circular join:

```
WHERE Table1.Field1=Table2.Field1 AND Table2.Field2=Table3.F
```

Columns that are part of the primary key(s) for a table must be defined first in priority order, followed by any nonprimary key columns. Persistent columns must be defined before temporary columns. The sort sequence of a text column is undefined; however, identical text values always group together.

Note that when adding or creating a column, you must specify the column type.

Tables may not contain more than one column of type 'object'.

The maximum size that can be explicitly specified for a string column in a SQL query is 255. A string column of infinite length is represented as having size 0. For more information, see [Column Definition Format](#).

To execute any SQL statement, a view must be created. However, a view that does not create a result set, such as CREATE TABLE, or INSERT INTO, cannot be used with [MsiViewModify](#) or the [Modify](#) method to update tables through the view.

Note that you cannot fetch a record containing binary data from one

database and then use that record to insert the data into a completely different database. To move binary data from one database to another, you should export the data to a file and then import it into the new database through a query and the [MsiRecordSetStream](#) function. This ensures that each database has its own copy of the binary data.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Passing null as the Argument of Windows Installer Functions

Windows Installer functions that return data in a user provided–memory location should not be called with null as the value for the pointer. These functions return a string or return data as integer pointers, but return inconsistent values when passing null as the value for the output argument.

Do not pass Null as the value of the output argument for any of the following functions:

**[MsiGetProperty](#)**

**[MsiRecordGetString](#)**

**[MsiFormatRecord](#)**

**[MsiGetSourcePath](#)**

**[MsiGetTargetPath](#)**

**[MsiGetFeatureState](#)**

**[MsiViewGetError](#)**

**[MsiSummaryInfoGetProperty](#)**

**[MsiEvaluateCondition](#)**

**[MsiGetFeatureCost](#)**

**[MsiGetFeatureState](#)**

**[MsiGetComponentState](#)**

**[MsiGetFeatureCost](#)**

**[MsiGetFeatureValidStates](#)**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiCreateRecord Function

The **MsiCreateRecord** function creates a new record object with the specified number of fields. This function returns a handle that should be closed using [MsiCloseHandle](#).

## Syntax

```
C++MSIHANDLE MsiCreateRecord(  
    __in unsigned int cParams  
);
```

## Parameters

*cParams* [in]

Specifies the number of fields the record will have. The maximum number of fields in a record is limited to 65535.

## Return Value

If the function succeeds, the return value is handle to a new record object.

If the function fails, the return value is null.

## Remarks

Field 0 of the record object created by the **MsiCreateRecord** function is used for format strings and operation codes and is not included in the count specified by *cParams*. All fields are initialized to null.

Note that it is recommended to use variables of type PMSIHANDLE because the installer closes PMSIHANDLE objects as they go out of scope, whereas you must close MSIHANDLE objects by calling **MsiCloseHandle**. For more information see [Use PMSIHANDLE instead of HANDLE](#) section in the [Windows Installer Best Practices](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[Database Functions](#)

[Record Processing Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiCreateTransformSummaryInfo Function

The **MsiCreateTransformSummaryInfo** function creates summary information of an existing transform to include validation and error conditions. Execution of this function sets the error record, which is accessible by using [MsiGetLastErrorRecord](#).

## Syntax

```
C++UINT MsiCreateTransformSummaryInfo(  
    __in MSIHANDLE hDatabase,  
    __in MSIHANDLE hDatabaseReference,  
    __in LPCTSTR szTransformFile,  
    __in int iErrorConditions,  
    __in int iValidation  
);
```

## Parameters

*hDatabase* [in]

The handle to the database that contains the new database summary information.

*hDatabaseReference* [in]

The handle to the database that contains the original summary information.

*szTransformFile* [in]

The name of the transform to which the summary information is added.

*iErrorConditions* [in]

The error conditions that should be suppressed when the transform is applied. Use one or more of the following values.

Error condition	Meaning
none	None of the

0x00000000	following conditions.
MSITRANSFORM_ERROR_ADDEXISTINGROW 0x00000001	Adding a row that exists.
MSITRANSFORM_ERROR_DELMISSINGROW 0x00000002	Deleting a row that does not exist.
MSITRANSFORM_ERROR_ADDEXISTINGTABLE 0x00000004	Adding a table that exists.
MSITRANSFORM_ERROR_DELMISSINGTABLE 0x00000008	Deleting a table that does not exist.
MSITRANSFORM_ERROR_UPDATEMISSINGROW 0x00000010	Updating a row that does not exist.
MSITRANSFORM_ERROR_CHANGECODEPAGE 0x00000020	Transform and database code pages do not match, and their code pages are neutral.

*iValidation* [in]

Specifies the properties to be validated to verify that the transform can be applied to the database. This parameter can be one or more of the following values.

<b>Validation flag</b>	<b>Meaning</b>
none 0x00000000	Do not validate properties.

MSITRANSFORM_VALIDATE_LANGUAGE 0x00000001	Default language must match base database.
MSITRANSFORM_VALIDATE_PRODUCT 0x00000002	Product must match base database.

Validate product version flags.

<b>Validation flag</b>	<b>Meaning</b>
MSITRANSFORM_VALIDATE_MAJORVERSION 0x00000008	Check major version only.
MSITRANSFORM_VALIDATE_MINORVERSION 0x00000010	Check major and minor versions only.
MSITRANSFORM_VALIDATE_UPDATEVERSION 0x00000020	Check major, minor, and update versions.

Product version relationship flags. In the following table the installed version is the version of the package that is being transformed, and the base version is the version of the package that is used to create the transform.

<b>Validation flag</b>
MSITRANSFORM_VALIDATE_NEWLESSBASEVERSION 0x00000040
MSITRANSFORM_VALIDATE_NEWLESSEQUALBASEVERSION 0x00000080

---

MSITRANSFORM\_VALIDATE\_NEWEQUALBASEVERSION  
0x00000100

---

MSITRANSFORM\_VALIDATE\_NEWGREATEREQUALBASEVERSIC  
0x00000200

---

MSITRANSFORM\_VALIDATE\_NEWGREATERBASEVERSION  
0x00000400

---

Upgrade code validation flags.

<b>Validation flag</b>	<b>Meaning</b>
MSITRANSFORM_VALIDATE_UPGRADECODE 0x00000800	UpgradeCode must match base database.

## Return Value

ERROR\_INVALID\_HANDLE

An invalid or inactive handle is supplied.

ERROR\_INVALID\_PARAMETER

An invalid parameter is passed to the function.

ERROR\_OPEN\_FAILED

The transform storage file cannot be opened.

ERROR\_SUCCESS

The function succeeds.

## ERROR\_INSTALL\_PACKAGE\_INVALID

A reference to an invalid Windows Installer package.

### Remarks

The **ProductCode** Property and **ProductVersion** Property must be defined in the [Property Table](#) of both the base and reference databases. If MSITRANSFORM\_VALIDATE\_UPGRADECODE is used, the **UpgradeCode** Property must also be defined in both databases. If these conditions are not met, **MsiCreateTransformSummaryInfo** returns ERROR\_INSTALL\_PACKAGE\_INVALID.

- Do not use the semicolon for filenames or paths, because it is used as a list delimiter for transforms, sources, and patches.
- This function cannot be called from custom actions. A call to this function from a custom action causes the function to fail.

### Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiCreateTransformSummaryInfoW</b> (Unicode) and <b>MsiCreateTransformSummaryInfoA</b> (ANSI)

### See Also

[Summary Information Stream Property Set Database Transforms](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiDatabaseApplyTransform Function

The **MsiDatabaseApplyTransform** function applies a transform to a database.

## Syntax

```
C++UINT MsiDatabaseApplyTransform(  
    __in MSIHANDLE hDatabase,  
    __in LPCTSTR szTransformFile,  
    __in int iErrorConditions  
);
```

## Parameters

*hDatabase* [in]

Handle to the database obtained from [MsiOpenDatabase](#) to the transform.

*szTransformFile* [in]

Specifies the name of the transform file to apply.

*iErrorConditions* [in]

Error conditions that should be suppressed. This parameter is a bit field that can contain the following bits.

Error condition	Meaning
MSITRANSFORM_ERROR_ADDEXISTINGROW 0x0001	Adding a row that already exists.
MSITRANSFORM_ERROR_DELMISSINGROW 0x0002	Deleting a row that does not exist.
MSITRANSFORM_ERROR_ADDEXISTINGTABLE 0x0004	Adding a table that already

	exists.
MSITRANSFORM_ERROR_DELMISSINGTABLE 0x0008	Deleting a table that does not exist.
MSITRANSFORM_ERROR_UPDATEMISSINGROW 0x0010	Updating a row that does not exist.
MSITRANSFORM_ERROR_CHANGECODEPAGE 0x0020	Transform and database code pages do not match and neither has a neutral code page.
MSITRANSFORM_ERROR_VIEWTRANSFORM 0x0100	Create the temporary <a href="#">_TransformView table</a> .

## Return Value

The **MsiDatabaseApplyTransform** function returns one of the following values:

**ERROR\_INVALID\_HANDLE**

An invalid or inactive handle was supplied.

**ERROR\_INVALID\_PARAMETER**

An invalid parameter was passed to the function.

**ERROR\_INSTALL\_TRANSFORM\_FAILURE**

The transform could not be applied.

**ERROR\_OPEN\_FAILED**

The transform storage file could not be opened.

ERROR\_SUCCESS

The function succeeded.

## Remarks

The **MsiDatabaseApplyTransform** function delays transforming tables until it is necessary. Any tables to be added or dropped are processed immediately. However, changes to the existing table are delayed until the table is loaded or the database is committed.

An error occurs if **MsiDatabaseApplyTransform** is called when tables have already been loaded and saved to storage.

Because the list delimiter for transforms, sources and patches is a semicolon, this character should not be used for filenames or paths.

This function cannot be called from custom actions. A call to this function from a custom action causes the function to fail.

If the function fails, you can obtain extended error information by using [MsiGetLastErrorRecord](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiDatabaseApplyTransformW</b> (Unicode) and <b>MsiDatabaseApplyTransformA</b> (ANSI)

## See Also

[Database Management Functions](#)

## Database Transforms

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiDatabaseCommit Function

The [MsiDatabaseCommit](#) function commits changes to a database.

## Syntax

```
C++UINT MsiDatabaseCommit(  
    __in MSIHANDLE hDatabase  
);
```

## Parameters

*hDatabase* [in]

Handle to the database obtained from [MsiOpenDatabase](#).

## Return Value

The [MsiDatabaseCommit](#) function returns one of the following values:

ERROR\_FUNCTION\_FAILED

The function failed.

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_INVALID\_HANDLE\_STATE

The handle is in an invalid state.

ERROR\_SUCCESS

The function succeeded.

## Remarks

The [MsiDatabaseCommit](#) function finalizes the persistent form of the database. All persistent data is then written to the writable database. No temporary columns or rows are written. The [MsiDatabaseCommit](#) function has no effect on a database that is opened as read-only. You can call this function multiple times to save the current state of tables loaded into memory. When the database is finally closed, any changes made after the database is committed are rolled back. This function is normally

called prior to shutdown when all database changes have been finalized. If the function fails, you can obtain extended error information by using [MsiGetLastErrorRecord](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[General Database Access Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiDatabaseExport Function

The **MsiDatabaseExport** function exports a Microsoft Installer table from an open database to a [Text Archive File](#).

## Syntax

```
C++UINT MsiDatabaseExport(  
    __in MSIHANDLE hDatabase,  
    __in LPCTSTR szTableName,  
    __in LPCTSTR szFolderPath,  
    __in LPCTSTR szFileName  
);
```

## Parameters

*hDatabase* [in]

The handle to a database from [MsiOpenDatabase](#).

*szTableName* [in]

The name of the table to export.

*szFolderPath* [in]

The name of the folder that contains archive files.

*szFileName* [in]

The name of the exported table archive file.

## Return Value

The **MsiDatabaseExport** function returns one of the following values:

Return code	Description
ERROR_BAD_PATHNAME	An invalid path is passed to the function.
ERROR_FUNCTION_FAILED	The function fails.
ERROR_INVALID_HANDLE	An invalid or inactive handle is

	supplied.
ERROR_INVALID_PARAMETER	An invalid parameter is passed to the function.
ERROR_SUCCESS	The function succeeds.

## Remarks

If a table contains streams, **MsiDatabaseExport** exports each stream to a separate file.

For more information, see [MsiDatabaseImport](#).

This function cannot be called from custom actions. A call to this function from a custom action causes the function to fail.

If the function fails, you can get extended error information by using [MsiGetLastErrorRecord](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiDatabaseExportW</b> (Unicode) and <b>MsiDatabaseExportA</b> (ANSI)

## See Also

[Database Management Functions](#)

[Text Archive Files](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiDatabaseGenerateTransform Function

The **MsiDatabaseGenerateTransform** function generates a transform file of differences between two databases. A transform is a way of recording changes to a database without altering the original database. You can also use **MsiDatabaseGenerateTransform** to test whether two databases are identical without creating a transform.

## Syntax

```
C++UINT MsiDatabaseGenerateTransform(  
    __in MSIHANDLE hDatabase,  
    __in MSIHANDLE hDatabaseReference,  
    __in LPCTSTR szTransformFile,  
    __in int iReserved1,  
    __in int iReserved2  
);
```

## Parameters

*hDatabase* [in]

Handle to the database obtained from [MsiOpenDatabase](#) that includes the changes.

*hDatabaseReference* [in]

Handle to the database obtained from **MsiOpenDatabase** that does not include the changes.

*szTransformFile* [in]

A null-terminated string that specifies the name of the transform file being generated. This parameter can be null. If *szTransformFile* is null, you can use **MsiDatabaseGenerateTransform** to test whether two databases are identical without creating a transform. If the databases are identical, the function returns `ERROR_NO_DATA`. If the databases are different the function returns `NOERROR`.

*iReserved1* [in]

This is a reserved argument and must be set to 0.

*iReserved2* [in]

This is a reserved argument and must be set to 0.

## Return Value

The **MsiDatabaseGenerateTransform** function returns one of the following values:

ERROR\_NO\_DATA

If *szTransform* is null, this value is returned if the two databases are identical. No transform file is generated.

ERROR\_CREATE\_FAILED

The storage for the transform file could not be created.

ERROR\_INSTALL\_TRANSFORM\_FAILURE

The transform could not be generated.

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_INVALID\_PARAMETER

An invalid parameter was passed to the function.

ERROR\_SUCCESS

The function succeeded.

NOERROR

If *szTransformFile* is null, this is returned if the two databases are different.

## Remarks

To generate a difference file between two databases, use the **MsiDatabaseGenerateTransform** function. A transform contains information regarding insertion and deletion of columns and rows. The validation flags are stored in the summary information stream of the transform file.

For tables that exist in both databases, the only difference between the two schemas that is allowed is the addition of columns to the end of the reference table. You cannot add primary key columns to a table or change the order or names or column definitions of the existing columns

as defined in the base table. In other words, if neither table contains data and columns are removed from the reference table, the resulting table is identical to the base table.

Because the list delimiter for transforms, sources and patches is a semicolon, this character should not be used for filenames or paths.

This function does not generate a Summary Information stream. Use [MsiCreateTransformSummaryInfo](#) to create the stream for an existing transform.

If *szTransformFile* is null, you can test whether two databases are identical without creating a transform. If the databases are identical, `ERROR_NO_DATA` is returned, `NOERROR` is returned if differences are found.

This function cannot be called from custom actions. A call to this function from a custom action causes the function to fail.

If the function fails, you can obtain extended error information by using [MsiGetLastErrorRecord](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiDatabaseGenerateTransformW</b> (Unicode) and <b>MsiDatabaseGenerateTransformA</b> (ANSI)

## See Also

[Database Management Functions](#)  
[Database Transforms](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiDatabaseGetPrimaryKeys Function

The **MsiDatabaseGetPrimaryKeys** function returns a record containing the names of all the primary key columns for a specified table. This function returns a handle that should be closed using [MsiCloseHandle](#).

## Syntax

```
C++UINT MsiDatabaseGetPrimaryKeys(  
    __in    MSIHANDLE hDatabase,  
    __in    LPCTSTR szTableName,  
    __out   MSIHANDLE *phRecord  
);
```

## Parameters

*hDatabase* [in]

Handle to the database. See [Obtaining a Database Handle](#).

*szTableName* [in]

Specifies the name of the table from which to obtain primary key names.

*phRecord* [out]

Pointer to the handle of the record that holds the primary key names.

## Return Value

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_INVALID\_PARAMETER

An invalid parameter was passed to the function.

ERROR\_INVALID\_TABLE

An invalid table was passed to the function.

ERROR\_SUCCESS

The function succeeded.

## Remarks

The field count of the returned record is the count of primary key columns returned by the **MsiDatabaseGetPrimaryKeys** function. The returned record contains the table name in Field (0) and the column names that make up the primary key names in succeeding fields. These primary key names correspond to the column numbers for the fields.

This function cannot be used with the [\\_Tables table](#) or the [\\_Columns table](#).

Note that it is recommended to use variables of type PMSIHANDLE because the installer closes PMSIHANDLE objects as they go out of scope, whereas you must close MSIHANDLE objects by calling **MsiCloseHandle**. For more information see [Use PMSIHANDLE instead of HANDLE](#) section in the [Windows Installer Best Practices](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiDatabaseGetPrimaryKeysW</b> (Unicode) and <b>MsiDatabaseGetPrimaryKeysA</b> (ANSI)

## See Also

[General Database Access Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiDatabaseImport Function

The **MsiDatabaseImport** function imports an installer [text archive file](#) into an open database table.

## Syntax

```
C++UINT MsiDatabaseImport(  
    __in MSIHANDLE hDatabase,  
    __in LPCTSTR szFolderPath,  
    __in LPCTSTR szFileName  
);
```

## Parameters

*hDatabase* [in]

Handle to the database obtained from [MsiOpenDatabase](#).

*szFolderPath* [in]

Specifies the path to the folder that contains archive files.

*szFileName* [in]

Specifies the name of the file to import.

## Return Value

The **MsiDatabaseImport** function returns one of the following values:

ERROR\_BAD\_PATHNAME

An invalid path was passed to the function.

ERROR\_FUNCTION\_FAILED

The function failed.

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_INVALID\_PARAMETER

An invalid parameter was passed to the function.

ERROR\_SUCCESS

The function succeeded.

## Remarks

When you use the **MsiDatabaseImport** function to import a text archive table named `_SummaryInformation` into an installer database, you write the "05SummaryInformation" stream. This stream contains standard properties that can be viewed using Windows Explorer and are defined by COM. The rows of the table are written to the property stream as pairs of property ID numbers and corresponding data values. See [Summary Information Stream Property Set](#). Date and time in `_SummaryInformation` are in the format: YYYY/MM/DD hh::mm::ss. For example, 1999/03/22 15:25:45. If the table contains binary streams, the name of the stream is in the data field, and the actual stream is retrieved from a file of that name in a subfolder with the same name as the table.

Text archive files that are exported from a database by **MsiDatabaseExport** are intended for use with version control systems, and are not intended to be used as a means of editing data. Use the database API functions and tools designed for that purpose. Note that control characters in text archive files are translated to avoid conflicts with file delimiters. If a text archive file contains non-ASCII data, it is stamped with the code page of the data, and can only be imported into a database of that exact code page, or into a neutral database. Neutral databases are set to the code page of the imported file. A database can be unconditionally set to a particular code page by importing a pseudo table named: `_ForceCodepage`. The format of such a file is: Two blank lines, followed by a line that contains the numeric code page, a tab delimiter and the exact string: `_ForceCodepage`

This function cannot be called from custom actions. A call to this function from a custom action causes the function to fail.

If the function fails, you can obtain extended error information by using [MsiGetLastErrorRecord](#).

## Requirements

	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or
--	--

<b>Version</b>	Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiDatabaseImportW</b> (Unicode) and <b>MsiDatabaseImportA</b> (ANSI)

## See Also

[Database Management Functions](#)

[Text Archive Files](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiDatabasesTablePersistent Function

The **MsiDatabasesTablePersistent** function returns an enumeration that describes the state of a specific table.

## Syntax

```
C++MSICONDITON MsiDatabaseIsTablePersistent(  
    __in MSIHANDLE hDatabase,  
    __in LPCTSTR szTableName  
);
```

## Parameters

*hDatabase* [in]

Handle to the database that belongs to the relevant table. For more information, see [Obtaining a Database Handle](#).

*szTableName* [in]

Specifies the name of the relevant table.

## Return Value

MSICONDITON\_ERROR

An invalid handle or invalid parameter is passed to the function.

MSICONDITON\_FALSE

The table is temporary.

MSICONDITON\_NONE

The table is unknown.

MSICONDITON\_TRUE

The table is persistent.

## Requirements



<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiDatabaseIsTablePersistentW</b> (Unicode) and <b>MsiDatabaseIsTablePersistentA</b> (ANSI)

## See Also

[General Database Access Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiDatabaseMerge Function

The **MsiDatabaseMerge** function merges two databases together, which allows duplicate rows.

## Syntax

```
C++UINT MsiDatabaseMerge(  
    __in MSIHANDLE hDatabase,  
    __in MSIHANDLE hDatabaseMerge,  
    __in LPCTSTR szTableName  
);
```

## Parameters

*hDatabase* [in]

The handle to the database obtained from **MsiOpenDatabase**.

*hDatabaseMerge* [in]

The handle to the database obtained from **MsiOpenDatabase** to merge into the base database.

*szTableName* [in]

The name of the table to receive merge conflict information.

## Return Value

The **MsiDatabaseMerge** function returns one of the following values:

Return code	Description
ERROR_FUNCTION_FAILED	Row merge conflicts were reported.
ERROR_INVALID_HANDLE	An invalid or inactive handle was supplied.
ERROR_INVALID_TABLE	An invalid table was supplied.
ERROR_SUCCESS	The function succeeded.

ERROR_DATATYPE_MISMATCH	Schema difference between the two databases.
-------------------------	--

## Remarks

The **MsiDatabaseMerge** function and the **Merge** method of the **Database** object cannot be used to merge a module that is included in the installation package. They should not be used to merge **Merge Modules** into a Windows Installer package. To include a merge module in an installation package, authors of installation packages should follow the guidelines that are described in the **Applying Merge Modules** topic.

**MsiDatabaseMerge** does not copy over embedded **Cabinet Files** or **embedded transforms** from the reference database into the target database. Embedded data streams that are listed in the **Binary Table** or **Icon Table** are copied from the reference database to the target database. Storage embedded in the reference database are not copied to the target database.

The **MsiDatabaseMerge** function merges the data of two databases. These databases must have the same code page. **MsiDatabaseMerge** fails if any tables or rows in the databases conflict. A conflict exists if the data in any row in the first database differs from the data in the corresponding row of the second database. Corresponding rows are in the same table of both databases and have the same primary key in both databases. The tables of non-conflicting databases must have the same number of primary keys, same number of columns, same column types, same column names, and the same data in rows with identical primary keys. Temporary columns however don't matter in the column count and corresponding tables can have a different number of temporary columns without creating conflict as long as the persistent columns match.

If the number, type, or name of columns in corresponding tables are different, the schema of the two databases are incompatible and the installer stops processing tables and the merge fails. The installer checks that the two databases have the same schema before checking for row merge conflicts. If ERROR\_DATATYPE\_MISMATCH is returned, you are guaranteed that the databases have not been changed.

If the data in particular rows differ, this is a row merge conflict, the installer returns `ERROR_FUNCTION_FAILED` and creates a new table named `szTableName`. The first column of this table is the name of the table having the conflict. The second column gives the number of rows in the table having the conflict. The table that reports conflicts appears as follows.

Column	Type	Key	Nullable
Table	Text	Y	N
NumRowMergeConflicts	Integer		N

This function cannot be called from custom actions. A call to this function from a custom action causes the function to fail.

If the function fails, you can obtain extended error information by using [MsiGetLastErrorRecord](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiDatabaseMergeW</b> (Unicode) and <b>MsiDatabaseMergeA</b> (ANSI)

## See Also

[Column Definition Format](#)  
[Database Management Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiDatabaseOpenView Function

The **MsiDatabaseOpenView** function prepares a database query and creates a view object. This function returns a handle that should be closed using [MsiCloseHandle](#).

## Syntax

```
C++UINT MsiDatabaseOpenView(  
    __in    MSIHANDLE hDatabase,  
    __in    LPCTSTR  szQuery,  
    __out   MSIHANDLE *pView  
);
```

## Parameters

*hDatabase* [in]

Handle to the database to which you want to open a view object. You can get the handle as described in [Obtaining a Database Handle](#).

*szQuery* [in]

Specifies a SQL query string for querying the database. For correct syntax, see [SQL Syntax](#).

*pView* [out]

Pointer to a handle for the returned view.

## Return Value

The **MsiDatabaseOpenView** function returns one of the following values:

ERROR\_BAD\_QUERY\_SYNTAX

An invalid SQL query string was passed to the function.

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_SUCCESS

The function succeeded, and the handle is to a view object.

## Remarks

The **MsiDatabaseOpenView** function opens a view object for a database. You must open a view object for a database before performing any execution or fetching.

If an error occurs, you can call [MsiGetLastErrorRecord](#) for more information.

Note that it is recommended to use variables of type PMSIHANDLE because the installer closes PMSIHANDLE objects as they go out of scope, whereas you must close MSIHANDLE objects by calling **MsiCloseHandle**. For more information see [Use PMSIHANDLE instead of HANDLE](#) section in the [Windows Installer Best Practices](#).

If the function fails, you can obtain extended error information by using **MsiGetLastErrorRecord**.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiDatabaseOpenViewW</b> (Unicode) and <b>MsiDatabaseOpenViewA</b> (ANSI)

## See Also

[General Database Access Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiDoAction Function

The [MsiDoAction](#) function executes a built-in action, custom action, or user-interface wizard action.

## Syntax

```
C++UINT MsiDoAction(  
    __in MSIHANDLE hInstall,  
    __in LPCTSTR szAction  
);
```

## Parameters

*hInstall* [in]

Handle to the installation provided to a DLL custom action or obtained through [MsiOpenPackage](#), [MsiOpenPackageEx](#), or [MsiOpenProduct](#).

*szAction* [in]

Specifies the action to execute.

## Return Value

ERROR\_FUNCTION\_FAILED

The function failed.

ERROR\_FUNCTION\_NOT\_CALLED

The action was not found.

ERROR\_INSTALL\_FAILURE

The action failed.

ERROR\_INSTALL\_SUSPEND

The user suspended the installation.

ERROR\_INSTALL\_USEREXIT

The user canceled the action.

ERROR\_INVALID\_DATA

A failure occurred while calling the custom action.

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_INVALID\_HANDLE\_STATE

The handle state was invalid.

ERROR\_INVALID\_PARAMETER

An invalid parameter was passed to the function.

ERROR\_MORE\_DATA

The action indicates that the remaining actions should be skipped.

ERROR\_SUCCESS

The function succeeded.

## Remarks

The [MsiDoAction](#) function executes the action that corresponds to the name supplied. If the name is not recognized by the installer as a built-in action or as a custom action in the [CustomAction table](#), the name is passed to the user-interface handler object, which can invoke a function or a dialog box. If a null action name is supplied, the installer uses the upper-case value of the [ACTION](#) property as the action to perform. If no property value is defined, the default action is performed, defined as "INSTALL".

Actions that update the system, such as the [InstallFiles](#) and [WriteRegistryValues](#) actions, cannot be run by calling [MsiDoAction](#). The exception to this rule is if [MsiDoAction](#) is called from a custom action that is scheduled in the [InstallExecuteSequence table](#) between the [InstallInitialize](#) and [InstallFinalize](#) actions. Actions that do not update the system, such as [AppSearch](#) or [CostInitialize](#), can be called.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
----------------	--

<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiDoActionW</b> (Unicode) and <b>MsiDoActionA</b> (ANSI)

## See Also

[Installer Action Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiEnableUIPreview Function

The **MsiEnableUIPreview** function enables preview mode of the user interface to facilitate authoring of user-interface dialog boxes. This function returns a handle that should be closed using [MsiCloseHandle](#).

## Syntax

```
C++UINT MsiEnableUIPreview(  
    __in    MSIHANDLE hDatabase,  
    __out   MSIHANDLE *phPreview  
);
```

## Parameters

*hDatabase* [in]  
Handle to the database.

*phPreview* [out]  
Pointer to a returned handle for user-interface preview capability.

## Return Value

ERROR\_INVALID\_HANDLE  
An invalid or inactive handle was supplied.

ERROR\_INVALID\_PARAMETER  
Bad parameter.

ERROR\_OUTOFMEMORY  
Out of memory.

ERROR\_SUCCESS  
The function succeeded.

## Remarks

Note that it is recommended to use variables of type PMSIHANDLE because the installer closes PMSIHANDLE objects as they go out of scope, whereas you must close MSIHANDLE objects by calling

**MsiCloseHandle.** For more information see [Use PMSIHANDLE instead of HANDLE](#) section in the [Windows Installer Best Practices](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[User Interface Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiEnumComponentCosts Function

The [MsiEnumComponentCosts](#) function enumerates the disk-space per drive required to install a component. This information is needed to display the disk-space cost required for all drives in the user interface. The returned disk-space costs are expressed in multiples of 512 bytes.

[MsiEnumComponentCosts](#) should only be run after the installer has completed file costing and after the [CostFinalize](#) action. For more information, see [File Costing](#).

## Syntax

```
C++UINT MsiEnumComponentCosts(  
    __in     MSIHANDLE hInstall,  
    __in     LPCTSTR  szComponent,  
    __in     DWORD    dwIndex,  
    __in     INSTALLSTATE iState,  
    __out    LPTSTR  lpDriveBuf,  
    __inout  DWORD    *pcchDriveBuf,  
    __out    int     *piCost,  
    __out    int     *pTempCost  
);
```

## Parameters

*hInstall* [in]

Handle to the installation provided to a DLL custom action or obtained through [MsiOpenPackage](#), [MsiOpenPackageEx](#), or [MsiOpenProduct](#).

*szComponent* [in]

A null-terminated string specifying the component's name as it is listed in the Component column of the [Component table](#). This parameter can be null. If *szComponent* is null or an empty string, [MsiEnumComponentCosts](#) enumerates the total disk-space per drive used during the installation. In this case, *iState* is ignored. The costs of the installer include those costs for caching the database in the secure folder as well as the cost to create the installation script.

Note that the total disk-space used during the installation may be larger than the space used after the component is installed.

*dwIndex* [in]

0-based index for drives. This parameter should be zero for the first call to the **MsiEnumComponentCosts** function and then incremented for subsequent calls.

*iState* [in]

Requested component state to be enumerated. If *szComponent* is passed as Null or an empty string, the installer ignores the *iState* parameter.

*lpDriveBuf* [out]

Buffer that holds the drive name including the null terminator. This is an empty string in case of an error.

*pcchDriveBuf* [in, out]

Pointer to a variable that specifies the size, in TCHARs, of the buffer pointed to by the *lpDriveBuf* parameter. This size should include the terminating null character. If the buffer provided is too small, the variable pointed to by *pcchDriveBuf* contains the count of characters not including the null terminator.

*piCost* [out]

Cost of the component per drive expressed in multiples of 512 bytes. This value is 0 if an error has occurred. The value returned in *piCost* is final disk-space used by the component after installation. If *szComponent* is passed as Null or an empty string, the installer sets the value at *piCost* to 0.

*pTempCost* [out]

The component cost per drive for the duration of the installation, or 0 if an error occurred. The value in *\*piTempCost* represents the temporary space requirements for the duration of the installation. This temporary space requirement is space needed only for the duration of the installation. This does not affect the final disk space requirement.

## Return Value

Return Value	Meaning
--------------	---------

ERROR_INVALID_HANDLE_STATE	The configuration data is corrupt.
ERROR_INVALID_PARAMETER	An invalid parameter was passed to the function.
ERROR_NO_MORE_ITEMS	There are no more drives to return.
ERROR_SUCCESS	A value was enumerated.
ERROR_UNKNOWN_COMPONENT	The component is missing.
ERROR_FUNCTION_NOT_CALLED	Costing is not complete.
ERROR_MORE_DATA	Buffer not large enough for the drive name.
ERROR_INVALID_HANDLE	The supplied handle is invalid or inactive.

## Remarks

The recommended method for enumerating the disk-space costs per drive is as follows. Start with the `dwIndex` set to 0 and increment it by one after each call. Continue the enumeration as long as [MsiEnumComponentCosts](#) returns `ERROR_SUCCESS`.

[MsiEnumComponentCosts](#) may be called from custom actions.

The total final disk cost for the installation is the sum of the costs of all components plus the cost of the Windows Installer (`szComponent = null`).

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000

<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiEnumComponentCostsW</b> (Unicode) and <b>MsiEnumComponentCostsA</b> (ANSI)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiEvaluateCondition Function

The [MsiEvaluateCondition](#) function evaluates a conditional expression containing property names and values.

## Syntax

```
C++MSICONDITION MsiEvaluateCondition(  
    __in MSIHANDLE hInstall,  
    __in LPCTSTR szCondition  
);
```

## Parameters

*hInstall* [in]

Handle to the installation provided to a DLL custom action or obtained through [MsiOpenPackage](#), [MsiOpenPackageEx](#), or [MsiOpenProduct](#).

*szCondition* [in]

Specifies the conditional expression. This parameter must not be null. For the syntax of conditional expressions see [Conditional Statement Syntax](#).

## Return Value

MSICONDITION\_ERROR

An invalid handle was passed to the function, or the conditional expression has a syntax error.

MSICONDITION\_FALSE

An expression that evaluates to FALSE was passed to the function.

MSICONDITION\_NONE

No expression was passed to the function.

MSICONDITION\_TRUE

An expression that evaluates to TRUE was passed to the function.

## Remarks

The following table shows the feature and component state values used by the **MsiEvaluateCondition** function. These states are not set until **MsiSetInstallLevel** is called, either directly or by the **CostFinalize** action. Therefore, state checking is generally only useful for conditional expressions in an action sequence table.

Value	Meaning
INSTALLSTATE_ABSENT	Feature or component not present.
INSTALLSTATE_LOCAL	Feature or component on local computer.
INSTALLSTATE_SOURCE	Feature or component run from source.
(null value)	No action to be taken on feature or component.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiEvaluateConditionW</b> (Unicode) and <b>MsiEvaluateConditionA</b> (ANSI)

## See Also

[Installer Action Functions](#)

[Passing Null as the Argument of Windows Installer Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiFormatRecord Function

The [MsiFormatRecord](#) function formats record field data and properties using a format string.

## Syntax

```
C++UINT MsiFormatRecord(  
    __in     MSIHANDLE hInstall,  
    __in     MSIHANDLE hRecord,  
    __out    LPTSTR  szResultBuf,  
    __inout  DWORD   pcchResultBuf  
);
```

## Parameters

### *hInstall* [in]

Handle to the installation. This may be omitted, in which case only the record field parameters are processed and properties are not available for substitution.

### *hRecord* [in]

Handle to the record to format. The template string must be stored in record field 0 followed by referenced data parameters.

### *szResultBuf* [out]

Pointer to the buffer that receives the null terminated formatted string. Do not attempt to determine the size of the buffer by passing in a null (value=0) for *szResultBuf*. You can get the size of the buffer by passing in an empty string (for example ""). The function then returns `ERROR_MORE_DATA` and *pcchResultBuf* contains the required buffer size in TCHARs, not including the terminating null character. On return of `ERROR_SUCCESS`, *pcchResultBuf* contains the number of TCHARs written to the buffer, not including the terminating null character.

### *pcchResultBuf* [in, out]

Pointer to the variable that specifies the size, in TCHARs, of the buffer pointed to by the variable *szResultBuf*. When the function

returns `ERROR_SUCCESS`, this variable contains the size of the data copied to `szResultBuf`, not including the terminating null character. If `szResultBuf` is not large enough, the function returns `ERROR_MORE_DATA` and stores the required size, not including the terminating null character, in the variable pointed to by `pcchResultBuf`.

## Return Value

The [MsiFormatRecord](#) function returns one of the following values:

`ERROR_INVALID_HANDLE`

An invalid or inactive handle was supplied.

`ERROR_INVALID_PARAMETER`

An invalid parameter was passed to the function.

`ERROR_MORE_DATA`

The buffer passed in was too small to hold the entire value.

`ERROR_SUCCESS`

The function succeeded.

## Remarks

The [MsiFormatRecord](#) function uses the following format process.

Parameters that are to be [formatted](#) are enclosed in square brackets [...]. The square brackets can be iterated because the substitutions are resolved from the inside out.

If a part of the string is enclosed in curly braces { } and contains no square brackets, it is left unchanged, including the curly braces.

If a part of the string is enclosed in curly braces { } and contains one or more property names, and if all the properties are found, the text (with the resolved substitutions) is displayed without the curly braces. If any of the properties is not found, all the text in the braces and the braces themselves are removed.

Note in the case of deferred execution custom actions, [MsiFormatRecord](#) only supports [CustomActionData](#) and [ProductCode](#) properties. For more information, see [Obtaining Context Information for](#)

## Deferred Execution Custom Actions.

The following steps describe how to format strings using the **MsiFormatRecord** function:

### ▶ To format strings using the MsiFormatRecord function

1. The numeric parameters are substituted by replacing the marker with the value of the corresponding record field, with missing or null values producing no text.
2. The resultant string is processed by replacing the nonrecord parameters with the corresponding values, described next.
  - If a substring of the form [propertyname] is encountered, it is replaced by the value of the property.
  - If a substring of the form [%environmentvariable] is found, the value of the environment variable is substituted.
  - If a substring of the form [#filekey] is found, it is replaced by the full path of the file, with the value *filekey* used as a key into the [File table](#). The value of [#filekey] remains blank and is not replaced by a path until the installer runs the [CostInitialize action](#), [FileCost action](#), and [CostFinalize action](#). The value of [#filekey] depends upon the installation state of the component to which the file belongs. If the component is being run from source, the value is the path to the source location of the file. If the component is being run locally, the value is the path to the target location of the file after installation. If the component is absent, the path is blank. For more information about checking the installation state of components, see [Checking the Installation of Features, Components, Files](#).
  - If a substring of the form [\$componentkey] is found, it is replaced by the install directory of the component, with the value *componentkey* used as a key into the [Component](#)

[table](#). The value of [*\$componentkey*] remains blank and is not replaced by a directory until the installer runs the [CostInitialize action](#), [FileCost action](#), and [CostFinalize action](#). The value of [*\$componentkey*] depends upon the installation state of the component. If the component is being run from source, the value is the source directory of the file. If the component is being run locally, the value is the target directory after installation. If the component is absent, the value is left blank. For more information about checking the installation state of components, see [Checking the Installation of Features, Components, Files](#).

- Note that if a component is already installed, and is not being reinstalled, removed, or moved during the current installation, the action state of the component is null and therefore the string [*\$componentkey*] evaluates to Null.
- If a substring of the form [*c*] is found, it is replaced by the character without any further processing. Only the first character after the backslash is kept; everything else is removed.

If `ERROR_MORE_DATA` is returned, the parameter which is a pointer gives the size of the buffer required to hold the string. If `ERROR_SUCCESS` is returned, it gives the number of characters written to the string buffer. Therefore you can get the size of the buffer by passing in an empty string (for example `""`) for the parameter that specifies the buffer. Do not attempt to determine the size of the buffer by passing in a Null (`value=0`).

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows

<b>Version</b>	Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiFormatRecordW</b> (Unicode) and <b>MsiFormatRecordA</b> (ANSI)

## See Also

[Passing Null as the Argument of Windows Installer Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetActiveDatabase Function

The **MsiGetActiveDatabase** function returns the active database for the installation. This function returns a read-only handle that should be closed using [MsiCloseHandle](#).

## Syntax

```
C++MSIHANDLE MsiGetActiveDatabase(  
    __in MSIHANDLE hInstall  
);
```

## Parameters

*hInstall* [in]

Handle to the installation provided to a DLL custom action or obtained through [MsiOpenPackage](#), [MsiOpenPackageEx](#), or [MsiOpenProduct](#).

## Return Value

If the function succeeds, it returns a read-only handle to the database currently in use by the installer. If the function fails, the function returns zero, 0.

## Remarks

The **MsiGetActiveDatabase** function accesses the database in use by the running the installation.

Note that it is recommended to use variables of type PMSIHANDLE because the installer closes PMSIHANDLE objects as they go out of scope, whereas you must close MSIHANDLE objects by calling **MsiCloseHandle**. For more information see [Use PMSIHANDLE instead of HANDLE](#) section in the [Windows Installer Best Practices](#).

## Requirements

---

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[General Database Access Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetComponentState Function

The **MsiGetComponentState** function obtains the state of a component.

## Syntax

```
C++UINT MsiGetComponentState(  
    __in    MSIHANDLE hInstall,  
    __in    LPCTSTR szComponent,  
    __out   INSTALLSTATE *piInstalled,  
    __out   INSTALLSTATE *piAction  
);
```

## Parameters

*hInstall* [in]

Handle to the installation provided to a DLL custom action or obtained through [MsiOpenPackage](#), [MsiOpenPackageEx](#), or [MsiOpenProduct](#).

*szComponent* [in]

A null-terminated string that specifies the component name within the product.

*piInstalled* [out]

Receives the current installed state. This parameter must not be null. This parameter can be one of the following values.

Value	Meaning
INSTALLSTATE_ABSENT	The component is not installed.
INSTALLSTATE_DEFAULT	The component is installed in the default location: local or source.
INSTALLSTATE_LOCAL	The component is installed on the local drive.
INSTALLSTATE_REMOVED	The component is being removed. In action state and not settable.

INSTALLSTATE_SOURCE	The component runs from the source, CD-ROM, or network.
INSTALLSTATE_UNKNOWN	An unrecognized product or feature name was passed to the function.

*piAction* [out]

Receives the action taken during the installation. This parameter must not be null. For return values, see *piInstalled*.

## Return Value

The **MsiGetComponentState** function returns the following values:

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_SUCCESS

The function succeeded.

ERROR\_UNKNOWN\_COMPONENT

An unknown component was requested.

## Remarks

If the function fails, you can obtain extended error information by using [MsiGetLastErrorRecord](#).

For more information, see [Calling Database Functions From Programs](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h

<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiGetComponentStateW</b> (Unicode) and <b>MsiGetComponentStateA</b> (ANSI)

## See Also

[Installer Selection Functions](#)

[Passing Null as the Argument of Windows Installer Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetDatabaseState Function

The **MsiGetDatabaseState** function returns the state of the database.

## Syntax

```
C++MSIDBSTATE MsiGetDatabaseState(  
    __in MSIHANDLE hDatabase  
);
```

## Parameters

*hDatabase* [in]

Handle to the database obtained from [MsiOpenDatabase](#).

## Return Value

MSIDBSTATE\_READ

The database is opened as read-only; there can be no persistent changes.

MSIDBSTATE\_ERROR

An invalid database handle was passed to the function.

MSIDBSTATE\_WRITE

The database is readable and writable.

## Remarks

The **MsiGetDatabaseState** function returns the update state of the database.

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows

<b>Version</b>	Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[Database Management Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetFeatureCost Function

The **MsiGetFeatureCost** function returns the disk space required by a feature and its selected children and parent features.

## Syntax

```
C++UINT MsiGetFeatureCost(  
    __in    MSIHANDLE hInstall,  
    __in    LPCTSTR szFeature,  
    __in    MSICOSTTREE iCostTree,  
    __in    INSTALLSTATE iState,  
    __out   INT *piCost  
);
```

## Parameters

*hInstall* [in]

Handle to the installation provided to a DLL custom action or obtained through [MsiOpenPackage](#), [MsiOpenPackageEx](#), or [MsiOpenProduct](#).

*szFeature* [in]

Specifies the name of the feature.

*iCostTree* [in]

Specifies the value the function uses to determine disk space requirements. This parameter can be one of the following values.

Value	Meaning
MSICOSTTREE_CHILDREN	The children of the indicated feature are included in the cost.
MSICOSTTREE_PARENTS	The parent features of the indicated feature are included in the cost.
MSICOSTTREE_SELFONLY	The feature only is included in the cost.

*iState* [in]

Specifies the installation state. This parameter can be one of the following values.

<b>Value</b>	<b>Meaning</b>
INSTALLSTATE_UNKNOWN	The product or feature is unrecognized.
INSTALLSTATE_ABSENT	The product or feature is uninstalled.
INSTALLSTATE_LOCAL	The product or feature is installed on the local drive.
INSTALLSTATE_SOURCE	The product or feature is installed to run from source, CD, or network.
INSTALLSTATE_DEFAULT	The product or feature will be installed to use the default location: local or source.

*piCost* [out]

Receives the disk space requirements in units of 512 bytes. This parameter must not be null.

## **Return Value**

The **MsiGetFeatureCost** function returns the following values:

ERROR\_FUNCTION\_FAILED

The function failed.

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_INVALID\_HANDLE\_STATE

An invalid handle state was supplied.

ERROR\_INVALID\_PARAMETER

An invalid parameter was passed to the function.

ERROR\_SUCCESS

The function succeeded.

ERROR\_UNKNOWN\_FEATURE

An unknown feature was requested.

## Remarks

See [Calling Database Functions From Programs](#).

With the **MsiGetFeatureCost** function, the MSICOSTTREE\_SELFONLY value indicates the total amount of disk space (in units of 512 bytes) required by the specified feature only. This returned value does not include the children or the parent features of the specified feature. This total cost is made up of the disk costs attributed to every component linked to the feature.

The MSICOSTTREE\_CHILDREN value indicates the total amount of disk space (in units of 512 bytes) required by the specified feature and its children. For each feature, the total cost is made up of the disk costs attributed to every component linked to the feature.

The MSICOSTTREE\_PARENTS value indicates the total amount of disk space (in units of 512 bytes) required by the specified feature and its parent features (up to the root of the [Feature table](#)). For each feature, the total cost is made up of the disk costs attributed to every component linked to the feature.

**MsiGetFeatureCost** is dependent upon several other functions to be successful. The following example demonstrates the order in which these functions must be called:

```
MSIHANDLE    hInstall;        //product handle, must be closed
int          iCost;          //cost returned by MsiGetFeatureCost

MsiOpenPackage("Path to package....",&hInstall);    //"Path to package"
MsiDoAction(hInstall, "CostInitialize");            //
MsiDoAction(hInstall, "FileCost");
MsiDoAction(hInstall, "CostFinalize");
MsiDoAction(hInstall, "InstallValidate");
MsiGetFeatureCost(hInstall, "FeatureName", MSICOSTTREE_SELFONLY, &iCost);
MsiCloseHandle(hInstall);                          //close the handle
```

The process to query the cost of features scheduled to be removed is slightly different:

```
MSIHANDLE    hInstall;        //product handle, must be closed
int          iCost;          //cost returned by MsiGetFeatureC

MsiOpenPackage("Path to package....",&hInstall);
MsiDoAction(hInstall, "CostInitialize");
MsiDoAction(hInstall, "FileCost");
MsiDoAction(hInstall, "CostFinalize");
MsiSetFeatureState(hInstall, "FeatureName", INSTALLSTATE_ABSEN
MsiDoAction(hInstall, "InstallValidate");
MsiGetFeatureCost(hInstall, "FeatureName", MSICOSTTREE_SELFONL
MsiCloseHandle(hInstall);
```

If the function fails, you can obtain extended error information by using [MsiGetLastErrorRecord](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiGetFeatureCostW</b> (Unicode) and <b>MsiGetFeatureCostA</b> (ANSI)

## See Also

[Installer Selection Functions](#)

[Passing Null as the Argument of Windows Installer Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetFeatureState Function

The **MsiGetFeatureState** function gets the requested state of a feature.

## Syntax

```
C++UINT MsiGetFeatureState(  
    __in    MSIHANDLE hInstall,  
    __in    LPCTSTR szFeature,  
    __out   INSTALLSTATE *piInstalled,  
    __out   INSTALLSTATE *piAction  
);
```

## Parameters

*hInstall* [in]

Handle to the installation provided to a DLL custom action or obtained through [MsiOpenPackage](#), [MsiOpenPackageEx](#), or [MsiOpenProduct](#).

*szFeature* [in]

Specifies the feature name within the product.

*piInstalled* [out]

Specifies the returned current installed state. This parameter must not be null. This parameter can be one of the following values.

Value	Meaning
INSTALLSTATE_BADCONFIG	The configuration data is corrupt.
INSTALLSTATE_INCOMPLETE	The installation is suspended or in progress.
INSTALLSTATE_SOURCEABSENT	The feature must run from the source, and the source is unavailable.
INSTALLSTATE_MOREDATA	The return buffer is full.

INSTALLSTATE_INVALIDARG	An invalid parameter was passed to the function.
INSTALLSTATE_UNKNOWN	An unrecognized product or feature was specified.
INSTALLSTATE_BROKEN	The feature is broken.
INSTALLSTATE_ADVERTISED	The advertised feature.
INSTALLSTATE_ABSENT	The feature was uninstalled.
INSTALLSTATE_LOCAL	The feature was installed on the local drive.
INSTALLSTATE_SOURCE	The feature must run from the source, CD-ROM, or network.
INSTALLSTATE_DEFAULT	The feature is installed in the default location: local or source.

*piAction* [out]

Receives the action taken during the installation session. This parameter must not be null. For return values, see *piInstalled*.

## Return Value

The **MsiGetFeatureState** function returns the following values:

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_SUCCESS

The function succeeded.

ERROR\_UNKNOWN\_FEATURE

An unknown feature was requested.

## Remarks

See [Calling Database Functions From Programs](#).

If the function fails, you can obtain extended error information by using [MsiGetLastErrorRecord](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiGetFeatureStateW</b> (Unicode) and <b>MsiGetFeatureStateA</b> (ANSI)

## See Also

[Installer Selection Functions](#)

[Passing Null as the Argument of Windows Installer Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetFeatureValidStates Function

The **MsiGetFeatureValidStates** function returns a valid installation state.

## Syntax

```
C++UINT MsiGetFeatureValidStates(  
    __in MSIHANDLE hInstall,  
    __in LPCTSTR szFeature,  
    __out DWORD *pInstallState  
);
```

## Parameters

*hInstall* [in]

Handle to the installation provided to a DLL custom action or obtained through [MsiOpenPackage](#), [MsiOpenPackageEx](#), or [MsiOpenProduct](#).

*szFeature* [in]

Specifies the feature name.

*pInstallState* [out]

Receives the location to hold the valid installation states. For each valid installation state, the installer sets *pInstallState* to a combination of the following values. This parameter should not be null.

Decimal Value	Meaning
2 INSTALLSTATE_ADVERTISED	The feature can be advertised.
4 INSTALLSTATE_ABSENT	The feature can be absent.
8 INSTALLSTATE_LOCAL	The feature can be installed on the local drive.
16	The feature can be configured to

INSTALLSTATE_SOURCE	run from source, CD-ROM, or network.
32 INSTALLSTATE_DEFAULT	The feature can be configured to use the default location: local or source.

## Return Value

The **MsiGetFeatureValidStates** function returns the following values:

**ERROR\_FUNCTION\_FAILED**

The function failed.

**ERROR\_INVALID\_HANDLE**

An invalid or inactive handle was supplied.

**ERROR\_INVALID\_HANDLE\_STATE**

An invalid handle state was supplied.

**ERROR\_INVALID\_PARAMETER**

An invalid parameter was passed to the function.

**ERROR\_SUCCESS**

The function succeeded.

**ERROR\_UNKNOWN\_FEATURE**

An unknown feature was requested.

## Remarks

See [Calling Database Functions From Programs](#).

The **MsiGetFeatureValidStates** function determines state validity by querying all components that are linked to the specified feature without taking into account the current installed state of any component.

The possible valid states for a feature are determined as follows:

- If the feature does not contain components, both **INSTALLSTATE\_LOCAL** and **INSTALLSTATE\_SOURCE** are valid

states for the feature.

- If at least one component of the feature has an attribute of `msidbComponentAttributesLocalOnly` or `msidbComponentAttributesOptional`, `INSTALLSTATE_LOCAL` is a valid state for the feature.
- If at least one component of the feature has an attribute of `msidbComponentAttributesSourceOnly` or `msidbComponentAttributesOptional`, `INSTALLSTATE_SOURCE` is a valid state for the feature.
- If a file of a component that belongs to the feature is patched or from a compressed source, then `INSTALLSTATE_SOURCE` is not included as a valid state for the feature.
- `INSTALLSTATE_ADVERTISE` is not a valid state if the feature disallows advertisement (`msidbFeatureAttributesDisallowAdvertise`) or the feature requires platform support for advertisement (`msidbFeatureAttributesNoUnsupportedAdvertise`) and the platform does not support it.
- `INSTALLSTATE_ABSENT` is a valid state for the feature if its attributes do not include `msidbFeatureAttributesUIDisallowAbsent`.
- Valid states for child features marked to follow the parent feature (`msidbFeatureAttributesFollowParent`) are based upon the parent feature's action or installed state.

After calling **MsiGetFeatureValidStates** a conditional statement may then be used to test the valid installation states of a feature. For example, the following call to **MsiGetFeatureValidStates** gets the installation state of Feature1.

```
MsiGetFeatureValidStates(hProduct, "Feature1", &dwValidStates)
```

If Feature1 has attributes of value 0 (favor local), and Feature1 has one component with attributes of value 0 (local only), the value of `dwValidStates` after the call is 14. This indicates that `INSTALLSTATE_LOCAL`, `INSTALLSTATE_ABSENT`, and

INSTALLSTATE\_ADVERTISED are valid states for Feature1. The following conditional statement evaluates to True if local is a valid state for this feature.

```
(( dwValidStates & ( 1 << INSTALLSTATE_LOCAL ) ) == ( 1 << INSTALLSTATE_LOCAL ) )
```

If the function fails, you can obtain extended error information by using [MsiGetLastErrorRecord](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiGetFeatureValidStatesW</b> (Unicode) and <b>MsiGetFeatureValidStatesA</b> (ANSI)

## See Also

[Installer Selection Functions](#)

[Passing Null as the Argument of Windows Installer Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetLanguage Function

The [MsiGetLanguage](#) function returns the numeric language of the installation that is currently running.

## Syntax

```
C++LANGID MsiGetLanguage(  
    __in MSIHANDLE hInstall  
);
```

## Parameters

*hInstall* [in]

Handle to the installation provided to a DLL custom action or obtained through [MsiOpenPackage](#), [MsiOpenPackageEx](#), or [MsiOpenProduct](#).

## Return Value

If the function succeeds, the return value is the numeric LANGID for the install.

If the function fails, the return value can be the following value.

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

## Remarks

The [MsiGetLanguage](#) function returns 0 if an installation is not running.

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on

<b>Version</b>	Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[Installer State Access Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetLastErrorRecord Function

The **MsiGetLastErrorRecord** function returns the error record that was last returned for the calling process. This function returns a handle that should be closed using [MsiCloseHandle](#).

## Syntax

```
C++MSIHANDLE MsiGetLastErrorRecord(void);
```

## Parameters

This function has no parameters.

## Return Value

A handle to the error record. If the last function was successful, **MsiGetLastErrorRecord** returns a null **MSIHANDLE**.

## Remarks

With the **MsiGetLastErrorRecord** function, field 1 of the record contains the installer error code. Other fields contain data specific to the particular error. The error record is released internally after this function is run.

If the record is passed to [MsiProcessMessage](#), it is formatted by looking up the string in the current database. If there is no installation session but a product database is open, the format string may be obtained by a query on the [Error table](#) using the error code, followed by a call to [MsiFormatRecord](#). If the error code is known, the parameters may be individually interpreted.

The following functions set the per-process error record or reset it to null if no error occurred. **MsiGetLastErrorRecord** also clears the error record after returning it.

- [MsiOpenDatabase](#)
- [MsiDatabaseCommit](#)

- **MsiDatabaseOpenView**
- **MsiDatabaseImport**
- **MsiDatabaseExport**
- **MsiDatabaseMerge**
- **MsiDatabaseGenerateTransform**
- **MsiDatabaseApplyTransform**
- **MsiViewExecute**
- **MsiViewModify**
- **MsiRecordSetStream**
- **MsiGetSummaryInformation**
- **MsiGetSourcePath**
- **MsiGetTargetPath**
- **MsiSetTargetPath**
- **MsiGetComponentState**
- **MsiSetComponentState**
- **MsiGetFeatureState**
- **MsiSetFeatureState**
- **MsiGetFeatureCost**
- **MsiGetFeatureValidStates**
- **MsiSetInstallLevel**

Note that it is recommended to use variables of type **PMSIHANDLE** because the installer closes **PMSIHANDLE** objects as they go out of scope, whereas you must close **MSIHANDLE** objects by calling **MsiCloseHandle**. For more information see [Use PMSIHANDLE instead of HANDLE](#) section in the [Windows Installer Best Practices](#).

The following sample uses a call to **MsiDatabaseOpenView** to show how to obtain extended error information from one of the Windows Installer functions that supports **MsiGetLastErrorRecord**. The example, `OpenViewOnDatabase`, attempts to open a view on a database handle. The `hDatabase` handle can be obtained by a call to **MsiOpenDatabase**. If opening the view fails, the function then tries to obtain extended error

information by using **MsiGetLastErrorRecord**.

```
#include <windows.h>
#include <Msiquery.h>
#pragma comment(lib, "msi.lib")
//-----
// Function: OpenViewOnDatabase
//
// Arguments: hDatabase - handle to a MSI package obtained
//                                     via a call to Msi
//
// Returns: UINT status code. ERROR_SUCCESS for success.
//-----
UINT __stdcall OpenViewOnDatabase(MSIHANDLE hDatabase)
{
    if (!hDatabase)
        return ERROR_INVALID_PARAMETER;

    PMSIHANDLE hView = 0;
    UINT uiReturn = MsiDatabaseOpenView(hDatabase,
                                        TEXT("SELECT
                                        &hView);

    if (ERROR_SUCCESS != uiReturn)
    {
        // try to obtain extended error information

        PMSIHANDLE hLastErrorRec = MsiGetLastErrorRecord(hDatabase);

        TCHAR* szExtendedError = NULL;
        DWORD cchExtendedError = 0;
        if (hLastErrorRec)
        {
            // Since we are not currently calling MsiDatabaseOpenView
            // install session, hInstall is NULL. If the error occurred
            // via a DLL custom action, the hInstall parameter to the
            // custom action entry point could be NULL. In this case,
            // properties that might be contained in the session
            // object are not available.

            // To determine the size of the buffer, call MsiGetLastErrorText
            // szResultBuf must be provided as a pointer to a buffer.
            // *pcchResultBuf set to 0.
        }
    }
}
```

```

        UINT uiStatus = MsiFormatRecord(NULL,
                                        hI,
                                        TE,
                                        &C);

    if (ERROR_MORE_DATA == uiStatus)
    {
        // returned size does not include
        cchExtendedError++;

        szExtendedError = new TCHAR[cchExtendedError];
        if (szExtendedError)
        {
            uiStatus = MsiFormatRecord(NULL,
                                        hI,
                                        TE,
                                        &C);

            if (ERROR_SUCCESS == uiStatus)
            {
                // We now have a valid error
                // message

                // PLACE A COMMENT HERE
                // TO LOG THE MESSAGE
                // IN szExtendedError

            }

            delete [] szExtendedError;
            szExtendedError = NULL;
        }
    }
}

return uiReturn;
}

```

## Requirements

---

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[Installer State Access Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetMode Function

The **MsiGetMode** function is used to determine whether the installer is currently running in a specified mode, as listed in the table. The function returns a Boolean value of TRUE or FALSE, indicating whether the specific property passed into the function is currently set (TRUE) or not set (FALSE).

## Syntax

```
C++BOOL MsiGetMode(  
    __in MSIHANDLE hInstall,  
    __in MSIRUNMODE iRunMode  
);
```

## Parameters

*hInstall* [in]

Handle to the installation provided to a DLL custom action or obtained through **MsiOpenPackage**, **MsiOpenPackageEx**, or **MsiOpenProduct**.

*iRunMode* [in]

Specifies the run mode. This parameter must have one of the following values.

Value	Meaning
MSIRUNMODE_ADMIN	The administrative mode is installing, or the product is installing.
MSIRUNMODE_ADVERTISE	The advertisements are installing or the product is installing or updating.
MSIRUNMODE_MAINTENANCE	An existing installation is being modified or there is a new installation.

MSIRUNMODE_ROLLBACKENABLED	Rollback is enabled.
MSIRUNMODE_LOGENABLED	The log file is active. It was enabled prior to the installation session.
MSIRUNMODE_OPERATIONS	Execute operations are in the determination phase.
MSIRUNMODE_REBOOTATEND	A reboot is necessary after a successful installation (settable).
MSIRUNMODE_REBOOTNOW	A reboot is necessary to continue the installation (settable).
MSIRUNMODE_CABINET	Files from cabinets and <a href="#">Media table</a> files are installing.
MSIRUNMODE_SOURCESHORTNAMES	The source LongFileNames is suppressed through the PID_MSISOURCE summary property.
MSIRUNMODE_TARGETSHORTNAMES	The target LongFileNames is suppressed through the <a href="#">SHORTFILENAME</a> property.
MSIRUNMODE_RESERVED11	Reserved for future use.
MSIRUNMODE_WINDOWS9X	The operating system is Windows 98 or Windows 95.

MSIRUNMODE_ZAWENABLED	The operating system supports demand installation.
MSIRUNMODE_RESERVED14	Reserved for future use.
MSIRUNMODE_RESERVED15	Reserved for future use.
MSIRUNMODE_SCHEDULED	A custom action called from install script execution.
MSIRUNMODE_ROLLBACK	A custom action called from rollback execution script.
MSIRUNMODE_COMMIT	A custom action called from commit execution script.

## Return Value

TRUE indicates the specific property passed into the function is currently set.

FALSE indicates the specific property passed into the function is currently not set.

## Remarks

Note that not all the run mode values of *iRunMode* are available when calling **MsiGetMode** from a deferred custom action. For details, see [Obtaining Context Information for Deferred Execution Custom Actions](#).

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2

<b>Version</b>	or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[Installer State Access Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetProperty Function

The [MsiGetProperty](#) function gets the value for an installer property.

## Syntax

```
C++UINT MsiGetProperty(  
    __in     MSIHANDLE hInstall,  
    __in     LPCTSTR szName,  
    __out    LPTSTR szValueBuf,  
    __inout  DWORD *pchValueBuf  
);
```

## Parameters

*hInstall* [in]

Handle to the installation provided to a DLL custom action or obtained through [MsiOpenPackage](#), [MsiOpenPackageEx](#), or [MsiOpenProduct](#).

*szName* [in]

A null-terminated string that specifies the name of the property.

*szValueBuf* [out]

Pointer to the buffer that receives the null terminated string containing the value of the property. Do not attempt to determine the size of the buffer by passing in a null (value=0) for *szValueBuf*. You can get the size of the buffer by passing in an empty string (for example ""). The function will then return `ERROR_MORE_DATA` and *pchValueBuf* will contain the required buffer size in TCHARs, not including the terminating null character. On return of `ERROR_SUCCESS`, *pcchValueBuf* contains the number of TCHARs written to the buffer, not including the terminating null character.

*pchValueBuf* [in, out]

Pointer to the variable that specifies the size, in TCHARs, of the buffer pointed to by the variable *szValueBuf*. When the function returns `ERROR_SUCCESS`, this variable contains the size of the data copied to *szValueBuf*, not including the terminating null

character. If *szValueBuf* is not large enough, the function returns `ERROR_MORE_DATA` and stores the required size, not including the terminating null character, in the variable pointed to by *pchValueBuf*.

## Return Value

`ERROR_INVALID_HANDLE`

An invalid or inactive handle was supplied.

`ERROR_INVALID_PARAMETER`

An invalid parameter was passed to the function.

`ERROR_MORE_DATA`

The provided buffer was too small to hold the entire value.

`ERROR_SUCCESS`

The function succeeded.

## Remarks

If the value for the property retrieved by the [MsiGetProperty](#) function is not defined, it is equivalent to a 0-length value. It is not an error.

If `ERROR_MORE_DATA` is returned, the parameter which is a pointer gives the size of the buffer required to hold the string. If `ERROR_SUCCESS` is returned, it gives the number of characters written to the string buffer. Therefore you can get the size of the buffer by passing in an empty string (for example `""`) for the parameter that specifies the buffer. Do not attempt to determine the size of the buffer by passing in a Null (value=0).

The following example shows how a DLL custom action could access the value of a property by dynamically determining the size of the value buffer.

```
UINT __stdcall MyCustomAction(MSIHANDLE hInstall)
{
    TCHAR* szValueBuf = NULL;
    DWORD cchValueBuf = 0;
    UINT uiStat = MsiGetProperty(hInstall, TEXT("MyProperty"),
        //cchValueBuf now contains the size of the property
```

```

if (ERROR_MORE_DATA == uiStat)
{
    ++cchValueBuf; // add 1 for null termination
    szValueBuf = new TCHAR[cchValueBuf];
    if (szValueBuf)
    {
        uiStat = MsiGetProperty(hInstall, TEXT("MyPropert
    }
}
if (ERROR_SUCCESS != uiStat)
{
    if (szValueBuf != NULL)
        delete[] szValueBuf;
    return ERROR_INSTALL_FAILURE;
}

// custom action uses MyProperty
// ...

delete[] szValueBuf;

return ERROR_SUCCESS;
}

```

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiGetPropertyW</b> (Unicode) and <b>MsiGetPropertyA</b> (ANSI)

## See Also

[Installer State Access Functions](#)

[Passing Null as the Argument of Windows Installer Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetSourcePath Function

The **MsiGetSourcePath** function returns the full source path for a folder in the [Directory table](#).

## Syntax

```
C++UINT MsiGetSourcePath(  
    __in     MSIHANDLE hInstall,  
    __in     LPCTSTR szFolder,  
    __out    LPTSTR szPathBuf,  
    __inout  DWORD *pcchPathBuf  
);
```

## Parameters

*hInstall* [in]

Handle to the installation provided to a DLL custom action or obtained through [MsiOpenPackage](#), [MsiOpenPackageEx](#), or [MsiOpenProduct](#).

*szFolder* [in]

A null-terminated string that specifies a record of the [Directory table](#). If the directory is a root directory, this can be a value from the DefaultDir column. Otherwise it must be a value from the Directory column.

*szPathBuf* [out]

Pointer to the buffer that receives the null terminated full source path. Do not attempt to determine the size of the buffer by passing in a null (value=0) for *szPathBuf*. You can get the size of the buffer by passing in an empty string (for example ""). The function then returns `ERROR_MORE_DATA` and *pcchPathBuf* contains the required buffer size in TCHARs, not including the terminating null character. On return of `ERROR_SUCCESS`, *pcchPathBuf* contains the number of TCHARs written to the buffer, not including the terminating null character.

*pcchPathBuf* [in, out]

Pointer to the variable that specifies the size, in TCHARs, of the buffer pointed to by the variable *szPathBuf*. When the function returns `ERROR_SUCCESS`, this variable contains the size of the data copied to *szPathBuf*, not including the terminating null character. If *szPathBuf* is not large enough, the function returns `ERROR_MORE_DATA` and stores the required size, not including the terminating null character, in the variable pointed to by *pcchPathBuf*.

## Return Value

The **MsiGetSourcePath** function returns the following values:

`ERROR_DIRECTORY`

The directory specified was not found in the [Directory table](#).

`ERROR_INVALID_HANDLE`

An invalid or inactive handle was supplied.

`ERROR_INVALID_PARAMETER`

An invalid parameter was passed to the function.

`ERROR_MORE_DATA`

The buffer passed in was too small to hold the entire value.

`ERROR_SUCCESS`

The function succeeded.

## Remarks

Before calling this function, the installer must first run the [CostInitialize action](#), [FileCost action](#), and [CostFinalize action](#). For more information, see [Calling Database Functions from Programs](#).

If `ERROR_MORE_DATA` is returned, the parameter which is a pointer gives the size of the buffer required to hold the string. If `ERROR_SUCCESS` is returned, it gives the number of characters written to the string buffer. Therefore you can get the size of the buffer by passing in an empty string (for example `""`) for the parameter that specifies the buffer. Do not attempt to determine the size of the buffer by passing in a Null (value=0).

If the function fails, you can obtain extended error information by using [MsiGetLastErrorRecord](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiGetSourcePathW</b> (Unicode) and <b>MsiGetSourcePathA</b> (ANSI)

## See Also

[Installer Location Functions](#)

[Passing Null as the Argument of Windows Installer Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetSummaryInformation Function

The **MsiGetSummaryInformation** function obtains a handle to the `_SummaryInformation` stream for an installer database. This function returns a handle that should be closed using [MsiCloseHandle](#).

## Syntax

```
C++UINT MsiGetSummaryInformation(  
    __in    MSIHANDLE hDatabase,  
    __in    LPCTSTR szDatabasePath,  
    __in    UINT uiUpdateCount,  
    __out   MSIHANDLE *phSummaryInfo  
);
```

## Parameters

*hDatabase* [in]

Handle to the database.

*szDatabasePath* [in]

Specifies the path to the database.

*uiUpdateCount* [in]

Specifies the maximum number of updated values.

*phSummaryInfo* [out]

Pointer to the location from which to receive the summary information handle.

## Return Value

The **MsiGetSummaryInformation** function returns the following values:

`ERROR_INSTALL_PACKAGE_INVALID`

The installation package is invalid.

`ERROR_INVALID_HANDLE`

An invalid or inactive handle was supplied.

ERROR\_INVALID\_PARAMETER

An invalid parameter was passed to the function.

ERROR\_SUCCESS

The function succeeded.

## Remarks

If the database specified by the **MsiGetSummaryInformation** function is not open, you must specify 0 for *hDatabase* and specify the path to the database in *szDatabasePath*. If the database is open, you must set *szDatabasePath* to 0.

If a value of *uiUpdateCount* greater than 0 is used to open an existing summary information stream, **MsiSummaryInfoPersist** must be called before closing the *phSummaryInfo* handle. Failing to do this will lose the existing stream information.

To view the summary information of a patch using **MsiGetSummaryInformation**, set *szDatabasePath* to the path to the patch. Alternately, you can create a handle to the patch using **MsiOpenDatabase** and then pass that handle to **MsiGetSummaryInformation** as the *hDatabase* parameter.

Note that it is recommended to use variables of type PMSIHANDLE because the installer closes PMSIHANDLE objects as they go out of scope, whereas you must close MSIHANDLE objects by calling **MsiCloseHandle**. For more information see [Use PMSIHANDLE instead of HANDLE](#) section in the [Windows Installer Best Practices](#).

If the function fails, you can obtain extended error information by using **MsiGetLastErrorRecord**.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
----------------	--

<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiGetSummaryInformationW</b> (Unicode) and <b>MsiGetSummaryInformationA</b> (ANSI)

## See Also

[Summary Information Property Functions](#)

[Summary Information Stream Property Set](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetTargetPath Function

The **MsiGetTargetPath** function returns the full target path for a folder in the [Directory table](#).

## Syntax

```
C++UINT MsiGetTargetPath(  
    __in     MSIHANDLE hInstall,  
    __in     LPCTSTR szFolder,  
    __out    LPTSTR szPathBuf,  
    __inout  DWORD *pcchPathBuf  
);
```

## Parameters

*hInstall* [in]

Handle to the installation provided to a DLL custom action or obtained through [MsiOpenPackage](#), [MsiOpenPackageEx](#), or [MsiOpenProduct](#).

*szFolder* [in]

A null-terminated string that specifies a record of the [Directory table](#). If the directory is a root directory, this can be a value from the DefaultDir column. Otherwise it must be a value from the Directory column.

*szPathBuf* [out]

Pointer to the buffer that receives the null terminated full target path. Do not attempt to determine the size of the buffer by passing in a null (value=0) for *szPathBuf*. You can get the size of the buffer by passing in an empty string (for example ""). The function then returns ERROR\_MORE\_DATA and *pcchPathBuf* contains the required buffer size in TCHARs, not including the terminating null character. On return of ERROR\_SUCCESS, *pcchPathBuf* contains the number of TCHARs written to the buffer, not including the terminating null character.

*pcchPathBuf* [in, out]

Pointer to the variable that specifies the size, in **TCHARs**, of the buffer pointed to by the variable *szPathBuf*. When the function returns `ERROR_SUCCESS`, this variable contains the size of the data copied to *szPathBuf*, not including the terminating null character. If *szPathBuf* is not large enough, the function returns `ERROR_MORE_DATA` and stores the required size, not including the terminating null character, in the variable pointed to by *pcchPathBuf*.

## Return Value

The **MsiGetTargetPath** function returns the following values:

`ERROR_DIRECTORY`

The directory specified was not found in the [Directory table](#).

`ERROR_INVALID_HANDLE`

An invalid or inactive handle was supplied.

`ERROR_INVALID_PARAMETER`

An invalid parameter was supplied.

`ERROR_MORE_DATA`

The buffer passed in was too small to hold the entire value.

`ERROR_SUCCESS`

The function succeeded.

## Remarks

If `ERROR_MORE_DATA` is returned, the parameter which is a pointer gives the size of the buffer required to hold the string. If `ERROR_SUCCESS` is returned, it gives the number of characters written to the string buffer. Therefore you can get the size of the buffer by passing in an empty string (for example `""`) for the parameter that specifies the buffer. Do not attempt to determine the size of the buffer by passing in a Null (value=0).

Before calling this function, the installer must first run the [CostInitialize action](#), [FileCost action](#), and [CostFinalize action](#). For more information, see [Calling Database Functions from Programs](#).

**MsiGetTargetPath** returns the default path of the target directory authored in the package if the target's current location is unavailable for an installation. For example, if during a [Maintenance Installation](#) a target directory at a network location is unavailable, the installer resets the target directory paths back to their defaults. To get the actual path of the target directory in this case call **MsiProvideComponent** for a component that is known to have been previously installed into the searched for directory and set *dwInstallMode* to `INSTALLMODE_NODETECTION`.

For more information, see [Calling Database Functions From Programs](#).

If the function fails, you can obtain extended error information by using [MsiGetLastErrorRecord](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiGetTargetPathW</b> (Unicode) and <b>MsiGetTargetPathA</b> (ANSI)

## See Also

[Installer Location Functions](#)

[Passing Null as the Argument of Windows Installer Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiOpenDatabase Function

The **MsiOpenDatabase** function opens a database file for data access. This function returns a handle that should be closed using [MsiCloseHandle](#).

## Syntax

```
C++UINT MsiOpenDatabase(  
    __in    LPCTSTR szDatabasePath,  
    __in    LPCTSTR szPersist,  
    __out   MSIHANDLE *phDatabase  
);
```

## Parameters

*szDatabasePath* [in]

Specifies the full path or relative path to the database file.

*szPersist* [in]

Receives the full path to the file or the persistence mode. You can use the *szPersist* parameter to direct the persistent output to a new file or to specify one of the following predefined persistence modes.

Value	Meaning
MSIDBOPEN_CREATEDIRECT	Create a new database, direct mode read/write.
MSIDBOPEN_CREATE	Create a new database, transact mode read/write.
MSIDBOPEN_DIRECT	Open a database direct read/write without transaction.
MSIDBOPEN_READONLY	Open a database read-only, no persistent changes.
MSIDBOPEN_TRANSACT	Open a database read/write in

	transaction mode.
MSIDBOPEN_PATCHFILE	Add this flag to indicate a patch file.

*phDatabase* [out]

Pointer to the location of the returned database handle.

## Return Value

The **MsiOpenDatabase** function returns the following values:

ERROR\_CREATE\_FAILED

The database could not be created.

ERROR\_INVALID\_PARAMETER

One of the parameters was invalid.

ERROR\_OPEN\_FAILED

The database could not be opened as requested.

ERROR\_SUCCESS

The function succeeded.

## Remarks

To make and save changes to a database first open the database in transaction (MSIDBOPEN\_TRANSACT), create (MSIDBOPEN\_CREATE or MSIDBOPEN\_CREATEDIRECT), or direct (MSIDBOPEN\_DIRECT) mode. After making the changes, always call **MsiDatabaseCommit** before closing the database handle. **MsiDatabaseCommit** flushes all buffers.

Always call **MsiDatabaseCommit** on a database that has been opened in direct mode (MSIDBOPEN\_DIRECT or MSIDBOPEN\_CREATEDIRECT) before closing the database's handle. Failure to do this may corrupt the database.

Because **MsiOpenDatabase** initiates database access, it cannot be used with a running installation.

Note that it is recommended to use variables of type PMSIHANDLE

because the installer closes PMSIHANDLE objects as they go out of scope, whereas you must close MSIHANDLE objects by calling **MsiCloseHandle**. For more information see [Use PMSIHANDLE instead of HANDLE](#) section in the [Windows Installer Best Practices](#).

**Note** When a database is opened as the output of another database, the summary information stream of the output database is actually a read-only mirror of the original database, and, thus, cannot be changed. Additionally, it is not persisted with the database. To create or modify the summary information for the output database, it must be closed and reopened.

If the function fails, you can obtain extended error information by using [MsiGetLastErrorRecord](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiOpenDatabaseW</b> (Unicode) and <b>MsiOpenDatabaseA</b> (ANSI)

## See Also

[General Database Access Functions](#)  
[A Database and Patch Example](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiPreviewBillboard Function

The **MsiPreviewBillboard** function displays a billboard with the host control in the displayed dialog box.

## Syntax

```
C++UINT MsiPreviewBillboard(  
    __in MSIHANDLE hPreview,  
    __in LPCTSTR szControlName,  
    __in LPCTSTR szBillboard  
);
```

## Parameters

*hPreview* [in]

Handle to the preview.

*szControlName* [in]

Specifies the name of the host control.

*szBillboard* [in]

Specifies the name of the billboard to display.

## Return Value

ERROR\_FUNCTION\_FAILED

The function failed.

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_INVALID\_PARAMETER

An invalid parameter was supplied.

ERROR\_SUCCESS

The function succeeded.

## Remarks

Supplying a null billboard name in the [MsiPreviewBillboard](#) function removes any billboard displayed.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiPreviewBillboardW</b> (Unicode) and <b>MsiPreviewBillboardA</b> (ANSI)

## See Also

[User Interface Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiPreviewDialog Function

The [MsiPreviewDialog](#) function displays a dialog box as modeless and inactive.

## Syntax

```
C++UINT MsiPreviewDialog(  
    __in MSIHANDLE hPreview,  
    __in LPCTSTR szDialogName  
);
```

## Parameters

*hPreview* [in]

Handle to the preview.

*szDialogName* [in]

Specifies the name of the dialog box to preview.

## Return Value

ERROR\_FUNCTION\_FAILED

The function failed.

ERROR\_FUNCTION\_NOT\_CALLED

The function was not called.

ERROR\_INSTALL\_FAILURE

An installation failure occurred.

ERROR\_INSTALL\_SUSPEND

The installation was suspended.

ERROR\_INSTALL\_USEREXIT

The user exited the installation.

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_INVALID\_HANDLE\_STATE

An invalid handle state was supplied.

ERROR\_INVALID\_PARAMETER

An invalid parameter was supplied.

ERROR\_SUCCESS

The function succeeded.

## Remarks

Supplying a null name in the [MsiPreviewDialog](#) function removes any current dialog box.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiPreviewDialogW</b> (Unicode) and <b>MsiPreviewDialogA</b> (ANSI)

## See Also

[User Interface Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiProcessMessage Function

The [MsiProcessMessage](#) function sends an error record to the installer for processing.

## Syntax

```
C++int MsiProcessMessage(  
    __in MSIHANDLE hInstall,  
    __in INSTALLMESSAGE eMessageType,  
    __in MSIHANDLE hRecord  
);
```

## Parameters

*hInstall* [in]

Handle to the installation provided to a DLL custom action or obtained through [MsiOpenPackage](#), [MsiOpenPackageEx](#), or [MsiOpenProduct](#).

*eMessageType* [in]

The *eMessage* parameter must be a value specifying one of the following message types. To display a message box with push buttons or icons, use OR-operators to add `INSTALLMESSAGE_ERROR`, `INSTALLMESSAGE_WARNING`, or `INSTALLMESSAGE_USER` to the standard message box styles used by the **MessageBox** and **MessageBoxEx** functions. For more information, see the Remarks below.

Value	Meaning
<code>INSTALLMESSAGE_FATALEXIT</code>	Premature termination, possibly fatal out of memory.
<code>INSTALLMESSAGE_ERROR</code>	Formatted error message, [1] is message number in <a href="#">Error table</a> .

INSTALLMESSAGE_WARNING	Formatted warning message, [1] is message number in Error table.
INSTALLMESSAGE_USER	User request message, [1] is message number in Error table.
INSTALLMESSAGE_INFO	Informative message for log, not to be displayed.
INSTALLMESSAGE_FILESIUSE	List of files currently in use that must be closed before being replaced.
INSTALLMESSAGE_RESOLVESOURCE	Request to determine a valid source location.
INSTALLMESSAGE_RMFILESINUSE	List of files currently in use that must be closed before being replaced. Available beginning with Windows Installer version 4.0. For more information about this message see <a href="#">Using Restart Manager with an External UI</a> .
INSTALLMESSAGE_OUTOFDISKSPACE	Insufficient disk space message.
INSTALLMESSAGE_ACTIONSTART	Progress: start of action, [1] action name, [2] description, [3] template for ACTIONDATA

	messages.
INSTALLMESSAGE_ACTIONDATA	Action data. Record fields correspond to the template of ACTIONSTART message.
INSTALLMESSAGE_PROGRESS	Progress bar information. See the description of record fields below.
INSTALLMESSAGE_COMMONDATA	To enable the Cancel button set [1] to 2 and [2] to 1.  To disable the Cancel button set [1] to 2 and [2] to 0

*hRecord* [in]

Handle to a record containing message format and data.

## Return Value

-1

An invalid parameter or handle was supplied.

0

No action was taken.

IDABORT

The process was stopped.

IDCANCEL

The process was canceled.

IDIGNORE

The process was ignored.

IDOK

The function succeeded.

IDNO

No.

IDRETRY

Retry.

IDYES

Yes.

## Remarks

The [MsiProcessMessage](#) function performs any enabled logging operations and defers execution. You can selectively enable logging for various message types.

For `INSTALLMESSAGE_FATALEXIT`, `INSTALLMESSAGE_ERROR`, `INSTALLMESSAGE_WARNING`, and `INSTALLMESSAGE_USER` messages, if field 0 is not set field 1 must be set to the error code corresponding to the error message in the [Error table](#). Then, the message is formatted using the template from the Error table before passing it to the user-interface handler for display.

## Record Fields for Progress Bar Messages

The following describes the record fields when `eMessageType` is set to `INSTALLMESSAGE_PROGRESS`. Field 1 specifies the type of the progress message. The meaning of the other fields depend upon the value in this field. The possible values that can be set into Field 1 are as follows.

<b>Field 1 value</b>	<b>Field 1 description</b>
0	Resets progress bar and sets the expected total number of ticks in the bar.
1	Provides information related to progress messages to be sent by the current action.
2	Increments the progress bar.

3	Enables an action (such as CustomAction) to add ticks to the expected total number of progress of the progress bar.
---	---

The meaning of Field 2 depends upon the value in Field 1 as follows.

<b>Field 1</b>	
<b>value</b>	<b>Field 2 description</b>
0	Expected total number of ticks in the progress bar.
1	Number of ticks the progress bar moves for each ActionData message that is sent by the current action. This field is ignored if Field 3 is 0.
2	Number of ticks the progress bar has moved.
3	Number of ticks to add to total expected progress.

The meaning of Field 3 depends upon the value in Field 1 as follows.

<b>Field 1</b>		
<b>value</b>	<b>Field 3 value</b>	<b>Field 3 description</b>
0	0	Forward progress bar (left to right)
	1	Backward progress bar (right to left)
1	0	The current action will send explicit ProgressReport messages.
	1	Increment the progress bar by the number of ticks specified in Field 2 each time an ActionData message is sent by the current action.
2	Unused	
3	Unused	

The meaning of Field 4 depends upon the value in Field 1 as follows.



Field		
1	Field 4	
value	value	Field 4 description
0	0	Execution is in progress. In this case, the UI could calculate and display the time remaining.
	1	Creating the execution script. In this case, the UI could display a message to please wait while the installer finishes preparing the installation.
1	Unused	
2	Unused	
3	Unused	

For more information and a code sample, see [Adding Custom Actions to the ProgressBar](#).

## Display of Message Boxes

To display a message box with push buttons or icons, use OR-operators to add `INSTALLMESSAGE_ERROR`, `INSTALLMESSAGE_WARNING`, or `INSTALLMESSAGE_USER` with the message box options used by **MessageBox** and **MessageBoxEx**. The available push button options are `MB_OK`, `MB_OKCANCEL`, `MB_ABORTRETRYIGNORE`, `MB_YESNOCANCEL`, `MB_YESNO`, and `MB_RETRYCANCEL`. The available default button options are `MB_DEFBUTTON1`, `MB_DEFBUTTON2`, and `MB_DEFBUTTON3`. The available icon options are `MB_ICONERROR`, `MB_ICONQUESTION`, `MB_ICONWARNING`, and `MB_ICONINFORMATION`. If no icon options is specified, Windows Installer chooses a default icon style based upon the message type.

For example, the following call to **MsiProcessMessage** sends an `INSTALLMESSAGE_ERROR` message with the `MB_ICONWARNING` icon and the `MB_ABORTRETRYCANCEL` buttons.

```
PMSIHANDLE hInstall;
PMSIHANDLE hRec;
MsiProcessMessage(hInstall,
```

```
INSTALLMESSAGE ( INSTALLMES  
hRec );
```

If a custom action calls [MsiProcessMessage](#), the custom action should be capable of handling a cancellation by the user and should return `ERROR_INSTALL_USEREXIT`.

For more information on sending messages with [MsiProcessMessage](#), see the [Sending Messages to Windows Installer Using MsiProcessMessage](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[Installer Action Functions](#)

[Sending Messages to Windows Installer Using MsiProcessMessage](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiRecordClearData Function

The [MsiRecordClearData](#) function sets all fields in a record to null.

## Syntax

```
C++unsigned int MsiRecordClearData(  
    __in MSIHANDLE hRecord  
);
```

## Parameters

*hRecord* [in]  
Handle to the record.

## Return Value

ERROR\_INVALID\_HANDLE  
The handle is inactive or does not represent a record.

ERROR\_SUCCESS  
The function succeeded.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

## Record Processing Functions

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiRecordDataSize Function

The **MsiRecordDataSize** function returns the length of a record field. The count does not include the terminating null character.

## Syntax

```
C++unsigned int MsiRecordDataSize(  
    __in MSIHANDLE hRecord,  
    __in unsigned int iField  
);
```

## Parameters

*hRecord* [in]

Handle to the record.

*iField* [in]

Specifies a field of the record.

## Return Value

The **MsiRecordDataSize** function returns 0 if the field is null, nonexistent, or an internal object pointer. The function also returns 0 if the handle is not a valid record handle.

If the data is in integer format, the function returns sizeof(int).

If the data is in string format, the function returns the character count (not including the null character).

If the data is in stream format, the function returns the byte count.

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows

<b>Version</b>	Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[Record Processing Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiRecordGetFieldCount Function

The [MsiRecordGetFieldCount](#) function returns the number of fields in a record.

## Syntax

```
C++unsigned int MsiRecordGetFieldCount(  
    __in MSIHANDLE hRecord  
);
```

## Parameters

*hRecord* [in]  
Handle to a record.

## Return Value

If the function succeeds, the return value is the number of fields in the record.

If the function is given an invalid or inactive handle, it returns -1 or 0xFFFFFFFF.

## Remarks

The count returned by the [MsiRecordGetFieldCount](#) parameter does not include field 0. Read access to fields beyond this count returns null values. Write access fails.

## Requirements

### Version

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000

<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[Record Processing Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiRecordGetInteger Function

The [MsiRecordGetInteger](#) function returns the integer value from a record field.

## Syntax

```
C++int MsiRecordGetInteger(  
    __in MSIHANDLE hRecord,  
    __in unsigned int iField  
);
```

## Parameters

*hRecord* [in]  
Handle to a record.

*iField* [in]  
Specifies the field of the record from which to obtain the value.

## Return Value

If the function succeeds, the return value is the integer value of the field.

## Remarks

The [MsiRecordGetInteger](#) function returns MSI\_NULL\_INTEGER if the field is null or if the field is a string that cannot be converted to an integer.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h

<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[Record Processing Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiRecordGetString Function

The [MsiRecordGetString](#) function returns the string value of a record field.

## Syntax

```
C++UINT MsiRecordGetString(  
    __in     MSIHANDLE hRecord,  
    __in     unsigned int iField,  
    __out    LPTSTR szValueBuf,  
    __inout  DWORD *pcchValueBuf  
);
```

## Parameters

*hRecord* [in]

Handle to the record.

*iField* [in]

Specifies the field requested.

*szValueBuf* [out]

Pointer to the buffer that receives the null terminated string containing the value of the record field. Do not attempt to determine the size of the buffer by passing in a null (value=0) for *szValueBuf*. You can get the size of the buffer by passing in an empty string (for example ""). The function then returns `ERROR_MORE_DATA` and *pcchValueBuf* contains the required buffer size in TCHARs, not including the terminating null character. On return of `ERROR_SUCCESS`, *pcchValueBuf* contains the number of **TCHARs** written to the buffer, not including the terminating null character.

*pcchValueBuf* [in, out]

Pointer to the variable that specifies the size, in TCHARs, of the buffer pointed to by the variable *szValueBuf*. When the function returns `ERROR_SUCCESS`, this variable contains the size of the data copied to *szValueBuf*, not including the terminating null character. If *szValueBuf* is not large enough, the function returns

ERROR\_MORE\_DATA and stores the required size, not including the terminating null character, in the variable pointed to by *pcchValueBuf*.

## Return Value

The **MsiRecordGetString** function returns one of the following values:

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_INVALID\_PARAMETER

An invalid parameter was supplied.

ERROR\_MORE\_DATA

The provided buffer was too small to hold the entire value.

ERROR\_SUCCESS

The function succeeded.

## Remarks

If ERROR\_MORE\_DATA is returned, the parameter which is a pointer gives the size of the buffer required to hold the string. If ERROR\_SUCCESS is returned, it gives the number of characters written to the string buffer. To get the size of the buffer, pass in the address of a 1 character buffer as *szValueBuf* and specify the size of the buffer with *pcchValueBuf* as 0. This ensures that no string value returned by the function fits into the buffer. Do not attempt to determine the size of the buffer by passing in a Null (value=0).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h

<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiRecordGetStringW</b> (Unicode) and <b>MsiRecordGetStringA</b> (ANSI)

## See Also

[Record Processing Functions](#)

[Passing Null as the Argument of Windows Installer Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiRecordIsNull Function

The **MsiRecordIsNull** function reports a null record field.

## Syntax

```
C++BOOL MsiRecordIsNull(  
    __in MSIHANDLE hRecord,  
    __in unsigned int iField  
);
```

## Parameters

*hRecord* [in]  
Handle to a record.

*iField* [in]  
Specifies the field to check.

## Return Value

FALSE  
The function succeeds, and the field is not null or the record handle is invalid.

TRUE  
The function succeeds, and the field is null or the field does not exist.

## Remarks

The *iField* parameter is based on 1 (one).

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or

<b>Version</b>	Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[Record Processing Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiRecordReadStream Function

The [MsiRecordReadStream](#) function reads bytes from a record stream field into a buffer.

## Syntax

```
C++UINT MsiRecordReadStream(  
    __in     MSIHANDLE hRecord,  
    __in     UINT iField,  
    __out    char *szDataBuf,  
    __inout  DWORD *pcbDataBuf  
);
```

## Parameters

*hRecord* [in]

Handle to the record.

*iField* [in]

Specifies the field of the record.

*szDataBuf* [out]

A buffer to receive the stream field. You should ensure the destination buffer is the same size or larger than the source buffer. See the Remarks section.

*pcbDataBuf* [in, out]

Specifies the in and out buffer count. On input, this is the full size of the buffer. On output, this is the number of bytes that were actually written to the buffer. See the Remarks section.

## Return Value

ERROR\_INVALID\_DATATYPE

The field is not a stream column.

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

## ERROR\_INVALID\_PARAMETER

An invalid parameter was passed to the function.

## ERROR\_SUCCESS

The function succeeded.

## Remarks

To read a stream, set *pcbDataBuf* to the number of bytes that are to be transferred from stream to buffer each time the function is called. On return, the [MsiRecordReadStream](#) resets *pcbDataBuf* to the number of bytes that were actually transferred. If the buffer is smaller than the stream, the stream is repositioned when the buffer becomes full such that the next data in the stream is transferred by the next call to the function. When no more bytes are available, [MsiRecordReadStream](#) returns `ERROR_SUCCESS`.

If you pass 0 for *szDataBuf* then *pcbDataBuf* is reset to the number of bytes in the stream remaining to be read.

The following code sample reads from a stream that is in field 1 of a record specified by *hRecord* and reads the entire stream 8 bytes at a time.

```
char szBuffer[8];
PMSIHANDLE hRecord;
DWORD cbBuf = sizeof(szBuffer);
do
{
    if (MsiRecordReadStream(hRecord, 1, szBuffer,
        &cbBuf) != ERROR_SUCCESS)
        break; /* error */
}
while (cbBuf == 8); //continue reading the stream while you
//cbBuf will be less once you reach the end of the stream a
//buffer with stream data
```

See also [OLE Limitations on Streams](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[Record Processing Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiRecordSetInteger Function

The [MsiRecordSetInteger](#) function sets a record field to an integer field.

## Syntax

```
C++UINT MsiRecordSetInteger(  
    __in MSIHANDLE hRecord,  
    __in unsigned int iField,  
    __in int iValue  
);
```

## Parameters

*hRecord* [in]

Handle to the record.

*iField* [in]

Specifies the field of the record to set.

*iValue* [in]

Specifies the value to which to set the field.

## Return Value

ERROR\_INVALID\_FIELD

An invalid field of the record was supplied.

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_INVALID\_PARAMETER

An invalid parameter was passed to the function.

ERROR\_SUCCESS

The function succeeded.

## Remarks

In the [MsiRecordSetInteger](#) function, attempting to store a value in a

nonexistent field causes an error. Note that the following code returns `ERROR_INVALID_PARAMETER`.

```
MSIHANDLE hRecord;
UINT lReturn;

//create an msirecord with no fields
hRecord = MsiCreateRecord(0);

//attempting to set the first field's value gives you ERROR
lReturn = MsiRecordSetInteger(hRecord, 1, 0);
```

To set a record integer field to `NULL_INTEGER`, set *iValue* to `MSI_NULL_INTEGER`.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[Record Processing Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiRecordSetStream Function

The **MsiRecordSetStream** function sets a record stream field from a file. Stream data cannot be inserted into temporary fields.

## Syntax

```
C++UINT MsiRecordSetStream(  
    __in MSIHANDLE hRecord,  
    __in UNIT iField,  
    __in LPCTSTR szFilePath  
);
```

## Parameters

*hRecord* [in]

Handle to the record.

*iField* [in]

Specifies the field of the record to set.

*szFilePath* [in]

Specifies the path to the file containing the stream.

## Return Value

The **MsiRecordSetStream** function returns the following values:

ERROR\_BAD\_PATHNAME

An invalid path was supplied.

ERROR\_FUNCTION\_FAILED

The function failed.

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_INVALID\_PARAMETER

An invalid parameter was passed to the function.

ERROR\_SUCCESS

The function succeeded.

## Remarks

The contents of the file specified in the **MsiRecordSetStream** function is read into a stream object. The stream persists if the record is inserted into the database and the database is committed.

To reset the stream to its beginning you must pass in a Null pointer for *szFilePath*. Do not pass a pointer to an empty string, "", to reset the stream.

See also [OLE Limitations on Streams](#).

If the function fails, you can obtain extended error information by using [MsiGetLastErrorRecord](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiRecordSetStreamW</b> (Unicode) and <b>MsiRecordSetStreamA</b> (ANSI)

## See Also

[Record Processing Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiRecordSetString Function

The [MsiRecordSetString](#) function copies a string into the designated field.

## Syntax

```
C++UINT MsiRecordSetString(  
    __in MSIHANDLE hRecord,  
    __in unsigned int iField,  
    __in LPCTSTR szValue  
);
```

## Parameters

*hRecord* [in]

Handle to the record.

*iField* [in]

Specifies the field of the record to set.

*szValue* [in]

Specifies the string value of the field.

## Return Value

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_INVALID\_PARAMETER

An invalid parameter was passed to the function.

ERROR\_SUCCESS

The function succeeded.

## Remarks

In the [MsiRecordSetString](#) function, a null string pointer and an empty string both set the field to null. Attempting to store a value in a nonexistent field causes an error.

To set a record string field to null, set szValue to either a null string or an empty string.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiRecordSetStringW</b> (Unicode) and <b>MsiRecordSetStringA</b> (ANSI)

## See Also

[Record Processing Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSequence Function

The **MsiSequence** function executes another action sequence, as described in the specified table.

## Syntax

```
C++UINT MsiSequence(  
    __in MSIHANDLE hInstall,  
    __in LPCTSTR szTable,  
    __in INT iSequenceMode  
);
```

## Parameters

*hInstall* [in]

Handle to the installation provided to a DLL custom action or obtained through **MsiOpenPackage**, **MsiOpenPackageEx**, or **MsiOpenProduct**.

*szTable* [in]

Specifies the name of the table containing the action sequence.

*iSequenceMode* [in]

This parameter is currently unimplemented. It is reserved for future use and must be 0.

## Return Value

ERROR\_FUNCTION\_FAILED

The function failed.

ERROR\_FUNCTION\_NOT\_CALLED

The function was not called.

ERROR\_INSTALL\_FAILURE

An installation failure occurred.

ERROR\_INSTALL\_SUSPEND

The installation was suspended.

ERROR\_INSTALL\_USEREXIT

The user exited the installation.

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_INVALID\_HANDLE\_STATE

An invalid handle state occurred.

ERROR\_INVALID\_PARAMETER

An invalid parameter was passed to the function.

ERROR\_SUCCESS

The function succeeded.

## Remarks

The [MsiSequence](#) function queries the specified table, ordering the actions by the numbers in the Sequence column. For each row retrieved, an action is executed, provided that any supplied condition expression does not evaluate to FALSE.

An action sequence containing any actions that update the system, such as the [InstallFiles](#) and [WriteRegistryValues](#) actions, cannot be run by calling [MsiSequence](#). The exception to this rule is if [MsiSequence](#) is called from a custom action that is scheduled in the [InstallExecuteSequence table](#) between the [InstallInitialize](#) and [InstallFinalize actions](#). Actions that do not update the system, such as [AppSearch](#) or [CostInitialize](#), can be called.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[Installer Action Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiGetComponentState Function

The **MsiGetComponentState** function sets a component to the requested state.

## Syntax

```
C++UINT MsiGetComponentState(  
    __in MSIHANDLE hInstall,  
    __in LPCTSTR szComponent,  
    __in INSTALLSTATE iState  
);
```

## Parameters

*hInstall* [in]

Handle to the installation provided to a DLL custom action or obtained through [MsiOpenPackage](#), [MsiOpenPackageEx](#), or [MsiOpenProduct](#).

*szComponent* [in]

Specifies the name of the component.

*iState* [in]

Specifies the state to set. This parameter can be one of the following values.

Value	Meaning
INSTALLSTATE_ABSENT	The component was uninstalled.
INSTALLSTATE_LOCAL	The component was installed on the local drive.
INSTALLSTATE_SOURCE	The component will run from source, CD, or network.

## Return Value

The **MsiSetComponentState** function returns the following values:

**ERROR\_FUNCTION\_FAILED**

The function failed.

**ERROR\_INSTALL\_USEREXIT**

The user exited the installation.

**ERROR\_INVALID\_HANDLE**

An invalid or inactive handle was supplied.

**ERROR\_SUCCESS**

The function succeeded.

**ERROR\_UNKNOWN\_COMPONENT**

An unknown component was requested.

## Remarks

The **MsiSetComponentState** function requests a change in the Action state of a record in the [Component table](#).

For more information, see [Calling Database Functions From Programs](#).

If the function fails, you can obtain extended error information by using [MsiGetLastErrorRecord](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiSetComponentStateW</b> (Unicode) and <b>MsiSetComponentStateA</b> (ANSI)

## See Also

[Installer Selection Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSetFeatureAttributes Function

The **MsiSetFeatureAttributes** function can modify the default attributes of a feature at runtime. Note that the default attributes of features are authored in the Attributes column of the [Feature table](#).

## Syntax

```
C++UINT MsiSetFeatureAttributes(  
    __in MSIHANDLE hInstall,  
    __in LPCTSTR szFeature,  
    __in DWORD dwAttributes  
);
```

## Parameters

*hInstall* [in]

Handle to the installation provided to a DLL custom action or obtained through [MsiOpenPackage](#), [MsiOpenPackageEx](#), or [MsiOpenProduct](#).

*szFeature* [in]

Specifies the feature name within the product.

*dwAttributes* [in]

Feature attributes specified at run time as a set of bit flags:

Constant	M
INSTALLFEATUREATTRIBUTE_FAVORLOCAL 1	M m: Se de
INSTALLFEATUREATTRIBUTE_FAVORSOURCE 2	M m: Se de
INSTALLFEATUREATTRIBUTE_FOLLOWPARENT	M

4		m: No to; Fe
8	INSTALLFEATUREATTRIBUTE_FAVORADVERTISE	M m: tir fo
16	INSTALLFEATUREATTRIBUTE_DISALLOWADVERTISE	M m: tir fo
32	INSTALLFEATUREATTRIBUTE_NOUNSUPPORTEDADVERTISE	M m: at tal

## Return Value

### ERROR\_SUCCESS

The function returned successfully.

### ERROR\_INVALID\_HANDLE

The product handle is invalid.

### ERROR\_UNKNOWN\_FEATURE

The feature is not known.

### ERROR\_FUNCTION\_FAILED

The function was called at an invalid time during the install, before the [CostInitialize action](#) or after the [CostFinalize action](#).

## Remarks

**MsiSetFeatureAttributes** must be called after [CostInitialize action](#) and

before [CostFinalize action](#). The function returns `ERROR_FUNCTION_FAILED` if called at any other time.

The `INSTALLFEATUREATTRIBUTE_FAVORLOCAL`, `INSTALLFEATUREATTRIBUTE_FAVORSOURCE`, and `INSTALLFEATUREATTRIBUTE_FOLLOWPARENT` flags are mutually exclusive. Only one of these bits can be set for any feature. If more than one of these flags is set, the behavior of that feature is undefined.

See [Calling Database Functions From Programs](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiSetFeatureAttributesW</b> (Unicode) and <b>MsiSetFeatureAttributesA</b> (ANSI)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSetFeatureState Function

The **MsiSetFeatureState** function sets a feature to a specified state.

## Syntax

```
C++UINT MsiSetFeatureState(  
    __in MSIHANDLE hInstall,  
    __in LPCTSTR szFeature,  
    __in INSTALLSTATE iState  
);
```

## Parameters

*hInstall* [in]

Handle to the installation provided to a DLL custom action or obtained through [MsiOpenPackage](#), [MsiOpenPackageEx](#), or [MsiOpenProduct](#).

*szFeature* [in]

Specifies the name of the feature.

*iState* [in]

Specifies the state to set. This parameter can be one of the following values.

Value	Meaning
INSTALLSTATE_ABSENT	The feature is not installed.
INSTALLSTATE_LOCAL	The feature is installed on the local drive.
INSTALLSTATE_SOURCE	The feature is run from the source, CD, or network.
INSTALLSTATE_ADVERTISED	The feature is advertised.

## Return Value

The **MsiSetFeatureState** function returns the following values:

**ERROR\_FUNCTION\_FAILED**

The function failed.

**ERROR\_INVALID\_HANDLE**

An invalid or inactive handle was supplied.

**ERROR\_INVALID\_PARAMETER**

One of the parameters was invalid.

**ERROR\_SUCCESS**

The function succeeded.

**ERROR\_UNKNOWN\_FEATURE**

An unknown feature was requested.

## Remarks

The **MsiSetFeatureState** function requests a change in the select state of a feature in the [Feature](#) table and its children. In turn, the action state of all the components linked to the changed feature records are also updated appropriately, based on the new feature select state.

The [MsiSetInstallLevel](#) function must be called before calling **MsiSetFeatureState**.

When **MsiSetFeatureState** is called, the installer attempts to set the action state of each component tied to the specified feature to the specified state. However, there are common situations when the request cannot be fully implemented. For example, if a feature is tied to two components, component A and component B, through the [FeatureComponents](#) table, and component A has the **msidbComponentAttributesLocalOnly** attribute and component B has the **msidbComponentAttributesSourceOnly** attribute. In this case, if **MsiSetFeatureState** is called with a requested state of either **INSTALLSTATE\_LOCAL** or **INSTALLSTATE\_SOURCE**, the request cannot be fully implemented for both components. In this case, both components are turned ON, with component A set to Local and component B set to Source.

If more than one feature is linked to a single component (a common scenario), the final action state of that component is determined as follows:

- If at least one feature requires the component to be installed locally, the feature is installed with a state of local.
- If at least one feature requires the component to be run from the source, the feature is installed with a state of source.
- If at least one feature requires the removal of the component, the action state is absent.

See [Calling Database Functions from Programs](#).

If the function fails, you can obtain extended error information by using [MsiGetLastErrorRecord](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiSetFeatureStateW</b> (Unicode) and <b>MsiSetFeatureStateA</b> (ANSI)

## See Also

[Installer Selection Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSetInstallLevel Function

The **MsiSetInstallLevel** function sets the installation level for a full product installation.

## Syntax

```
C++UINT MsiSetInstallLevel(  
    __in MSIHANDLE hInstall,  
    __in int iInstallLevel  
);
```

## Parameters

*hInstall* [in]

Handle to the installation that is provided to a DLL custom action or obtained by using [MsiOpenPackage](#), [MsiOpenPackageEx](#), or [MsiOpenProduct](#).

*iInstallLevel* [in]

The installation level.

## Return Value

The **MsiSetInstallLevel** function returns one of the following values:

ERROR\_FUNCTION\_FAILED

The function failed.

ERROR\_INVALID\_HANDLE

An invalid or inactive handle is supplied.

ERROR\_SUCCESS

The function succeeded.

## Remarks

The **MsiSetInstallLevel** function sets the following:

- The installation level for the current installation to a specified value.

- The Select and Installed states for all features in the [Feature table](#).
- The Action state of each component in the [Component table](#), based on the new level.

For any installation, there is a defined install level, which is an integral value from 1 to 32,767. The initial value is determined by the **INSTALLLEVEL** property, which is set in the [Property Table](#).

If 0 (zero) or a negative number is passed in the *installLevel* parameter, the current installation level does not change, but all features are still updated based on the current installation level. For more information, see [Calling Database Functions From Programs](#).

If the function fails, you can obtain extended error information by using [MsiGetLastErrorRecord](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[Installer Selection Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSetMode Function

The [MsiSetMode](#) function sets an internal engine Boolean state.

## Syntax

```
C++UINT MsiSetMode(  
    __in MSIHANDLE hInstall,  
    __in unsigned int iRunMode,  
    __in BOOL fState  
);
```

## Parameters

*hInstall* [in]

Handle to the installation provided to a DLL custom action or obtained through [MsiOpenPackage](#), [MsiOpenPackageEx](#), or [MsiOpenProduct](#).

*iRunMode* [in]

Specifies the run mode. This parameter must be one of the following values. While there are many values for this parameter, as described in [MsiGetMode](#), only one of the following values can be set.

Value	Meaning
MSIRUNMODE_REBOOTATEND	A reboot is necessary after a successful installation.
MSIRUNMODE_REBOOTNOW	A reboot is necessary to continue installation.

*fState* [in]

Specifies the state to set to TRUE or FALSE.

## Return Value

ERROR\_ACCESS\_DENIED

Access was denied.

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_SUCCESS

The function succeeded.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[Installer State Access Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Msi SetProperty Function

The [Msi SetProperty](#) function sets the value for an installation property.

## Syntax

```
C++UINT Msi SetProperty(  
    __in MSIHANDLE hInstall,  
    __in LPCTSTR szName,  
    __in LPCTSTR szValue  
);
```

## Parameters

*hInstall* [in]

Handle to the installation provided to a DLL custom action or obtained through [Msi OpenPackage](#), [Msi OpenPackageEx](#), or [Msi OpenProduct](#).

*szName* [in]

Specifies the name of the property.

*szValue* [in]

Specifies the value of the property.

## Return Value

ERROR\_FUNCTION\_FAILED

The function failed.

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_INVALID\_PARAMETER

An invalid parameter was passed to the function.

ERROR\_SUCCESS

The function succeeded.

## Remarks

If the property is not defined, it is created by the [Msi SetProperty](#) function. If the value is null or an empty string, the property is removed.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>Msi SetPropertyW</b> (Unicode) and <b>Msi SetPropertyA</b> (ANSI)

## See Also

[Installer State Access Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSetTargetPath Function

The **MsiSetTargetPath** function sets the full target path for a folder in the [Directory table](#).

## Syntax

```
C++UINT MsiSetTargetPath(  
    __in MSIHANDLE hInstall,  
    __in LPCTSTR szFolder,  
    __in LPCTSTR szFolderPath  
);
```

## Parameters

*hInstall* [in]

Handle to the installation provided to a DLL custom action or obtained through [MsiOpenPackage](#), [MsiOpenPackageEx](#), or [MsiOpenProduct](#).

*szFolder* [in]

Specifies the folder identifier. This is a primary key in the Directory table.

*szFolderPath* [in]

Specifies the full path for the folder, ending in a directory separator.

## Return Value

The **MsiSetTargetPath** function returns the following values:

ERROR\_DIRECTORY

The directory specified was not found in the [Directory table](#).

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_INVALID\_PARAMETER

An invalid parameter was passed to the function.

ERROR\_SUCCESS

The function succeeded.

## Remarks

The **MsiSetTargetPath** function changes the path specification for the target directory named in the in-memory [Directory table](#). Also, the path specifications of all other path objects in the table that are either subordinate or equivalent to the changed path are updated to reflect the change. The properties for each affected path are also updated.

**MsiSetTargetPath** fails if the selected directory is read only.

If an error occurs in this function, all updated paths and properties revert to their previous values. Therefore, it is safe to treat errors returned by this function as nonfatal.

Do not attempt to configure the target path if the components using those paths are already installed for the current user or for a different user. Check the [ProductState](#) property before calling **MsiSetTargetPath** to determine if the product containing this component is installed.

See [Calling Database Functions From Programs](#).

If the function fails, you can obtain extended error information by using [MsiGetLastErrorRecord](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiSetTargetPathW</b> (Unicode) and <b>MsiSetTargetPathA</b> (ANSI)

## See Also

[Installer Location Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSummaryInfoGetProperty Function

The **MsiSummaryInfoGetProperty** function gets a single property from the [summary information stream](#).

**Note** The meaning of the property value depends on whether the summary information stream is for an installation database (.msi file), transform (.mst file) or patch (.msp file). See [Summary Property Descriptions](#) and [Summary Information Stream Property Set](#) for more information about summary information properties.

## Syntax

```
C++UINT MsiSummaryInfoGetProperty(  
    __in      MSIHANDLE hSummaryInfo,  
    __in      UINT uiProperty,  
    __out     UINT *puiDataType,  
    __out     INT *piValue,  
    __out     FILETIME *pftValue,  
    __out     LPTSTR szValueBuf,  
    __inout   DWORD *pcchValueBuf  
);
```

## Parameters

*hSummaryInfo* [in]

Handle to summary information.

*uiProperty* [in]

Specifies the property ID of the summary property. This parameter can be a property ID listed in the [Summary Information Stream Property Set](#). This function does not return values for PID\_DICTIONARY OR PID\_THUMBNAIL property.

*puiDataType* [out]

Receives the returned property type. This parameter can be a type listed in the [Summary Information Stream Property Set](#).

*piValue* [out]

Receives the returned integer property data.

*pftValue* [out]

Pointer to a file value.

*szValueBuf* [out]

Pointer to the buffer that receives the null terminated summary information property value. Do not attempt to determine the size of the buffer by passing in a null (value=0) for *szValueBuf*. You can get the size of the buffer by passing in an empty string (for example ""). The function then returns `ERROR_MORE_DATA` and *pcchValueBuf* contains the required buffer size in **TCHARs**, not including the terminating null character. On return of `ERROR_SUCCESS`, *pcchValueBuf* contains the number of **TCHARs** written to the buffer, not including the terminating null character. This parameter is an empty string if there are no errors.

*pcchValueBuf* [in, out]

Pointer to the variable that specifies the size, in **TCHARs**, of the buffer pointed to by the variable *szValueBuf*. When the function returns `ERROR_SUCCESS`, this variable contains the size of the data copied to *szValueBuf*, not including the terminating null character. If *szValueBuf* is not large enough, the function returns `ERROR_MORE_DATA` and stores the required size, not including the terminating null character, in the variable pointed to by *pcchValueBuf*.

## Return Value

The **MsiSummaryInfoGetProperty** function returns one of the following values:

`ERROR_INVALID_HANDLE`

An invalid or inactive handle was supplied.

`ERROR_INVALID_PARAMETER`

An invalid parameter was passed to the function.

`ERROR_MORE_DATA`

The buffer passed in was too small to hold the entire value.

ERROR\_SUCCESS

The function succeeded.

ERROR\_UNKNOWN\_PROPERTY

The property is unknown.

## Remarks

If ERROR\_MORE\_DATA is returned, the parameter which is a pointer gives the size of the buffer required to hold the string. If ERROR\_SUCCESS is returned, it gives the number of characters written to the string buffer. Therefore you can get the size of the buffer by passing in an empty string (for example "") for the parameter that specifies the buffer. Do not attempt to determine the size of the buffer by passing in a Null (value=0).

Windows Installer functions that return data in a user provided memory location should not be called with null as the value for the pointer. These functions return a string or return data as integer pointers, but return inconsistent values when passing null as the value for the output argument. For more information, see [Passing Null as the Argument of Windows Installer Functions](#).

The property information returned by the **MsiSummaryInfoGetProperty** function is received by the *piValue*, *pftValue*, or *szValueBuf* parameter depending upon the type of property value that has been specified in the *puiDataType* parameter.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

**Unicode and ANSI  
names**

**MsiSummaryInfoGetPropertyW (Unicode) and  
MsiSummaryInfoGetPropertyA (ANSI)**

## See Also

[Passing Null as the Argument of Windows Installer Functions](#)

[Summary Information Property Functions](#)

[Summary Information Stream Property Set](#)

**[Summaryinfo.Property](#)**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSummaryInfoGetPropertyCount Function

The [MsiSummaryInfoGetPropertyCount](#) function returns the number of existing properties in the summary information stream.

## Syntax

```
C++UINT MsiSummaryInfoGetPropertyCount(  
    __in    MSIHANDLE hSummaryInfo,  
    __out   UINT *puiPropertyCount  
);
```

## Parameters

*hSummaryInfo* [in]

Handle to summary information.

*puiPropertyCount* [out]

Location to receive the total property count.

## Return Value

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_SUCCESS

The function succeeded.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h

<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[Summary Information Property Functions](#)

[Summary Information Stream Property Set](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSummaryInfoPersist Function

The [MsiSummaryInfoPersist](#) function writes changed summary information back to the summary information stream.

## Syntax

```
C++UINT MsiSummaryInfoPersist(  
    __in MSIHANDLE hSummaryInfo  
);
```

## Parameters

*hSummaryInfo* [in]  
Handle to summary information.

## Return Value

ERROR\_FUNCTION\_FAILED  
The function failed.

ERROR\_INVALID\_HANDLE  
An invalid or inactive handle was supplied.

ERROR\_SUCCESS  
The function succeeded.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

---

## See Also

[Summary Information Property Functions](#)  
[Summary Information Stream Property Set](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSummaryInfo SetProperty Function

The [MsiSummaryInfo SetProperty](#) function sets a single summary information property.

**Note** The meaning of the property value depends on whether the summary information stream is for an installation database (.msi file), transform (.mst file) or patch (.msp file). See [Summary Property Descriptions](#) and [Summary Information Stream Property Set](#) for more information about summary information properties.

## Syntax

```
C++UINT MsiSummaryInfoSetProperty(  
    __in MSIHANDLE hSummaryInfo,  
    __in UINT uiProperty,  
    __in UINT uiDataType,  
    __in INT iValue,  
    __in FILETIME *pftValue,  
    __in LPTSTR szValue  
);
```

## Parameters

*hSummaryInfo* [in]

Handle to summary information.

*uiProperty* [in]

Specifies the property ID of the summary property being set. This parameter can be a property ID listed in the [Summary Information Stream Property Set](#). This function does not set values for PID\_DICTIONARY OR PID\_THUMBNAIL property.

*uiDataType* [in]

Specifies the type of property to set. This parameter can be a type listed in the [Summary Information Stream Property Set](#).

*iValue* [in]

Specifies the integer value.

*pftValue* [in]

Specifies the file-time value.

*szValue* [in]

Specifies the text value.

## Return Value

The [MsiSummaryInfo SetProperty](#) function returns the following values:

ERROR\_DATATYPE\_MISMATCH

The data types were mismatched.

ERROR\_FUNCTION\_FAILED

The function failed.

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_INVALID\_PARAMETER

An invalid parameter was passed to the function.

ERROR\_SUCCESS

The function succeeded.

ERROR\_UNKNOWN\_PROPERTY

The summary information property is unknown.

ERROR\_UNSUPPORTED\_TYPE

The type is unsupported.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib

**DLL**

Msi.dll

**Unicode and ANSI  
names**

**MsiSummaryInfo SetPropertyW** (Unicode) and  
**MsiSummaryInfo SetPropertyA** (ANSI)

## See Also

[Summary Information Property Functions](#)  
[Summary Information Stream Property Set](#)  
**[Summaryinfo.Property](#)**

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiVerifyDiskSpace Function

The [MsiVerifyDiskSpace](#) function checks to see if sufficient disk space is present for the current installation.

## Syntax

```
C++UINT MsiVerifyDiskSpace(  
    __in MSIHANDLE hInstall  
);
```

## Parameters

*hInstall* [in]

Handle to the installation provided to a DLL custom action or obtained through [MsiOpenPackage](#), [MsiOpenPackageEx](#), or [MsiOpenProduct](#).

## Return Value

ERROR\_DISK\_FULL

The disk is full.

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_INVALID\_HANDLE\_STATE

The handle is in an invalid state.

ERROR\_SUCCESS

The function succeeded.

## Remarks

See [Calling Database Functions From Programs](#).

## Requirements

Windows Installer 5.0 on Windows Server 2008 R2

<b>Version</b>	or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[Installer Selection Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiViewClose Function

The [MsiViewClose](#) function releases the result set for an executed view.

## Syntax

```
C++UINT MsiViewClose(  
    __in MSIHANDLE hView  
);
```

## Parameters

*hView* [in]

Handle to a view that is set to release.

## Return Value

ERROR\_FUNCTION\_FAILED

The function failed.

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_INVALID\_HANDLE\_STATE

The handle is in an invalid state.

ERROR\_SUCCESS

The function succeeded.

Note that in low memory situations, this function can raise a STATUS\_NO\_MEMORY exception.

## Remarks

The [MsiViewClose](#) function must be called before the [MsiViewExecute](#) function is called again on the view, unless all rows of the result set have been obtained with the [MsiViewFetch](#) function.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[General Database Access Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiViewExecute Function

The **MsiViewExecute** function executes a SQL view query and supplies any required parameters. The query uses the question mark token to represent parameters as described in [SQL Syntax](#). The values of these parameters are passed in as the corresponding fields of a parameter record.

## Syntax

```
C++UINT MsiViewExecute(  
    __in MSIHANDLE hView,  
    __in MSIHANDLE hRecord  
);
```

## Parameters

*hView* [in]

Handle to the view upon which to execute the query.

*hRecord* [in]

Handle to a record that supplies the parameters. This parameter contains values to replace the parameter tokens in the SQL query. It is optional, so *hRecord* can be zero. For a reference on syntax, see [SQL Syntax](#).

## Return Value

ERROR\_FUNCTION\_FAILED

A view could not be executed.

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_SUCCESS

The function succeeded.

Note that in low memory situations, this function can raise a STATUS\_NO\_MEMORY exception.

## Remarks

The **MsiViewExecute** function must be called before any calls to **MsiViewFetch**.

If the SQL query specifies values with parameter markers (?), a record must be supplied that contains all of the replacement values in the exact order and of compatible data types. When used with INSERT and UPDATE queries all the parameterized values must precede all nonparameterized values.

For example, these queries are valid.

```
UPDATE {table-list} SET {column}= ? , {column}= {constant}
```

```
INSERT INTO {table} ({column-list}) VALUES (?, {constant-list})
```

However these queries are invalid.

```
UPDATE {table-list} SET {column}= {constant}, {column}=?
```

```
INSERT INTO {table} ({column-list}) VALUES ({constant-list}, ? )
```

If the function fails, you can obtain extended error information by using **MsiGetLastErrorRecord**.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[General Database Access Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiViewFetch Function

The **MsiViewFetch** function fetches the next sequential record from the view. This function returns a handle that should be closed using [MsiCloseHandle](#).

## Syntax

```
C++UINT MsiViewFetch(  
    __in    MSIHANDLE hView,  
    __out   MSIHANDLE *phRecord  
);
```

## Parameters

*hView* [in]

Handle to the view to fetch from.

*phRecord* [out]

Pointer to the handle for the fetched record.

## Return Value

ERROR\_FUNCTION\_FAILED

An error occurred during fetching.

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_INVALID\_HANDLE\_STATE

The handle was in an invalid state.

ERROR\_NO\_MORE\_ITEMS

No records remain, and a null handle is returned.

ERROR\_SUCCESS

The function succeeded, and a handle to the record is returned.

Note that in low memory situations, this function can raise a STATUS\_NO\_MEMORY exception.

## Remarks

If the **MsiViewFetch** function returns `ERROR_FUNCTION_FAILED`, it is possible that the **MsiViewExecute** function was not called first. If more rows are available in the result set, **MsiViewFetch** returns *phRecord* as a handle to a record containing the requested column data, or *phRecord* is 0. For maximum performance, the same record should be used for all retrievals, or the record should be released by going out of scope.

Note that it is recommended to use variables of type `PMSIHANDLE` because the installer closes `PMSIHANDLE` objects as they go out of scope, whereas you must close `MSIHANDLE` objects by calling **MsiCloseHandle**. For more information see [Use PMSIHANDLE instead of HANDLE](#) section in the [Windows Installer Best Practices](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[General Database Access Functions](#)  
[Working with Queries](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiViewGetColumnInfo Function

The **MsiViewGetColumnInfo** function returns a record containing column names or definitions. This function returns a handle that should be closed using **MsiCloseHandle**.

## Syntax

```
C++UINT MsiViewGetColumnInfo(  
    __in    MSIHANDLE hView,  
    __in    MSICOLINFO eColumnInfo,  
    __out   MSIHANDLE *phRecord  
);
```

## Parameters

*hView* [in]

Handle to the view from which to obtain column information.

*eColumnInfo* [in]

Specifies a flag indicating what type of information is needed. This parameter must be one of the following values.

Value	Meaning
MSICOLINFO_NAMES	Column names are returned.
MSICOLINFO_TYPES	Definitions are returned.

*phRecord* [out]

Pointer to a handle to receive the column information data record.

## Return Value

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_INVALID\_HANDLE\_STATE

The view is not in an active state.

#### ERROR\_INVALID\_PARAMETER

An invalid MSICOLINFO column information enumeration value was passed to the function.

#### ERROR\_SUCCESS

The function succeeded, and a handle to a record was returned.

Note that in low memory situations, this function can raise a STATUS\_NO\_MEMORY exception.

## Remarks

The column description returned by **MsiViewGetColumnInfo** is in the format described in the section: [Column Definition Format](#). Each column is described by a string in the corresponding record field. The definition string consists of a single letter representing the data type followed by the width of the column (in characters when applicable, bytes otherwise). A width of zero designates an unbounded width (for example, long text fields and streams). An uppercase letter indicates that null values are allowed in the column.

Note that it is recommended to use variables of type PMSIHANDLE because the installer closes PMSIHANDLE objects as they go out of scope, whereas you must close MSIHANDLE objects by calling **MsiCloseHandle**. For more information see [Use PMSIHANDLE instead of HANDLE](#) section in the [Windows Installer Best Practices](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiViewGetError Function

The **MsiViewGetError** function returns the error that occurred in the **MsiViewModify** function.

## Syntax

```
C++MSIDBERROR MsiViewGetError(  
    __in     MSIHANDLE hView,  
    __out    LPTSTR szColumnNameBuffer,  
    __inout  DWORD *pcchBuf  
);
```

## Parameters

*hView* [in]

Handle to the view.

*szColumnNameBuffer* [out]

Pointer to the buffer that receives the null-terminated column name. Do not attempt to determine the size of the buffer by passing in a null (value=0) for *szColumnName*. You can get the size of the buffer by passing in an empty string (for example ""). The function then returns MSIDBERROR\_MOREDATA and *pcchBuf* contains the required buffer size in TCHARs, not including the terminating null character. On return of MSIDBERROR\_NOERROR, *pcchBuf* contains the number of TCHARs written to the buffer, not including the terminating null character. This parameter is an empty string if there are no errors.

*pcchBuf* [in, out]

Pointer to the variable that specifies the size, in TCHARs, of the buffer pointed to by the variable *szColumnNameBuffer*. When the function returns MSIDBERROR\_NOERROR, this variable contains the size of the data copied to *szColumnNameBuffer*, not including the terminating null character. If *szColumnNameBuffer* is not large enough, the function returns MSIDBERROR\_MOREDATA and stores the required size, not including the terminating null character, in the variable pointed to by *pcchBuf*.

## Return Value

This function returns one of the following values.

<b>Error code</b>	<b>Meaning</b>
MSIDBERROR_INVALIDARG	An argument was invalid.
MSIDBERROR_MOREDATA	The buffer was too small to receive data.
MSIDBERROR_FUNCTIONERROR	The function failed.
MSIDBERROR_NOERROR	The function completed successfully with no errors.
MSIDBERROR_DUPLICATEKEY	The new record duplicates primary keys of the existing record in a table.
MSIDBERROR_REQUIRED	There are no null values allowed; or the column is about to be deleted, but is referenced by another row.
MSIDBERROR_BADLINK	The corresponding record in a foreign table was not found.
MSIDBERROR_OVERFLOW	The data is greater than the maximum value allowed.
MSIDBERROR_UNDERFLOW	The data is less than the minimum value allowed.
MSIDBERROR_NOTINSET	The data is not a member of the values permitted in the set.
MSIDBERROR_BADVERSION	An invalid version string was supplied.

MSIDBERROR_BADCASE	The case was invalid. The case must be all uppercase or all lowercase.
MSIDBERROR_BADGUID	An invalid GUID was supplied.
MSIDBERROR_BADWILDCARD	An invalid wildcard file name was supplied, or the use of wildcards was invalid.
MSIDBERROR_BADIDENTIFIER	An invalid identifier was supplied.
MSIDBERROR_BADLANGUAGE	Invalid language IDs were supplied.
MSIDBERROR_BADFILENAME	An invalid file name was supplied.
MSIDBERROR_BADPATH	An invalid path was supplied.
MSIDBERROR_BADCONDITION	An invalid conditional statement was supplied.
MSIDBERROR_BADFORMATTED	An invalid format string was supplied.
MSIDBERROR_BADTEMPLATE	An invalid template string was supplied.
MSIDBERROR_BADDEFAULTDIR	An invalid string was supplied in the DefaultDir column of the <a href="#">Directory</a> table.
MSIDBERROR_BADREGPATH	An invalid registry path string was supplied.
MSIDBERROR_BADCUSTOMSOURCE	An invalid string was supplied in the CustomSource column of the <a href="#">CustomAction</a> table.

MSIDBERROR_BADPROPERTY	An invalid property string was supplied.
MSIDBERROR_MISSINGDATA	The <a href="#">_Validation</a> table is missing a reference to a column.
MSIDBERROR_BADCATEGORY	The category column of the <a href="#">_Validation</a> table for the column is invalid.
MSIDBERROR_BADCABINET	An invalid cabinet name was supplied.
MSIDBERROR_BADKEYTABLE	The table in the Keytable column of the <a href="#">_Validation</a> table was not found or loaded.
MSIDBERROR_BADMAXMINVALUES	The value in the MaxValue column of the <a href="#">_Validation</a> table is less than the value in the MinValue column.
MSIDBERROR_BADSHORTCUT	An invalid shortcut target name was supplied.
MSIDBERROR_STRINGOVERFLOW	The string is too long for the length specified by the column definition.
MSIDBERROR_BADLOCALIZEATTRIB	An invalid localization attribute was supplied. (Primary keys cannot be localized.)

Note that in low memory situations, this function can raise a STATUS\_NO\_MEMORY exception.

## Remarks

You should only call the **MsiViewGetError** function when **MsiViewModify** returns `ERROR_INVALID_DATA`, indicating that the data is invalid. Errors are only recorded for `MSIMODIFY_VALIDATE`, `MSIMODIFY_VALIDATE_NEW`, and `MSIMODIFY_VALIDATEFIELD`.

If `ERROR_MORE_DATA` is returned, the parameter that is a pointer gives the size of the buffer required to hold the string. Upon success, it gives the number of characters written to the string buffer. Therefore you can get the required size of the buffer by passing a small buffer (one character minimum) and examining the value at *pcchPathBuf* when the function returns `MSIDBERROR_MOREDATA`. Do not attempt to determine the size of the buffer by passing in null as *szColumnNameBuffer* or a buffer size of 0 in the **DWORD** referenced by *pcchBuf*.

Once `MSIDBERROR_NOERROR` is returned, no more validation errors remain. The `MSIDBERROR` return value indicates the type of validation error that occurred for the value located in the column identified by the *szColumnNameBuffer*.

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll
<b>Unicode and ANSI names</b>	<b>MsiViewGetErrorW</b> (Unicode) and <b>MsiViewGetErrorA</b> (ANSI)

## See Also

General Database Access Functions  
Passing Null as the Argument of Windows Installer Functions

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiViewModify Function

The **MsiViewModify** function updates a fetched record.

## Syntax

```
C++UINT MsiViewModify(  
    __in MSIHANDLE hView,  
    __in MSIMODIFY eModifyMode,  
    __in MSIHANDLE hRecord  
);
```

## Parameters

*hView* [in]

Handle to a view.

*eModifyMode* [in]

Specifies the modify mode. This parameter can be one of the following values.

Value	Meaning
MSIMODIFY_SEEK -1	Refreshes the information in the supplied record without changing the position in the result set and without affecting subsequent fetch operations. The record may then be used for subsequent Update, Delete, and Refresh. All primary key columns of the table must be in the query and the record must have at least as many fields as the query. Seek cannot be used with multi-table queries. This mode cannot be used with a view containing joins. See

	also the remarks.
MSIMODIFY_REFRESH 0	Refreshes the information in the record. Must first call <b>MsiViewFetch</b> with the same record. Fails for a deleted row. Works with read-write and read-only records.
MSIMODIFY_INSERT 1	Inserts a record. Fails if a row with the same primary keys exists. Fails with a read-only database. This mode cannot be used with a view containing joins.
MSIMODIFY_UPDATE 2	Updates an existing record. Nonprimary keys only. Must first call <b>MsiViewFetch</b> . Fails with a deleted record. Works only with read-write records.
MSIMODIFY_ASSIGN 3	Writes current data in the cursor to a table row. Updates record if the primary keys match an existing row and inserts if they do not match. Fails with a read-only database. This mode cannot be used with a view containing joins.
MSIMODIFY_REPLACE 4	Updates or deletes and inserts a record into a table. Must first call <b>MsiViewFetch</b> with the same record. Updates record if the primary keys are unchanged. Deletes old row

	and inserts new if primary keys have changed. Fails with a read-only database. This mode cannot be used with a view containing joins.
MSIMODIFY_MERGE 5	Inserts or validates a record in a table. Inserts if primary keys do not match any row and validates if there is a match. Fails if the record does not match the data in the table. Fails if there is a record with a duplicate key that is not identical. Works only with read-write records. This mode cannot be used with a view containing joins.
MSIMODIFY_DELETE 6	Remove a row from the table. You must first call the <b>MsiViewFetch</b> function with the same record. Fails if the row has been deleted. Works only with read-write records. This mode cannot be used with a view containing joins.
MSIMODIFY_INSERT_TEMPORARY 7	Inserts a temporary record. The information is not persistent. Fails if a row with the same primary key exists. Works only with read-write records. This mode cannot be used with a view containing joins.
MSIMODIFY_VALIDATE	Validates a record. Does not

8	<p>validate across joins. You must first call the <b>MsiViewFetch</b> function with the same record. Obtain validation errors with <b>MsiViewGetError</b>. Works with read-write and read-only records. This mode cannot be used with a view containing joins.</p>
MSIMODIFY_VALIDATE_NEW 9	<p>Validate a new record. Does not validate across joins. Checks for duplicate keys. Obtain validation errors by calling <b>MsiViewGetError</b>. Works with read-write and read-only records. This mode cannot be used with a view containing joins.</p>
MSIMODIFY_VALIDATE_FIELD 10	<p>Validates fields of a fetched or new record. Can validate one or more fields of an incomplete record. Obtain validation errors by calling <b>MsiViewGetError</b>. Works with read-write and read-only records. This mode cannot be used with a view containing joins.</p>
MSIMODIFY_VALIDATE_DELETE 11	<p>Validates a record that will be deleted later. You must first call <b>MsiViewFetch</b>. Fails if another row refers to the primary keys of this row. Validation does not check for the existence of the primary</p>

keys of this row in properties or strings. Does not check if a column is a foreign key to multiple tables. Obtain validation errors by calling **MsiViewGetError**. Works with read-write and read-only records. This mode cannot be used with a view that contains joins.

*hRecord* [in]

Handle to the record to modify.

## Return Value

The **MsiViewModify** function returns the following values:

ERROR\_ACCESS\_DENIED

Access was not permitted.

ERROR\_FUNCTION\_FAILED

The function failed.

ERROR\_INVALID\_DATA

A validation was requested and the data did not pass. For more information, call **MsiViewGetError**.

ERROR\_INVALID\_HANDLE

An invalid or inactive handle was supplied.

ERROR\_INVALID\_HANDLE\_STATE

The handle is in an invalid state.

ERROR\_INVALID\_PARAMETER

An invalid parameter was passed to the function.

ERROR\_SUCCESS

The function succeeded.

Note that in low memory situations, this function can raise a STATUS\_NO\_MEMORY exception.

## Remarks

The `MSIMODIFY_VALIDATE`, `MSIMODIFY_VALIDATE_NEW`, `MSIMODIFY_VALIDATE_FIELD`, and `MSIMODIFY_VALIDATE_DELETE` values of the **MsiViewModify** function do not perform actual updates; they ensure that the data in the record is valid. Use of these validation enumerations requires that the database contains the [\\_Validation table](#).

You can call `MSIMODIFY_UPDATE` or `MSIMODIFY_DELETE` with a record immediately after using `MSIMODIFY_INSERT`, `MSIMODIFY_INSERT_TEMPORARY`, or `MSIMODIFY_SEEK` provided you have NOT modified the 0th field of the inserted or sought record.

To execute any SQL statement, a view must be created. However, a view that does not create a result set, such as `CREATE TABLE`, or `INSERT INTO`, cannot be used with **MsiViewModify** to update tables through the view.

You cannot fetch a record that contains binary data from one database and then use that record to insert the data into another database. To move binary data from one database to another, you should export the data to a file and then import it into the new database using a query and the [MsiRecordSetStream](#). This ensures that each database has its own copy of the binary data.

Note that custom actions can only add, modify, or remove temporary rows, columns, or tables from a database. Custom actions cannot modify persistent data in a database, such as data that is a part of the database stored on disk. For more information, see [Accessing the Current Installer Session from Inside a Custom Action](#).

If the function fails, you can obtain extended error information by using [MsiGetLastErrorRecord](#).

## Requirements

<b>Version</b>	Windows Installer 5.0 on Windows Server 2008 R2 or Windows 7. Windows Installer 4.0 or Windows Installer 4.5 on Windows Server 2008 or Windows Vista. Windows Installer on Windows Server 2003, Windows XP, and Windows 2000
----------------	--

<b>Header</b>	Msiquery.h
<b>Library</b>	Msi.lib
<b>DLL</b>	Msi.dll

## See Also

[General Database Access Functions](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Database Tables

The following table identifies the Windows Installer database tables.

Table	Description
ActionText	Lists text in a progress dialog box or action log.
AdminExecuteSequence	Lists ADMIN actions in sequence.
AdminUISequence	Lists UI ADMIN actions in sequence.
AdvtExecuteSequence	Lists ADVERTISE actions in sequence.
AdvtUISequence	The Windows Installer does not use this table. The AdvtUISequence table should not exist in the installation database or it should be left empty.
AppId	Specifies that the installer configure and register DCOM servers
AppSearch	Lists properties used to search by file signature.
BBControl	Lists controls displayed on each billboard.
Billboard	Lists billboards displayed in full UI.
Binary	Holds binary data for bitmaps and icons.
BindImage	Lists executables bound to the DLLs.
CCPSearch	Lists file signatures for the Compliance Checking Program.
CheckBox	Lists the values for the check boxes.
Class	Lists COM server information for product advertisement.
ComboBox	Lists the values for each combo box.
CompLocator	Find file or directory by installer configuration data.

Complus	Contains information needed to install COM+ applications.
Component	Lists installation components.
Condition	Modifies feature selection states conditionally.
Control	Lists the controls on each dialog box.
ControlCondition	Lists actions applied to controls based on a property.
ControlEvents	Specifies the actions of controls in dialog boxes.
CreateFolder	Lists folders that must be created for a component.
CustomAction	Integrates custom actions into the installation.
Dialog	Lists dialog boxes in the user interface.
Directory	Directory layout for the application.
DrLocator	Lists file searches using the directory tree.
DuplicateFile	Lists files to be duplicated.
Environment	Lists the environment variables.
Error	Lists error message formatting templates.
EventMapping	Lists the events subscribed to by controls.
Extension	Lists file name extension server information.
Feature	Defines the logical tree structure of features.
FeatureComponents	Defines features and component relationships.
File	Complete list of source files with their attributes.
FileSFPCatalog	Associates specified files with the catalog

	files.
Font	Registry information for font files.
Icon	Contains the icon files.
IniFile	Information needed to set in an .ini file.
IniLocator	Searches for file or directory using an .ini file.
InstallExecuteSequence	Lists INSTALL actions in sequence.
InstallUISequence	Lists UI INSTALL actions in sequence.
IsolatedComponent	Lists Isolated Components.
LaunchCondition	Lists conditions for the installation to begin.
ListBox	Lists the values for all list boxes.
ListView	Lists the values for all listviews.
LockPermissions	Defines locked-down portions of the application.
Media	Lists source media disks for the installation.
MIME	Lists MIME content type, file name extension, or CLSID.
MoveFile	Lists files to be moved or copied.
MsiAssembly	Specifies Windows Installer settings for Microsoft .NET Framework assemblies and Win32 assemblies.
MsiAssemblyName	Specifies the schema for the elements of a strong assembly cache name for a .NET Framework or Win32 assembly.
MsiDigitalCertificate	Stores certificates in binary stream format and associates each certificate with a primary key.

<a href="#">MsiDigitalSignature</a>	Contains the signature information for every digitally-signed object in the installation database.
<a href="#">MsiEmbeddedChainer</a>	Each row in this table references a different user-defined function that can be used to install multiple Windows Installer packages from a single package.  <b>Windows Installer 4.0 and earlier:</b> Not supported.
<a href="#">MsiEmbeddedUI</a>	Defines a user interface embedded in the Windows Installer package.  <b>Windows Installer 4.0 and earlier:</b> Not supported.
<a href="#">MsiFileHash</a>	Stores a 128-bit hash of source files provided by the Windows Installer package.
<a href="#">MsiLockPermissionsEx Table</a>	Secures services, files, registry keys, and created folders.  <b>Windows Installer 4.5 and earlier:</b> Not supported.
<a href="#">MsiPackageCertificate</a>	Lists digital signature certificates being used to verify the identity of installation packages that make this <a href="#">Multiple-Package Installation</a> . Available beginning with Windows Installer 4.5.
<a href="#">MsiPatchCertificate</a>	Contains the information needed to enable <a href="#">User Account Control (UAC) Patching</a> . Available beginning with Windows Installer 3.0.
<a href="#">MsiPatchHeaders</a>	Holds the binary patch header streams

	used for patch validation.
<a href="#">MsiPatchMetadata</a>	Holds information about a Windows Installer patch required to remove the patch and used by Add/Remove Programs. Available beginning with Windows Installer 3.0.
<a href="#">MsiPatchOldAssemblyName</a>	Specifies the old name for an assembly.
<a href="#">MsiPatchOldAssemblyFile</a>	Relates a file in the <a href="#">File Table</a> to an assembly name.
<a href="#">MsiPatchSequence</a>	Contains the information required to determine the sequence of application of a small update patch relative to all other patches. Available beginning with Windows Installer 3.0.
<a href="#">MsiServiceConfig</a>	Configures a service that is installed or being installed by the current package.  <b>Windows Installer 4.5 and earlier:</b> Not supported.
<a href="#">MsiServiceConfigFailureActions</a>	Lists operations to be run after a service fails. The operations specified in this table run the next time the system is started.  <b>Windows Installer 4.5 and earlier:</b> Not supported.
<a href="#">MsiSFCBypass</a>	Lists files that should bypass Windows File Protection on Windows Me.
<a href="#">ODBCAttribute</a>	Lists attributes of ODBC drivers and translators.
<a href="#">ODBCDataSource</a>	Lists data sources belonging to the installation.
<a href="#">ODBCDriver</a>	Lists ODBC drivers belonging to the

	installation.
ODBCSourceAttribute	Lists the attributes of data sources.
ODBCTranslator	Lists ODBC translators of the installation.
Patch	Lists files that are to receive a particular patch.
PatchPackage	Lists all patch packages applied to this product.
ProgId	Lists information for program IDs.
Property	Lists property names and values for all properties.
PublishComponent	Lists information used for component publishing.
RadioButton	Lists buttons for all the radio button groups.
Registry	Lists registry information for the application.
RegLocator	Searches for file or directory using the registry.
RemoveFile	Lists files to be removed by RemoveFiles action.
RemoveIniFile	Lists information needed to delete from an .ini file.
RemoveRegistry	Lists information needed to delete from system registry.
ReserveCost	Reserves disk space in any directory conditionally.
SelfReg	Lists information about self-registered modules.
ServiceControl	Controls installed or uninstalled services.
ServiceInstall	Lists information used to install a service.

<a href="#">SFPCatalog</a>	Lists SFP catalogs.
<a href="#">Shortcut</a>	Lists information needed to create shortcuts.
<a href="#">Signature</a>	Lists the unique file signatures that identify files.
<a href="#">TextStyle</a>	Lists text styles used in the text controls.
<a href="#">TypeLib</a>	Lists registry information for type libraries.
<a href="#">UIText</a>	Lists localized versions of some strings used in the user interface.
<a href="#">Verb</a>	Lists command-verb information for file name extensions.
<a href="#">_Validation</a>	Lists column names and values for all tables.
<a href="#">_Columns</a>	Read-only column catalog.
<a href="#">_Streams</a>	Lists embedded OLE data streams.
<a href="#">_Storages</a>	Lists embedded OLE sub-storages.
<a href="#">_Tables</a>	Read-only system table listing all the tables.
<a href="#">_TransformView Table</a>	A read-only temporary table used to view transforms.
<a href="#">Upgrade</a>	Lists information used in an upgrade of an application.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Column Data Types

The columns of the database tables are formatted in one of the following data types:

- Text
- UpperCase
- LowerCase
- Integer
- DoubleInteger
- Time/Date
- Identifier
- Property
- Filename
- WildCardFilename
- Path
- Paths
- AnyPath
- DefaultDir
- RegPath
- Formatted
- FormattedSDDLText
- Template
- Condition
- GUID
- Version
- Language
- Binary
- CustomSource

- [Cabinet](#)
- [Shortcut](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Text

The Text data type is a text string. It is not validated.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UpperCase

The UpperCase data type is a text string that must be all uppercase.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# LowerCase

The LowerCase data type is a text string that must be all lowercase.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Integer

The Integer data type is a two-byte integer value. Unless otherwise restricted, the range of legal values is from  $-32,767$  to  $+32,767$ .

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DoubleInteger

The DoubleInteger data type is a four-byte integer value. Unless otherwise restricted, the range of legal values is from  $-2,147,483,647$  to  $+2,147,483,647$ .

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Time/Date

The Time/Date data type has the time and the date stored individually, using unsigned integers as bit fields, packed as follows.

## Time

Time is encoded in an unsigned 2-byte integer with the following bit fields.

Contents	Bits	Value range
hours	0 1 2 3 4	0–23
minutes	5 6 7 8 9 A	0–59
2-second intervals	B C D E F	0–29

## Date

Date is encoded in an unsigned 2-byte integer with the following bit fields.

Contents	Bits	Value range
year	0 1 2 3 4 5 6	0–119 (relative to 1980)
month	7 8 9 A	1–12
day	B C D E F	1–31

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Identifier

The Identifier data type is a text string. Identifiers may contain the ASCII characters A-Z (a-z), digits, underscores (\_), or periods (.). However, every identifier must begin with either a letter or an underscore.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Property

The Property data type is a valid [Identifier](#) with the additional syntax "*%identifier*", which represents an environment variable.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Filename

The Filename data type is a text string containing a file name or folder. By default, the file name is assumed to use short file name syntax; that is, eight-character name, period (.), and 3-character extension. A short file name must always be provided because the **SHORTFILENAMES** property may be set or the target volume for the installation may only support short file names.

To include a long file name with the short file name, separate it from the short file name with a vertical bar (|).

For example, the following two strings are valid:

- status.txt
- projec~1.txt|Project Status.txt

Short and long file names must not contain the following characters:

- backward slash (/)
- question mark (?)
- vertical bar (|)
- right angle bracket (>)
- left angle bracket (<)
- colon (:)
- forward slash (\)
- asterisk (\*)
- quotation mark (")

In addition, short file names must not contain the following characters:

- plus sign (+)
- comma (,)
- semicolon (;)
- equals sign (=)

- left square bracket ([)
- right square bracket (])

No space is allowed preceding the vertical bar (|) separator for the short file name/long file name syntax. Short file names may not include a space, although a long file name may. A space can exist after the separator only if the long file name of the file name begins with the space. No full-path syntax is allowed.

**Note** The format of the FileName column of the [MsiEmbeddedUI](#) table is like the format Filename data type except that the vertical bar (|) separator for the short file name/long file name syntax is not available.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# WildcardFilename

The WildCardFilename data type is a [Filename](#) that may also contain the wild card characters "?" for any single character or "\*" for zero or more occurrences of any character.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Path

The Path data type is a text string containing a valid full path.

The string may also contain a property name enclosed in square brackets [ ]. In such a case, the name of the property, including the brackets, is replaced in the string by the value of the property.

Examples:

- UNC path: \\server\share
- Local drive: c:\temp
- With a property name: [DRIVE]\temp

## See Also

[Paths](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Paths

The Paths data type is a text string containing a list of [Path](#) data types, separated by semicolons (;).

## See Also

[Path](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# AnyPath

The AnyPath data type is a text string containing either a full path or a relative path. When specifying a relative path, you can include a long file name with the short file name by separating the short and long names with a vertical bar (|). Note that you cannot specify multiple levels of a directory or fully qualified paths in this way. The path may contain properties enclosed within square brackets ([ ]).

Examples of valid AnyPath data:

- \\server\share\temp
- c:\temp
- \temp
- projec~1|Project Status

Examples of invalid AnyPath data:

- c:\temp\projec~1|c:\temp one\Project Status
- \temp\projec~1|\temp one\Project Status

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DefaultDir

The DefaultDir data type is a text string containing either a valid [Filename](#) or a valid [Identifier](#). This is used only in the [Directory table](#). It must be an identifier if the directory is a root directory. If the directory is a non-root, this value must be a filename or a filename:filename pair. Note that "." is allowed as a file name and has special meaning in the Directory table.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RegPath

The RegPath data type is a text string containing a Registry path. Registry paths can include properties, as with the [Formatted](#) data type. A RegPath may not begin or end with a backslash (\). The [#file key] and [\$component key] can be embedded in the path or preceded by other characters.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Formatted

The Formatted data type is a text string that is processed to resolve embedded property names, table keys, environment variable references, and other special substrings. The following conventions are recognized to resolve the string:

- Square brackets ([ ]) or curly braces ({ }) with no matching pair are left in the text.
- If a substring of the form [*propertyname*] is encountered, it is replaced by the value of the property. If *propertyname* is not a valid property name, then the substring resolves as blank. For example, the Description column of the [LaunchCondition Table](#) takes a Formatted string. If ERRORTXT has been set to "Please contact your support personnel." then the text displayed for failing the launch condition would include this string. If ERRORTXT is not set then the text displayed for failing the launch condition would be just "System does not meet installation requirements."

Condition	Description
Version9X	System does not meet installation requirements. [ERRORTXT]

- The square brackets may be iterated and the property names are resolved from inside out. For example, suppose the substring [[PropertyA]] appears in the text. First, the value of property PropertyA is retrieved. If the value is a valid property name, such as PropertyB, then the value of PropertyB is retrieved, and the entire substring [[PropertyA]] is substituted with the value of PropertyB. If PropertyA is not a valid property name, or if the value of PropertyA is not a valid property name, then the substring is blank.

- If a substring of the form [%environmentvariable] is found, the value of the environment variable is substituted for the substring.
- If a substring of the form [x] is found, it is replaced by the character x, where x is one character, without any further processing. Only the first character after the backslash is kept; everything else is removed. For example, to include a literal left bracket ([), use [\[]. The text [\[Bracket Text\]] resolves to [Bracket Text].
- If a substring is enclosed in curly braces ({ }), and it contains no property names enclosed in square brackets ([ ]), the substring is left unchanged, including the curly braces.
- If a substring is enclosed in curly braces ({ }) and it contains one or more property names enclosed in square brackets ([ ]) then, if all the property names are valid, the text (with the resolved substitutions) is displayed without the curly braces.
- If a substring of the form [~] is found, it is replaced with the null character. This is used to author **REG\_MULTI\_SZ** character strings in the [Registry table](#). Note that [~] is also used to append or prefix values to environment variables using the [Environment table](#).
- If a substring of the form [#filekey] is found, it is replaced by the full path of the file, with the value filekey used as a key into the [File table](#). The value of [#filekey] remains blank and is not replaced by a path until the installer runs the [CostInitialize action](#), [FileCost action](#), and [CostFinalize action](#). The value of [#filekey] depends upon the installation state of the component to which the file belongs. If the component is run from the source, the value is the path to the source location of the file. If the component is run locally, the value is the path to the target location of the file after installation. If the component has an action state of absent, the installed state of the component is used to determine the [#filekey] value. If the installed state of the component is also absent or null, [#filekey] resolves to an empty string, otherwise it resolves to the value based upon the

component's installed state. For more information about checking the installation state of components, see [Checking the Installation of Features, Components, Files](#).

- If a substring of the form [*\$componentkey*] is found, it is replaced by the install directory of the component, with the value *componentkey* used as a key into the [Component table](#). The value of [*\$componentkey*] remains blank and is not replaced by a directory until the installer runs the [CostInitialize action](#), [FileCost action](#), and [CostFinalize action](#). The value of [*\$componentkey*] depends upon the installation state of the component and where it occurs. In the Value column of the [Registry Table](#), this substring may refer to the action state or the requested action state of the component. In all other cases, this substring refers to the action state of the component. For example, if the component is run from the source, the value is the source directory of the file. If the component is run locally, the value is the target directory after installation. If the component is absent, the value is left blank. Windows Installer tracks both the action and requested installation states of components. For example, if a component is already installed, it may have a requested state of local and an action state of null. For more information about checking the installation state of components, see [Checking the Installation of Features, Components, Files](#).
- Note that if a component is already installed, and is not reinstalled, removed, or moved during the current installation, the action state of the component is null and the string [*\$componentkey*] evaluates to Null.
- If a substring of the form [*!filekey*] is found, it is replaced by the full short path of the file, with the value *filekey* used as a key into the [File table](#).

This syntax is valid only when used in the Value column of the Registry or the IniFile tables. When used in other columns this

syntax is treated the same as [#filekey] .

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# FormattedSDDLText

A database field of the **FormattedSDDLText** data type holds a text string that describes a security descriptor using valid security descriptor definition language (SDDL.) This data type is used by the SDDLText field of the [MsiLockPermissionsEx Table](#) to secure a selected object.

**Windows Installer 4.5 or earlier:** Not supported. This datatype is available beginning with Windows Installer 5.0.

The **FormattedSDDLText** data type can hold a SDDL string written in valid Security Descriptor String Format. For more information about SDDL, see the Access Control section of the [Microsoft Windows Software Development Kit \(SDK\)](#). In addition, a **FormattedSDDLText** text string can use angle brackets (<>) to contain the domain and username of the user whose account SID is to be determined.

If the user having user name *SampleUser* belongs to a domain named *SampleDomain*, then the **FormattedSDDLText** value can identify the owner using the SID string, the user name and domain name, or the Windows environment variables. For example, the following strings would be possible.

```
O:owner_sid_stringG:BAD:(D;OICI;GA;;;BG)
(A;OICI;GRGWGX;;;owner_sid_string)
(A;OICI;GA;;;BA)S:ARAI(AU;SAFA;FA;;;WD)
O:<SampleDomain\SampleUser>G:BAD:(D;OICI;GA;;;BG)
(A;OICI;GRGWGX;;;<SampleDomain\SampleUser>)
(A;OICI;GA;;;BA)S:ARAI(AU;SAFA;FA;;;WD)
O:<[%USERDOMAIN]\[%USERNAME]>G:BAD:(D;OICI;GA;;;BG)
(A;OICI;GRGWGX;;;<[%USERDOMAIN]\[%USERNAME]>)
(A;OICI;GA;;;BA)S:ARAI(AU;SAFA;FA;;;WD)
```

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Template

The Template data type is a text string that may contain properties that are enclosed in brackets [ ]. The template type allows all of the [Formatted](#) type formats, plus [1] where 1 is a number.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Condition

The Condition data type is a text string containing a valid conditional statement that can be evaluated as true or false. For information on the syntax of conditional statements, see [Conditional Statement Syntax](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# GUID

The GUID data type is a text string representing a Class identifier (ID). COM must be able to convert the string to a valid Class ID. All GUIDs must be authored in uppercase.

The valid format for a GUID is {XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX} where X is a hex digit (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F).

Note that utilities such as GUIDGEN can generate GUIDs containing lowercase letters. These must all be changed to uppercase letters before the GUID can be used by the installer as a valid product code, package code, or component code.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Version

The Version data type is a text string containing a valid version string. A version string has the format

```
XXXXX.XXXXX.XXXXX.XXXXX
```

where x is a digit.

The maximum acceptable version string is 65535.65535.65535.65535.

The following are examples of valid version strings:

- 1
- 1.0
- 1.00
- 10.00
- 1.00.1
- 1.0.1
- 1.00.10
- 1.00.100
- 1.0.1000.0

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Language

The Language data type is a text string containing one or more valid numeric language IDs. If there are two or more language IDs, they must be separated by commas.

The Language data type is a 16-bit value that is the combination of a primary and sublanguage numeric IDs. The Primary LANGID comprises bits 0 through 9 while the subLanguage ID is bits 10 through 15. For a list of primary and sub language numeric identifiers, see the Language Identifier Constants and Strings topic.

For primary language IDs, the range 0x200 to 0x3ff is user definable. The range 0x000 to 0x1ff is reserved for system use. For sublanguage IDs, the range 0x20 to 0x3f is user definable. The range 0x00 to 0x1f is reserved for system use.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Binary

The Binary data type is a binary data stream containing any type of binary data. Examples include a bitmap or executable code.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# CustomSource

The CustomSource data type is either a valid [Identifier](#) or an external key depending on the Type column of the [CustomAction table](#). It can be an external key into the [Binary table](#), [File table](#), [Directory table](#), and [Property table](#). Note that external keys into the Property Table are not validated because properties can be added at run-time.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Cabinet

The Cabinet data type must be used in the Cabinet column of the [Media table](#).

If the cabinet name is preceded by the number sign, the cabinet is stored as a data stream inside the package. The character string which follows the # is an [Identifier](#) for this data stream. Note that if the cabinet is stored as a data stream, the name of a cabinet is case-sensitive.

If the cabinet name is not preceded by the number sign #, the cabinet is stored in a separate file located at the root of the source tree specified by the [Directory Table](#). The cabinet file must use the short file name syntax consisting of an eight character name, a period, and a three character extension. The Cabinet data type cannot use the short|longname syntax for file names. If the cabinet file is stored as a separate file, the name of the cabinet file is not case-sensitive.

To conserve disk space, the installer removes any cabinets embedded in the .msi file before caching the installation package on the user's computer.

See [Including a Cabinet File in an Installation](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Shortcut

The Shortcut data type is usually used in the Target column of the [Shortcut table](#). If it contains square brackets ([ ]), the shortcut target is evaluated as a Formatted string. Otherwise, the shortcut is evaluated as an [Identifier](#) and must be a valid foreign key into the [Feature table](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RowState Column

The reserved column name `_RowState` represents the non-persistent state associated with each table row in an installer database table. The pseudo-column is of type `Integer`, and the value consists of a set of bit flags. All the bits are readable, but only the `UserInfo` and `Temporary` bits can be set. This data is available only as long as the tables are locked or in use (that is, while a view containing the tables exists). The following table shows the attributes a row can have.

Name	Value	Meaning
<code>iraUserInfo</code>	<code>0x01</code>	The attribute is for external use. This value can be updated.
<code>iraTemporary</code>	<code>0x02</code>	The row is not persistent. This value can be updated.
<code>iraModified</code>	<code>0x04</code>	A row has been updated.
<code>iraInserted</code>	<code>0x08</code>	A row has been inserted.
<code>iraMergeFailed</code>	<code>0x0F</code>	An attempt was made to merge with non-identical, non-key data.

Bits 6 through 8 are reserved.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ActionText Table

The ActionText Table contains text to be displayed in a progress dialog box, and written to the log for actions that take a long time to execute. The displayed text consists of the action description and optionally formatted data from the action.

The ActionText Table has the following columns.

Column	Type	Key	Nullable
Action	Identifier	Y	N
Description	Text	N	Y
Template	Template	N	Y

## Columns

### Action

Name of the action.

Primary table key.

### Description

Localized description that is displayed in the progress dialog box, or written to the log when the action is executing.

### Template

A localized format template that is used to format action data records to display during action execution. If a template is not supplied, then the action data is not displayed.

## Remarks

Typically, the entries in the ActionText table refer to actions in sequence tables. There are other actions the installer performs that are not listed in the sequence table, but still need to be specified in the table. The following table identifies the required table entries where the action name

and template must be authored exactly as shown, but the description can be customized.

Action	Description	Template
Rollback	Undoes actions.	[1]
RollbackCleanup	Removes old files.	[1]
GenerateScript	Generates system operations for action.	[1]

**Note** Action text is only displayed for actions that run within the installation script. Therefore, action text is only displayed for actions that are sequenced between the [InstallInitialize](#) and [InstallFinalize](#) actions.

You can import a localized ActionText table into your database by using Msidb.exe or [MsiDatabaseImport](#). The SDK includes a localized ActionText Table for each of the languages listed in the [Localizing the Error and ActionText Tables](#) section. If the ActionText table is not populated, the installer loads localized strings for the language specified by the [ProductLanguage](#) property.

## Validation

[ICE03](#)

[ICE06](#)

[ICE46](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# AdminExecuteSequence Table

The AdminExecuteSequence table lists actions that the installer calls in sequence when the top-level [ADMIN action](#) is executed.

ADMIN actions in the install sequence, up to the [InstallValidate action](#) and any exit dialog boxes, are located in the [AdminUISequence table](#).

ADMIN actions from the InstallValidate action through the end of the install sequence are in the AdminExecuteSequence table. Because the AdminExecuteSequence table needs to stand alone, it also contains any necessary initialization actions such as [LaunchConditions](#), [CostInitialize](#), [FileCost](#), and [CostFinalize](#).

[Custom actions](#) requiring a user interface should use [MsiProcessMessage](#) instead of authored dialog boxes created using the [Dialog table](#).

The columns are identical to those of the [InstallExecuteSequence table](#). The AdminExecuteSequence table has the following columns.

Column	Type	Key	Nullable
Action	<a href="#">Identifier</a>	Y	N
Condition	<a href="#">Condition</a>	N	Y
Sequence	<a href="#">Integer</a>	N	Y

## Columns

### Action

Name of the action to execute. This is either a standard action or a custom action listed in the [CustomAction table](#).

Primary table key.

### Condition

Logical expression. If the expression evaluates to false, the action is skipped. If the expression syntax is invalid, the sequence terminates, returning `iesBadActionData`. For information on the syntax of conditional statements see [Conditional Statement Syntax](#).

## Sequence

A positive value indicates the sequence position of the action. The following negative values indicate that the action is called if the installer returns the termination flag. Each termination flag (negative value) can be used with no more than one action. Multiple actions can have termination flags, but they must be different flags. Termination flags (negative values) are typically used with [Dialog Boxes](#).

Termination flag	Value	Description
msiDoActionStatusSuccess	-1	Successful completion. Used with <a href="#">Exit</a> dialog boxes.
msiDoActionStatusUserExit	-2	User terminates install. Used with <a href="#">UserExit</a> dialog boxes.
msiDoActionStatusFailure	-3	Fatal exit terminates. Used with a <a href="#">FatalError</a> dialog boxes.
msiDoActionStatusSuspend	-4	Install is suspended.

Zero, all other negative numbers, or a null value indicate that the action is never called.

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE12](#)  
[ICE13](#)  
[ICE26](#)  
[ICE27](#)  
[ICE28](#)  
[ICE75](#)  
[ICE77](#)  
[ICE79](#)  
[ICE82](#)

ICE84  
ICE86  
ICEM04

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# AdminUISequence Table

The AdminUISequence table lists actions that the installer calls in sequence when the top-level [ADMIN action](#) is executed and the internal user interface level is set to full UI or reduced UI. The installer skips the actions in this table if the user interface level is set to basic UI or no UI. See [About the User Interface](#).

ADMIN actions in the install sequence up to the [InstallValidate action](#), and any exit dialog boxes, are located in the AdminUISequence table. All actions from the InstallValidate through the end of the install sequence are in the [AdminExecuteSequence table](#). Because the AdminExecuteSequence table needs to stand alone, it also contains any necessary initialization actions such as [LaunchConditions](#), [CostInitialize](#), [FileCost](#), and [CostFinalize](#). It also has the [ExecuteAction action](#).

The columns are identical to those of the [InstallUISequence table](#). The AdminUISequence table has the following columns.

Column	Type	Key	Nullable
Action	<a href="#">Identifier</a>	Y	N
Condition	<a href="#">Condition</a>	N	Y
Sequence	<a href="#">Integer</a>	N	Y

## Columns

### Action

Name of the action to execute. This is either a standard action, a user interface wizard, or a custom action listed in the [CustomAction table](#).

Primary table key.

### Condition

Logical expression. If the expression evaluates to false, the action is skipped. If the expression syntax is invalid, the sequence terminates, returning `iesBadActionData`. For information on the syntax of

conditional statements, see [Conditional Statement Syntax](#).

## Sequence

A positive value indicates the sequence position of the action. The following negative values indicate that the action is called if the installer returns the termination flag. Each termination flag (negative value) can be used with no more than one action. Multiple actions can have termination flags, but they must be different flags. Termination flags (negative values) are typically used with [Dialog Boxes](#).

Termination flag	Value	Description
msiDoActionStatusSuccess	-1	Successful completion. Used with <a href="#">Exit</a> dialog boxes.
msiDoActionStatusUserExit	-2	User terminates install. Used with <a href="#">UserExit</a> dialog boxes.
msiDoActionStatusFailure	-3	Fatal exit terminates. Used with a <a href="#">FatalError</a> dialog boxes.
msiDoActionStatusSuspend	-4	Install is suspended.

Zero, all other negative numbers, or a null value indicate that the action is never called.

## Validation

[ICE03](#)

[ICE06](#)

[ICE12](#)

[ICE13](#)

[ICE20](#)

[ICE26](#)

[ICE27](#)

[ICE28](#)

[ICE46](#)

[ICE75](#)

ICE79  
ICE82  
ICE84  
ICE86  
ICE96  
ICEM04

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# AdvtExecuteSequence Table

The AdvtExecuteSequence table lists actions the installer calls when the top-level [ADVVERTISE action](#) is executed.

Only the following actions can be used in the AdvtExecuteSequence table. Custom actions cannot be used in this table.

[CostFinalize](#)

[CostInitialize](#)

[CreateShortcuts](#)

[InstallFinalize](#)

[InstallInitialize](#)

[InstallValidate](#)

[MsiPublishAssemblies](#)

[PublishComponents](#)

[PublishFeatures](#)

[PublishProduct](#)

[RegisterClassInfo](#)

[RegisterExtensionInfo](#)

[RegisterMIMEInfo](#)

[RegisterProgIdInfo](#)

The columns are identical to those of the [InstallExecuteSequence table](#). The AdvtExecuteSequence table has the following columns.

Column	Type	Key	Nullable
Action	<a href="#">Identifier</a>	Y	N
Condition	<a href="#">Condition</a>	N	Y
Sequence	<a href="#">Integer</a>	N	Y

## Columns

### Action

Name of the [standard action](#) the installer is to execute. This is the primary key of the table.

### Condition

Logical expression. If the expression evaluates to false, the action is skipped. If the expression syntax is invalid, the sequence terminates, returning `iesBadActionData`. For information on the syntax of conditional statements, see [Conditional Statement Syntax](#).

### Sequence

A positive value indicates the sequence position of the action. The following negative values indicate that the action is called if the installer returns the termination flag. Each termination flag (negative value) can be used with no more than one action. Multiple actions can have termination flags, but they must be different flags. Termination flags (negative values) are typically used with [Dialog Boxes](#).

Termination flag	Value	Description
<code>msiDoActionStatusSuccess</code>	-1	Successful completion. Used with <a href="#">Exit</a> dialog boxes.
<code>msiDoActionStatusUserExit</code>	-2	User terminates install. Used with <a href="#">UserExit</a> dialog boxes.
<code>msiDoActionStatusFailure</code>	-3	Fatal exit terminates. Used with a <a href="#">FatalError</a> dialog boxes.
<code>msiDoActionStatusSuspend</code>	-4	Install is suspended.

Zero, all other negative numbers, or a null value indicate that the action is never called.

## Validation

[ICE03](#)

ICE06  
ICE12  
ICE13  
ICE27  
ICE46  
ICE72  
ICE79  
ICE82  
ICE83  
ICE84  
ICE86  
ICEM04

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# AdvtUISequence Table

The installer does not use this table. The AdvtUISequence table should not exist in the installation database or it should be left empty.

## Validation

[ICE03](#)

[ICE06](#)

[ICE78](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# AppId Table

The AppId table or the [Registry table](#) specifies that the installer configure and register DCOM servers to do one of the following during an installation.

- Run the DCOM server under a different identity than the user activating the server. For example, to configure a DCOM server to always run as an interactive user or as a predefined user.
- Run the DCOM server as a service.
- Configure the default security access for the DCOM server.
- Register the DCOM server such that it is activated on a different computer.

This table is processed at the installation of the component associated with the DCOM server in the `_Component` column of the [Class table](#). An AppId is not advertised.

The AppId table has the following columns.

Column	Type	Key	Nullable
AppId	GUID	Y	N
RemoteServerName	Formatted	N	Y
LocalService	Text	N	Y
ServiceParameters	Text	N	Y
DllSurrogate	Text	N	Y
ActivateAtStorage	Integer	N	Y
RunAsInteractiveUser	Integer	N	Y

## Columns

AppId

The AppId column of the [Class table](#) is a foreign key into this column of the AppId table. This column contains the AppId value that will be written under the CLSID and creates the AppId GUID key under HKCR\AppId.

#### RemoteServerName

This column contains the value of "RemoteServerName"=<xxxx> that will be written under HKCR\AppID\{AppID}\ .

#### LocalService

This column contains the value of LocalService that will be written under HKCR\AppID\{<appid>} "LocalService"=<xxx>.

#### ServiceParameters

This column contains the value of ServiceParameters that will be written under HKCR\AppID\{appid>} "ServiceParameters".

#### DllSurrogate

This column contains the value of DllSurrogate that will be written under HKCR\AppId\{<appid>} "DllSurrogate"=<xxx>. If this column is present it will typically be an empty string.

#### ActivateAtStorage

This column contains the value of ActivateAtStorage that will be written under HKCR\AppID\{<appid>} "ActivateAtStorage"="Y".

#### RunAsInteractiveUser

This column contains the value of RunAsInteractiveUser that will be written under HKCR\AppID\{appid>} "RunAs"="Interactive User".

## Remarks

This table is used by the [RegisterClassInfo action](#) and [UnregisterClassInfo action](#).

Note that the AppId table does not have a column for registering a Default name. Therefore in cases where you need to write a user friendly name as the Default name value, you must register using the [Registry table](#).

## Validation

ICE03  
ICE06  
ICE32  
ICE33  
ICE46  
ICE69

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# AppSearch Table

The AppSearch table contains properties needed to search for a file having a particular file signature. The AppSearch table can also be used to set a property to the existing value of a registry or .ini file entry.

The AppSearch table has the following columns.

Column	Type	Key	Nullable
Property	Identifier	Y	N
Signature_	Identifier	Y	N

## Columns

### Property

Running the [AppSearch](#) action sets this property to the location of the file indicated by the `Signature_` column. This property is set if the file signature exists on the user's computer. The properties used in this column must be [public](#) properties and have an identifier that contains no lowercase letters.

The property listed in the Property field may be initialized in the [Property](#) table or from a command line. If the AppSearch action locates the signature, the installer overrides the initialized property value with the found value. If the signature is not found, then the initial property value is used. If the property was never initialized, then the property will only be set if the signature is found. Otherwise, the property is undefined.

### Signature\_

The `Signature_` column contains a unique identifier called a signature and is also an external key into the [RegLocator](#), [IniLocator](#), [CompLocator](#), and [DrLocator](#) tables. When searching for a file, the value in this column must also be a foreign key into the [Signature](#) table. If the value in this column is not listed in the Signature table, the installer determines that the search is for a directory.

## Remarks

The AppSearch action in [sequence tables](#) processes the information in this table. For information about using sequence tables, see [Using a Sequence Table](#).

The AppSearch action searches for signatures using the CompLocator table first, the RegLocator table second, the IniLocator table third, and finally the DrLocator table. File signatures are listed in the Signature table. A signature that is not in the Signature table denotes a directory and the action sets the property to the directory path for that signature.

See [Searching for Existing Applications, Files, Registry Entries or .ini File Entries](#).

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE32](#)  
[ICE52](#)  
[ICE88](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# BBControl Table

The BBControl table lists the controls to be displayed on each billboard.

The BBControl table has the following columns.

Column	Type	Key	Nullable
Billboard_	Identifier	Y	N
BBControl	Identifier	Y	N
Type	Identifier	N	N
X	Integer	N	N
Y	Integer	N	N
Width	Integer	N	N
Height	Integer	N	N
Attributes	DoubleInteger	N	Y
Text	Text	N	Y

## Columns

### Billboard\_

Name of the billboard.

External key to column one of the [Billboard table](#).

### BBControl

Name of the control. This name must be unique within a billboard but can be repeated on different billboards. This column combined with the Billboard\_ column forms the primary key to the table.

### Type

The type of the control. Only static controls, such as a [Text](#), [Bitmap](#), [Icon](#), or custom control can be placed on a billboard. For a complete list of controls, see the [Controls](#) section.

### X

Horizontal coordinate of the upper-left corner of the rectangular boundary of the control. The units are [installer units](#). This coordinate is measured relative to the billboard control and not relative to the dialog. Use only non-negative numbers.

Y

Vertical coordinate of the upper-left corner of the rectangular boundary of the control. The units are [installer units](#). This coordinate is measured relative to the billboard control and not relative to the dialog. This number must be non-negative.

Width

Width of the rectangular boundary of the control. The units are [installer units](#). This number must be non-negative.

Height

Height of the rectangular boundary of the control. The units are [installer units](#). This number must be non-negative.

Attributes

A 32-bit word specifying the attribute flags to be applied to this control. This number must be non-negative and specify an attribute for a static control that is valid for placement on a billboard. For information on the numeric values to enter into this field, see the particular attribute under [Control Attributes](#).

Text

This column contains a localizable string used to set the initial text in the control if the control displays text. The string is truncated if the text is too long to fit on the control. This column contains a key into the [Binary table](#) if the control is a push button or a check box containing an icon or bitmap. It is not possible to show both text and a picture on the same button. This column may be left blank.

## Remarks

The integer values for x, y, width, and height are in the [installer units](#), not dialog units. An installer unit is equal to one-twelfth the height of the 10-point MS Sans Serif font size. Coordinates for the controls are relative to the billboard control not the dialog.

## Validation

[ICE03](#)

[ICE06](#)

[ICE32](#)

[ICE45](#)

[ICE95](#)

## See Also

[MsiSetExternalUI](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Billboard Table

The Billboard table lists the [Billboard controls](#) displayed in the full user interface.

The Billboard table has the following columns.

Column	Type	Key	Nullable
Billboard	<a href="#">Identifier</a>	Y	N
Feature_	<a href="#">Identifier</a>	N	N
Action	<a href="#">Identifier</a>	N	Y
Ordering	<a href="#">Integer</a>	N	Y

## Columns

### Billboard

Name of the Billboard control.

### Feature\_

An external key to the first column of the [Feature table](#). The billboard is displayed only if this feature is being installed.

### Action

The name of an action. The billboard is displayed during the progress messages received from this action.

### Ordering

If there is more than one billboard corresponding to an action, then they are displayed in the order defined by this column. This number must be non-negative.

## Validation

[ICE03](#)

[ICE06](#)

ICE32  
ICE95

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Binary Table

The Binary table holds the binary data for items such as bitmaps, animations, and icons. The binary table is also used to store data for custom actions. See [OLE Limitations on Streams](#).

The Binary table has the following columns.

Column	Type	Key	Nullable
Name	<a href="#">Identifier</a>	Y	N
Data	<a href="#">Binary</a>	N	N

## Columns

### Name

A unique key that identifies the particular binary data. If the binary data is for a control, the key appears in the Text column of the associated control in the [Control table](#). This key must be unique among all controls requiring binary data.

### Data

The unformatted binary data.

## Validation

[ICE03](#)

[ICE06](#)

[ICE17](#)

[ICE29](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# BindImage Table

The BindImage table contains information about each executable or DLL that needs to be bound to the DLLs imported by it.

The BindImage table has the following columns.

Column	Type	Key	Nullable
File_	Identifier	Y	N
Path	Paths	N	Y

## Columns

File\_

An external key to column one of the [File table](#). This must be an executable file or a DLL file.

Path

A list of paths, separated by semicolons, that represent the paths to be searched to find the imported DLLs. The list is usually a list of properties, with each property enclosed inside square brackets ([ ]).

## Remarks

The installer computes the virtual address of each function that is imported from all DLLs, and the computed virtual address is then saved in the importing image's Import Address Table (IAT).

This table is referred to when the [BindImage action](#) is executed.

## Validation

[ICE03](#)

[ICE06](#)

[ICE32](#)

[ICE46](#)

ICE76

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# CCPSearch Table

The CCPSearch table contains the list of file signatures used for the Compliance Checking Program (CCP). At least one of these files needs to be present on a user's computer for the user to be in compliance with the program.

The CCPSearch table has the following column.

Column	Type	Key	Nullable
Signature_	Identifier	Y	N

## Column

### Signature\_

The Signature\_ represents a unique file signature and is also the external key into the [Signature](#), [RegLocator](#), [IniLocator](#), [CompLocator](#), and [DrLocator](#) tables.

## Remarks

This table is referred to when the [CCPSearch action](#) or the [RMCCPSearch action](#) is executed.

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE32](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# CheckBox Table

The CheckBox table lists the values for the check boxes.

The CheckBox table has the following columns.

Column	Type	Key	Nullable
Property	Identifier	Y	N
Value	Formatted	N	Y

## Columns

### Property

A named property to be tied to this item.

### Value

The value string associated with this item.

## Remarks

If the check box is selected, then the corresponding property is set to the specified value. If there is no value specified or this table does not exist, then the property is set to its original value when the check box is selected. If the original value is null, the property is set to "1".

The contents of the Value column are formatted by the **MsiFormatRecord** function when the control is created. Therefore, it can contain any expression that the **MsiFormatRecord** function can interpret. The Value column is formatted only when the control is created, and it is not updated if a property involved in an expression is modified during the life of the control.

## Validation

ICE03

ICE06

ICE46

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Class Table

The Class table contains COM server-related information that must be generated as a part of the product advertisement. Each row may generate a set of registry keys and values. The associated ProgId information is included in this table.

The Class table has the following columns.

Column	Type	Key	Nullable
CLSID	GUID	Y	N
Context	Identifier	Y	N
Component_	Identifier	Y	N
ProgId_Default	Text	N	Y
Description	Text	N	Y
AppId_	GUID	N	Y
FileTypeMask	Text	N	Y
Icon_	Identifier	N	Y
IconIndex	Integer	N	Y
DefInprocHandler	Filename	N	Y
Argument	Formatted	N	Y
Feature_	Identifier	N	N
Attributes	Integer	N	Y

## Column Information

### CLSID

The Class identifier (ID) of a COM server.

### Context

The server context for this server. Enter one of the following values

for the CLSID Key.

CLSID KEY	Description
LocalServer	Specifies the full path to a 16-bit local server application.
LocalServer32	Specifies the full path to a 32-bit local server application.
InprocServer	Specifies the path to an in-process server DLL.
InprocServer32	Specifies the path to a 32-bit in-process server and the threading model.

#### Component\_

External key into the [Component table](#) specifying the component whose key file provides the COM server.

#### ProgId\_Default

The default Program ID associated with this Class ID. This column is a foreign key into the [ProgID table](#).

#### Description

Localized description associated with the Class ID and Program ID.

#### AppId\_

Application ID containing DCOM information for the associated application (string [GUID](#)). This column is a foreign key into the [AppId table](#).

#### FileTypeMask

Contains information for the HKCR (this CLSID) key.

If multiple patterns exist, they must be delimited by a semicolon, and numeric subkeys are generated: 0, 1, 2... For more information about this functionality, see **GetClassFile**.

#### Icon\_

The file providing the icon associated with this CLSID. The installer writes the entry in this column under the DefaultIcon key associated with the ProgId. If it is not null, the column is a foreign key into the [Icon table](#). If it is null, the COM server provides the icon resource. Advertised file associations and shortcuts require a non-null value in

this column to display properly.

#### IconIndex

Icon index into the icon file. This can be null.

Non-negative numbers only.

#### DefInprocHandler

This field specifies the default in-process handler for the server context specified in the Context field.

This field must be Null if an InprocServer or InprocServer CLSID key appears in the Context field.

If a LocalServer or LocalServer32 CLSID key appears in the Context field, the value in the DefInprocHandler field identifies the default in-process handler.

<b>Value</b>	<b>Description</b>
non-numeric value	The installer treats a non-numeric value in the DefInprocHandler field as a system file serving as the 32-bit in-process handler specified by the InprocHandler32 key.
Null	The DefInprocHandler and Argument fields can both be Null for a LocalServer or LocalServer32 CLSID key.
1 = default (system)	The default is the 16-bit in-process handler specified by InprocHandler. In this case, the value of InprocHandler is the name in the registry under which the value of the default in-process handler is stored. For example, HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID.
2 = default (system)	The default is the 32-bit in-process handler specified by InprocHandler32. In this case, the value of InprocHandler32 is the name in the registry under which the value of the default in-process handler is stored. For example, HKEY_LOCAL_MACHINE\SOFTWARE\Classes\CLSID.
3 = default (system)	The default is a 16-bit or 32-bit in-process handler.

## Argument

If a LocalServer or LocalServer32 CLSID key appears in the Context field, the text in this field is registered as the argument against the server and is used by COM to invoke the server. The DefInprocHandler and Argument fields can both be Null if LocalServer or LocalServer32 appears in the Context field.

Note that the resolution of properties in the Argument field is limited. A property formatted as [Property] in this field can only be resolved if the property already has the intended value when the component owning the class is installed. For example, for the argument "[#MyDoc.doc]" to resolve to the correct value, the same process must be installing the file MyDoc.doc and the component that owns the class.

## Feature\_

External key into the [Feature table](#) specifying the feature that provides the COM server.

External key to column one of the Feature table.

## Attributes

If msidbClassAttributesRelativePath is set in this column, the bare file name can be used for COM servers. The installer registers the file name only instead of the complete path. This enables the server in the current directory to take precedence and allows multiple copies of the same component.

Attribute	Decimal	Hexadecimal
msidbClassAttributesRelativePath	1	0x001

## Remarks

This table is referred to when the [RegisterClassInfo action](#) or the [UnregisterClassInfo action](#) are executed.

## Validation

ICE03  
ICE06  
ICE19  
ICE32  
ICE36  
ICE41  
ICE42  
ICE46  
ICE66  
ICE69

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ComboBox Table

The lines of a combo box are not treated as individual controls; they are part of a single combo box that functions as a control. This table lists the values for each combo box.

The ComboBox table has the following columns.

Column	Type	Key	Nullable
Property	Identifier	Y	N
Order	Integer	Y	N
Value	Formatted	N	N
Text	Text	N	Y

## Columns

### Property

A named property to be tied to this item. All the items tied to the same property become part of the same combo box.

### Order

A positive integer used to determine the ordering of the items that appear in a single combo box list. The integers do not have to be consecutive. If a combo box is defined as ordered, then all the items should have an Order value. If the combo box is defined as unordered, then this column is ignored.

Positive numbers only.

### Value

The value string associated with this item. Selecting this line of the combo box sets the associated property (specified in Property) to this value.

### Text

The visible, localizable text to be assigned to the item. If this entry or the entire column is missing, then the text defaults to the entry in

Value.

## Remarks

The contents of the Value and Text fields are formatted by the **MsiFormatRecord** function when the control is created, therefore they can contain any expression that the **MsiFormatRecord** function can interpret. The formatting occurs only when the control is created and it is not updated if a property involved in the expression is modified during the life of the control.

## Validation

[ICE03](#)

[ICE06](#)

[ICE17](#)

[ICE46](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# CompLocator Table

The CompLocator Table contains the information needed to find a file or directory that is using the installer configuration data.

The CompLocator table contains the following information.

Column	Type	Key	Nullable
Signature_	Identifier	Y	N
ComponentId	GUID	N	N
Type	Integer	N	Y

## Column Information

### Signature\_

This column represents a unique file signature and is also the external key into the [Signature Table](#). If the key is absent from the Signature Table, the search is assumed to be for the presence of a directory pointed to by the CompLocator Table.

### ComponentId

The component ID of the component whose key path is to be used for the search. This should be the GUID of a component that appears in the ComponentId field of the [Component Table](#). It may be the component ID of a component belonging to another product installed on the computer. It should not be the GUID of a published component appearing in the ComponentId field of the [PublishComponent Table](#).

To find the component ID GUID value for a file installed by another product, go to the installation package of the product. Go to the [File Table](#) and find the row that contains the file identifier for the file. The Component\_ column of this row contains the component identifier for the component that controls the file. Go to the Component table and find the row that contains this component identifier in the Component column. The ComponentId column of this row contains the component ID GUID.

## Type

A Boolean value that determines if the key path of the component is a file name or a directory location.

The following table lists valid values. If absent, Type is set to be 1 (one).

Constant	Hexadecimal	Decimal	Description
msidbLocatorTypeDirectory	0x000	0	Key path is a directory.
msidbLocatorTypeFileName	0x001	1	Key path is a file name.

## Remarks

This table is used with the [AppSearch Table](#).

Typically, the columns in this table are not localized. If an author decides to search for products in multiple languages, then there can be a separate entry included in the table for each language.

For more information, see [Searching for Existing Applications, Files, Registry Entries or .ini File Entries](#).

## Validation

[ICE03](#)

[ICE06](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Complus Table

The Complus table contains information needed to install COM+ applications.

The Complus table has the following columns.

Column	Type	Key	Nullable
Component_	Identifier	Y	N
ExpType	Integer	N	Y

## Columns

### Component\_

An external key into the first column of the [Component table](#). This is the component that contains the COM+ application.

### ExpType

Export flags used during the generation of the .msi file. For more information see the COM+ documentation in the Microsoft Windows Software Development Kit (SDK).

## Remarks

See the [RegisterComPlus action](#) and [UnregisterComPlus action](#).

See [Installing a COM+ Application with the Windows Installer](#).

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE32](#)  
[ICE66](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Component Table

The Component table lists components and it has the following columns.

Column	Type	Key	Nullable
Component	Identifier	Y	N
ComponentId	GUID	N	Y
Directory_	Identifier	N	N
Attributes	Integer	N	N
Condition	Condition	N	Y
KeyPath	Identifier	N	Y

## Columns

### Component

Identifies the component record.

Primary table key.

### ComponentId

A string GUID unique to this component, version, and language.

Note that the letters of these GUIDs must be uppercase. Utilities such as GUIDGEN can generate GUIDs containing lowercase letters. The lowercase letters must be changed to uppercase to make these valid component code GUIDs.

If this column is null the installer does not register the component and the component cannot be removed or repaired by the installer. This might be intentionally done if the component is only needed during the installation, such as a custom action that cleans up temporary files or removes an old product. It may also be useful when copying data files to a user's computer that do not need to be registered.

### Directory\_

External key of an entry in the [Directory table](#). This is a property

name whose value contains the actual path, which can be set either by the [AppSearch action](#) or with the default setting obtained from the Directory table.

Developers must avoid authoring components that place files into one of the User Profile folders. These files would not be available to all users in multi-user situations and could cause the installer to permanently view the component as requiring repair.

External key to column one of the Directory table.

### Attributes

This column contains a bit flag that specifies options for remote execution. Add the indicated bit to the total value in the column to include an option.

**Note** In the case of an .msi file that is being downloaded from a web location, the attribute flags should not be set to allow a component to be run-from-source. This is a limitation of the Windows Installer and can return a feature state of `INSTALLSTATE_BADCONFIG`.

Bit flag name	Decimal	Hexadec
msidbComponentAttributesLocalOnly	0	0x0000
msidbComponentAttributesSourceOnly	1	0x0001
msidbComponentAttributesOptional	2	0x0002
msidbComponentAttributesRegistryKeyPath	4	0x0004

msidbComponentAttributesSharedDllRefCount	8	0x0008
msidbComponentAttributesPermanent	16	0x0010
msidbComponentAttributesODBCDataSource	32	0x0020
msidbComponentAttributesTransitive	64	0x0040
msidbComponentAttributesNeverOverwrite	128	0x0080

msidbComponentAttributes64bit	256	0x0100
msidbComponentAttributesDisableRegistryReflection	512	0x0200
msidbComponentAttributesUninstallOnSupersedence	1024	0x0400

msidbComponentAttributesShared	2048	0x0800

## Condition

This column contains a conditional statement that can control whether a component is installed. If the condition is null or evaluates to true, then the component is enabled. If the condition evaluates to False, then the component is disabled and is not installed.

The Condition field enables or disables a component only during the [CostFinalize action](#). To enable or disable a component after CostFinalize, you must use a custom action or the [DoAction ControlEvent](#) to call [MsiSetComponentState](#).

Note that unless the Transitive bit in the Attributes column is set for a component, the component remains enabled once installed even if the conditional statement in the Condition column later evaluates to False on a subsequent maintenance installation of the product.

The Condition column in the Component table accepts conditional

expressions containing references to the installed states of features and components. For information on the syntax of conditional statements, see [Conditional Statement Syntax](#).

## KeyPath

This value points to a file or folder belonging to the component that the installer uses to detect the component. Two components cannot share the same key path value. The value in this column is also the path returned by the [MsiGetComponentPath](#) function.

If the value is not null, then KeyPath is either a primary key into the [Registry](#), [ODBCDataSource](#), or [File tables](#) depending upon the Attribute value. If KeyPath is null, then the folder of the Directory\_ column is used as the key path.

Because folders created by the installer are deleted when they become empty, you must author an entry into the [CreateFolder table](#) to install a component that consists of an empty folder.

Note that if a Windows Installer component contains a file or registry key that is protected by Windows Resource Protection (WRP) or a file that is protected by Windows File Protection (WFP), this resource must be used as the KeyPath for the component. In this case, Windows Installer does not install, update, or remove the component. You should not include any protected resources in an installation package. Instead, you should use the supported resource replacement mechanisms for Windows Resource Protection. For more information, see [Using Windows Installer and Windows Resource Protection](#).

## Remarks

For a discussion of the relationship between components and features, see [Feature Table](#).

The installer keeps track of shared DLLs independently of the shared DLL reference count in the registry. If a reference count for a shared DLL exists in the registry, the installer always increments the count when it is installing the file and decrements it when it is uninstalling. If `msidbComponentAttributesSharedDllRefCount`, is not set, and the

reference count does not already exist, the installer will not create one. Note that the SharedDLLs reference count in the registry is incremented for any files installed to the System folder (System32 on Windows 2000).

If `msidbComponentAttributesSharedDllRefCount` is not set, then another application can remove the component even if it is still needed. To see how this could happen consider the following scenario:

- An application that uses the installer installs a shared component.
- The `msidbComponentAttributesSharedDllRefCount` bit is not set and there is no reference count. Thus the installer does not begin a reference count.
- A legacy application that shares this component and does not use the installer is installed.
- The legacy application creates and increments a reference count for the shared component.
- The legacy application is uninstalled.
- The reference count for the shared component is decremented to zero and the component is removed.
- The application using the installer no longer has access to the component.

To avoid this behavior, set `msidbComponentAttributesSharedDllRefCount`.

Note that system services components should not be specified as run-from-source without being specifically designed for such use. See the [ServiceInstall table](#) for more details.

Note that attributes enabling run-from-source installation should never be set for components containing dynamic-link libraries that are going into the system folder. The reason is that if the installation state of the component becomes set to run-from-source by following a feature or by being set in the UI, subsequent calls to **LoadLibrary** on the DLL would fail.

See also, [Controlling Feature Selection States](#).

## Validation

ICE02  
ICE03  
ICE06  
ICE07  
ICE08  
ICE09  
ICE18  
ICE19  
ICE21  
ICE30  
ICE32  
ICE35  
ICE38  
ICE41  
ICE42  
ICE43  
ICE46  
ICE50  
ICE54  
ICE57  
ICE59  
ICE62  
ICE67  
ICE76  
ICE79  
ICE80  
ICE83  
ICE86  
ICE88  
ICE91  
ICE92  
ICE97

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Condition Table

The Condition table can be used to modify the selection state of any entry in the [Feature table](#) based on a conditional expression.

The Condition table has the following columns.

Column	Type	Key	Nullable
Feature_	<a href="#">Identifier</a>	Y	N
Level	<a href="#">Integer</a>	Y	N
Condition	<a href="#">Condition</a>	N	Y

## Columns

### Feature\_

External key into column one of the Feature table.

### Level

A conditional install level for the feature in the Feature\_ column of this table. The installer sets the install level of this feature to the level specified in this column if the expression in the Condition column evaluates to TRUE.

### Condition

If this conditional expression evaluates to TRUE, then the Level column in the Feature table is set to the conditional install level.

The expression in the Condition column should not contain reference to the installed state of any feature or component. This is because the expressions in the Condition column are evaluated before the installer evaluates the installed states of features and components. Any expression in the Condition table that attempts to check the installed state of a feature or component always evaluates to false.

For information on the syntax of conditional statements, see [Conditional Statement Syntax](#).

## Remarks

A feature can be permanently disabled by setting the Level column to 0.

The Level may be set based on any conditional statement, such as a test for platform, operating system, or a particular property setting.

Conditions should be carefully chosen so that a feature is not enabled on install and then disabled on uninstall. This will orphan the feature and the product will not be able to be uninstalled.

This table is referred to when the [CostFinalize action](#) is executed.

If the [Preselected](#) property has been set to 1, the installer does not evaluate the Condition table. The Condition table affects only the installation of features when none of the following properties have been set:

[ADDLOCAL](#)  
[REMOVE](#)  
[ADDSOURCE](#)  
[ADDDEFAULT](#)  
[REINSTALL](#)  
[ADVERTISE](#)  
[COMPADDLOCAL](#)  
[COMPADDSOURCE](#)  
[COMPADDDEFAULT](#)  
[FILEADDLOCAL](#)  
[FILEADDSOURCE](#)  
[FILEADDDEFAULT](#)

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE32](#)  
[ICE46](#)  
[ICE79](#)  
[ICE86](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Control Table

The Control table defines the controls that appear on each dialog box.

The Control table has the following columns.

Column	Type	Key	Nullable
Dialog_	Identifier	Y	N
Control	Identifier	Y	N
Type	Identifier	N	N
X	Integer	N	N
Y	Integer	N	N
Width	Integer	N	N
Height	Integer	N	N
Attributes	DoubleInteger	N	Y
Property	Identifier	N	Y
Text	Formatted	N	Y
Control_Next	Identifier	N	Y
Help	Text	N	Y

## Columns

### Dialog\_

External key to the first column of the [Dialog table](#), the name of the dialog box.

### Control

Name of the control. This name must be unique within a dialog box but can be repeated on different dialog boxes. The Control column combined with the Dialog\_ column form the primary key to this table.

### Type

The type of the control. For a list of control types, see [Controls](#).

## X

Horizontal coordinate of the upper-left corner of the rectangular boundary of the control. This must be a non-negative number. See [Position Control Attribute](#).

## Y

Vertical coordinate of the upper-left corner of the rectangular boundary of the control. This must be a non-negative number. See [Position Control Attribute](#).

## Width

Width of the rectangular boundary of the control. This must be a non-negative number. See [Position Control Attribute](#).

## Height

Height of the rectangular boundary of the control. This must be a non-negative number. See [Position Control Attribute](#).

## Attributes

A 32-bit word that specifies the bit flags to be applied to this control. This must be a non-negative number, and the allowed values depend upon the type of control. For a list of all control attributes, and the value to enter in this field, see [Control Attributes](#).

## Property

The name of a defined property to be linked to this control. Radio button, list box, and combo box values are tied into a group by being linked to the same property. This column is required for active controls.

## Text

A localizable string used to set the initial text contained in a control. The string can also contain embedded properties. For the syntax of a formatted string containing properties see the [MsiFormatRecord](#) function. Specify the size, font, and color of the text by prefixing the text string with {style}, where style is a text style authored into the TextStyle column of the [TextStyle table](#). The text string is truncated if it is too long to fit on to the control. The text string may be blank.

Special authoring of the [Formatted](#) text string in this field is required if the text is to be displayed by a [Text Control](#) located on a dialog box having the TrackDiskpace attribute. This is the case specified by the

[TrackDiskSpace Dialog Style Bit](#) appearing in the Attributes of the [Dialog table](#). In this case, if the Formatted string in the Text column of the Control table begins with "[" and ends with "]" then you must add a space at the end of the string. For example, if DlgTextFont is a property that will be set to "{DlgFontBold}" the formatted string "[DlgTextFont]MyText[ProductName]" requires the space at the end after the closing bracket. This extra space is required by the installer to correctly display the text in the Text control.

You can enter a short descriptive text string for the [VolumeCostList](#), [ListView](#), [DirectoryList](#), and the [SelectionTree controls](#). This text is not seen by the user but it can be read by screen readers as the description of the control.

See also [Accessibility](#).

#### Control\_Next

The name of another control on the same dialog box and an external key to the second column of the Control table. If the focus in the dialog box is on the control in the Control column, hitting the tab key moves the focus to the control listed in the Control\_Next column. Therefore this column is used to specify the tab order of the controls on the dialog box. The links between the controls must form a closed cycle. Some controls, such as static text controls, can be left out of the cycle. In this case, this field may be left blank.

See also [Accessibility](#).

#### Help

Optional, localizable text strings that are used with the Help button. The string is divided into two parts by a separator character (|). The first part of the string is used as ToolTip text. This text is used by screen readers for controls that contain a picture. The second part of the string is reserved for future use. The separator character is required even if only one of the two kinds of text is present.

## Remarks

The integer values for x, y, width, and height are in the [installer units](#), not dialog units. An installer unit is equal to one-twelfth the height of the 10-

point MS Sans Serif font size. Coordinates for the controls are relative to the billboard.

## **Validation**

ICE03  
ICE06  
ICE17  
ICE20  
ICE23  
ICE31  
ICE32  
ICE34  
ICE45  
ICE46  
ICE95

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ControlCondition Table

The ControlCondition table enables an author to specify special actions to be applied to controls based on the result of a conditional statement. For example, using this table the author could choose to hide a control based on the [VersionNT](#) property.

The ControlCondition table has the following columns.

Column	Type	Key	Nullable
Dialog_	Identifier	Y	N
Control_	Identifier	Y	N
Action	Text	Y	N
Condition	Condition	Y	N

## Columns

### Dialog\_

An external key to the first column of the [Dialog table](#). Combining this field with the Control\_ field identifies a unique control.

### Control\_

An external key to the second column of the [Control table](#). Combining this field the Dialog\_ field identifies a unique control.

### Action

The action that is to be taken on the control. The possible actions are shown in the following table.

Value	Meaning
Default	Set control as the default.
Disable	Disable the control.
Enable	Enable the control.
Hide	Hide the control.

Show	Display the control.
------	----------------------

## Condition

A conditional statement that specifies under which conditions the action should be triggered. This column may not be left blank. If this statement does not evaluate to TRUE, the action does not take place. If it is set to 1, the action is always applied. For information on the syntax of conditional statements, see [Conditional Statement Syntax](#).

## Remarks

If you want to hide and disable a [PushButton control](#) or [CheckBox control](#) based on a conditional statement in the Condition field of the ControlCondition table, you should use four records for each control to disable as well as hide the control. PushButton or CheckBox controls that have only been hidden can still be accessed by shortcut keys.

For example, the following records hide and disable ControlA on DialogA when the product is installed. The control will be visible and enabled when the product is not installed.

<b>Dialog</b>	<b>Control</b>	<b>Action</b>	<b>Condition</b>
DialogA	ControlA	Hide	<b>Installed</b>
DialogA	ControlA	Disable	Installed
DialogA	ControlA	Show	NOT Installed
DialogA	ControlA	Enable	NOT Installed

## Validation

[ICE03](#)

[ICE06](#)

[ICE17](#)

[ICE32](#)

ICE46  
ICE79  
ICE86

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ControlEvent Table

The ControlEvent table allows the author to specify the [Control Events](#) started when a user interacts with a [PushButton Control](#), [CheckBox Control](#), or [SelectionTree Control](#). These are the only controls users can use to initiate control events. Each control can publish multiple control events. The installer starts each event in the order specified in the Ordering column. For example, a push button control can publish events to initiate a transition to another dialog box, exit the dialog box sequence, and begin file installation. The exception is that each control can publish a most one [NewDialog](#) or one [SpawnDialog](#) event. If multiple NewDialog and SpawnDialog control events are selected for the same control, only the event with the largest value in the Ordering column gets published when the control is activated.

The ControlEvent table has the following columns.

Column	Type	Key	Nullable
Dialog_	Identifier	Y	N
Control_	Identifier	Y	N
Event	Formatted	Y	N
Argument	Formatted	Y	N
Condition	Condition	Y	Y
Ordering	Integer	N	Y

## Columns

### Dialog\_

An external key to the first column of the [Dialog table](#). Combining this field with the Control\_ field identifies a unique control.

### Control\_

An external key to the second column of the [Control table](#).

Combining this field with the Dialog\_ field identifies a unique control.

## Event

An identifier that specifies the type of event that should take place when the user interacts with the control specified by `Dialog_` and `Control_`. For a list of possible values see [ControlEvent Overview](#).

To set a property with a control, put `[Property_Name]` in this field and the new value in the argument field. Put `{ }` into the argument field to enter the null value.

## Argument

A value used as a modifier when triggering a particular event.

For example, the argument of the `NewDialog ControlEvent` or the `SpawnDialog ControlEvent` is the name of the dialog box and the argument of the [Install action](#) is a number defining the install level.

## Condition

A conditional statement that determines whether the installer activates the event in the Event column. The installer triggers the event if the conditional statement in the Condition field evaluates to True. Therefore put a 1 in this column to ensure that the installer triggers the event. The installer does not trigger the event if the Condition field contains a statement that evaluates to False. The installer does not trigger an event with a blank in the Condition field unless no other events of the control evaluate to True. If none of the Condition fields for the control named in the `Control_` field evaluate to True, the installer triggers the one event having a blank Condition field, and if more than one Condition field is blank it triggers the one event of these with the largest value in the Ordering field. See [Conditional Statement Syntax](#).

## Ordering

An integer used to order several events tied to the same control. This must be a non-negative number. This field may be left blank.

## Remarks

The [EventMapping table](#) lists the controls that subscribe to some control event and lists the control attribute to be changed when that event is published by the another control or the installer.

On Windows XP or earlier operating systems, users can publish a control event only by interacting with a [Checkbox Control](#) or [Pushbutton Control](#). With Windows Server 2003, users can publish a control event only by interacting with a [Checkbox Control](#), [SelectionTree Control](#), and [Pushbutton Control](#). Listing other controls in the Control\_ field has no effect.

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE17](#)  
[ICE20](#)  
[ICE32](#)  
[ICE44](#)  
[ICE46](#)  
[ICE79](#)  
[ICE86](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# CreateFolder Table

The CreateFolder table contains references to folders that need to be created explicitly for a particular component.

The CreateFolder table has the following columns.

Column	Type	Key	Nullable
Directory_	<a href="#">Identifier</a>	Y	N
Component_	<a href="#">Identifier</a>	Y	N

## Columns

Directory\_

External key into the first column of the [Directory table](#).

Component\_

External key into the first column of the [Component table](#).

## Remarks

The folders in this table are created when the component is installed. An attempt is made to remove these folders only when the component is uninstalled or moved to run-from-source. No automatic removal is triggered if the folders become empty. In contrast, folders created by the installer but not listed in this table are removed when they become empty.

Because folders created by the installer are deleted when they become empty, you must author an entry into the CreateFolder table to install a component that consists of an empty folder.

This table is referred to when the [CreateFolders](#) action or the [RemoveFolders](#) action is called.

For information on how to secure a folder see the [MsiLockPermissionsEx Table](#) and [LockPermissions Table](#).

## Validation

[ICE03](#)

[ICE06](#)

[ICE18](#)

[ICE32](#)

[ICE55](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# CustomAction Table

The CustomAction table provides the means of integrating custom code and data into the installation. The source of the code that is executed can be a stream contained within the database, a recently installed file, or an existing executable file.

The CustomAction table has the following columns.

Column	Type	Key	Nullable
Action	<a href="#">Identifier</a>	Y	N
Type	<a href="#">Integer</a>	N	N
Source	<a href="#">CustomSource</a>	N	Y
Target	<a href="#">Formatted</a>	N	Y
ExtendedType	<a href="#">DoubleInteger</a>	N	Y

## Columns

### Action

Name of the action. The action normally appears in a sequence table unless it is called by another custom action. If the name matches any built-in action, then the custom action is never called.

Primary table key.

### Type

A field of flag bits specifying the basic type of custom action and options. See [Summary List of All Custom Action Types](#) for a list of the basic types. See [Custom Action Return Processing Options](#), [Custom Action Execution Scheduling Options](#), [Custom Action Hidden Target Option](#), and [Custom Action In-Script Execution Options](#).

### Source

A property name or external key into another table. For a discussion of the possible custom action sources, see [Custom Action Sources](#)

and the [Summary List of All Custom Action Types](#). For example, the Source column may contain an external key into the first column of one of the following tables containing the source of the custom action code.

[Directory table](#) for calling existing executables.

[File table](#) for calling executables and DLLs that have just been installed.

[Binary table](#) for calling executables, DLLs, and data stored in the database.

[Property table](#) for calling executables whose paths are held by a property.

### Target

An execution parameter that depends on the basic type of custom action. See the [Summary List of All Custom Action Types](#) for a description of what should be entered in this field for each type of custom action. For example, this field may contain the following depending on the custom action.

Target	Custom action
Entry point (required)	Calling a DLL.
Executable name with arguments (required)	Calling an existing executable.
Command line arguments (optional)	Calling an executable just installed.
Target file name (required)	Creating a file from custom data.
Null	Executing script code.

### ExtendedType

Enter the msidbCustomActionTypePatchUninstall value in this field to specify a custom action with the [Custom Action Patch Uninstall Option](#).

**Windows Installer 4.0 and earlier:** Not supported. This option

is available beginning with Windows Installer 4.5.

For more information, see all the topics under [Custom Actions](#).

## **Validation**

[ICE03](#)

[ICE06](#)

[ICE12](#)

[ICE27](#)

[ICE46](#)

[ICE63](#)

[ICE68](#)

[ICE72](#)

[ICE75](#)

[ICE77](#)

[ICE80](#)

[ICE88](#)

[ICE93](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Dialog Table

The Dialog Table contains all the dialogs that appear in the user interface (UI) in both the full and reduced modes.

The Dialog Table has the following columns.

Column	Type	Key	Nullable
Dialog	Identifier	Y	N
HCentering	Integer	N	N
VCentering	Integer	N	N
Width	Integer	N	N
Height	Integer	N	N
Attributes	DoubleInteger	N	Y
Title	Formatted	N	Y
Control_First	Identifier	N	N
Control_Default	Identifier	N	Y
Control_Cancel	Identifier	N	Y

## Columns

### Dialog

The primary key and name of the dialog box.

### HCentering

The horizontal position of the dialog box.

The range is 0 to 100, with 0 at the left edge of the screen and 100 at the right edge.

### VCentering

The vertical position of the dialog box.

The range is 0 to 100, with 0 at the top edge of the screen and 100 at the bottom edge.

## Width

The width of the rectangular boundary of the dialog box.

This number must be non-negative.

## Height

The height of the rectangular boundary of the dialog box.

This number must be non-negative.

## Attributes

A 32-bit word that specifies the attribute flags to be applied to this dialog box.

This number must be non-negative. For more information, see [Dialog Style Bits](#).

## Title

A localizable text string specifying the title to be displayed in the title bar of the dialog box.

## Control\_First

An external key to the second column of the [Control Table](#).

Combining this field with the Dialog field specifies a unique control in the [Control Table](#) that takes the focus when the dialog box is opened. Typically, this can be an [Edit Control](#), [SelectionTree Control](#), or any other control that can take the focus. If the [PushButton Control](#) is the only control present on the dialog box that can take the focus, the PushButton entered in the ControlDefault field must also be entered into the Control First field. This column is ignored in an [Error Dialog](#) box.

Because static text cannot take the focus, a [Text Control](#) that describes an [Edit Control](#), [PathEdit Control](#), [ListView Control](#), [ComboBox Control](#) or [VolumeSelectCombo Control](#) must be made the first control in the dialog box to ensure compatibility with screen readers.

## Control\_Default

An external key to the second column of the Control Table.

Combining this field with the Dialog field specifies the default control that takes focus when the dialog box is opened. Typically, this can be a PushButton Control. If no PushButton Control on the dialog box

has the focus, the Return key is equivalent to clicking on the default control. If this column is left blank, then there is no default control. This column is ignored in an Error Dialog box.

### Control\_Cancel

An external key to the second column of the Control Table.

Combining this field with the Dialog field specifies a control that cancels the installation. This control is coupled to events in the [ControlEvent Table](#) used to cancel the installation. Hitting the ESC key or clicking the Close button is equivalent to clicking on the cancel control. This column is ignored in an Error Dialog box.

The cancel control is hidden during rollback or the removal of backed up files. The internal UI handler hides the control upon receiving a `INSTALLMESSAGE_COMMONDATA` message.

## Remarks

The integer values for width and height are in the [Installer Units](#), not dialog units.

The two centering values are ignored for subsequent dialog boxes in a wizard sequence. Dialog box positions are set by the user or as for the previous dialog box. These dialog box sequences are created by a [NewDialog ControlEvent](#).

## Validation

[ICE03](#)

[ICE06](#)

[ICE13](#)

[ICE20](#)

[ICE23](#)

[ICE27](#)

[ICE32](#)

[ICE44](#)

[ICE45](#)

ICE46

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Directory Table

The Directory table specifies the directory layout for the product. Each row of the table indicates a directory both at the source and the target.

The Directory table has the following columns.

Column	Type	Key	Nullable
Directory	Identifier	Y	N
Directory_Parent	Identifier	N	Y
DefaultDir	DefaultDir	N	N

## Columns

### Directory

The Directory column contains a unique identifier for a directory or directory path. This column can contain the name of a property that is set to the full path of a target directory. If this column contains a property, the target directory takes the name specified in the DefaultDir column and takes the parent directory specified in the Directory\_Parent column.

If the Directory\_Parent column is either null or equal to the value of the Directory column, the Directory column represents a root target directory. Only one root directory may be specified in the Directory table.

### Directory\_Parent

This column is a reference to the directory's parent directory. A record that has a Directory\_Parent column equal to null or equal to the Directory column represents a root directory. The full path of the parent directory is resolved by reference in the Directory\_Parent column is an external key into the Directory column. For example, if a folder has a parent directory named PDIR, the parent directory of PDIR is given in the Directory\_Parent column of the row with PDIR in the Directory column.

## DefaultDir

The DefaultDir column contains the directory's name (localizable) under the parent directory. By default, this is the name of both the target and source directories. To specify different source and target directory names, separate the target and source names with a colon as follows: [targetname]:[sourcename].

If the value of the Directory\_Parent column is null or is equal to the Directory column, the DefaultDir column specifies the name of a root source directory.

For a non-root source directory, a period (.) entered in the DefaultDir column for the source directory name or the target directory name indicates the directory should be located in its parent directory without a subdirectory.

The directory names in this column may be formatted as short filename | long filename pairs.

## Remarks

Each record in the table represents a directory in both the source and the destination images. The Directory table must specify a single root directory with a Directory column value equal to the **TARGETDIR** property.

For an [administrative installation](#), install the administrative image into the root directory named TARGETDIR and use the source directory names to resolve the target directories.

Note the installer sets a number of standard [properties](#) to system folder paths. See the [Property Reference](#) for a list of the properties that are set to system folders.

Directory resolution is performed during the [CostFinalize action](#) and is done as follows:

### Root Destination Directory

There may only be a single root destination directory. To specify the root

destination directory, set the Directory column to the **TARGETDIR** property and the DefaultDir column to the **SourceDir** property. If the **TARGETDIR** property is defined, the destination directory is resolved to the property's value. If the **TARGETDIR** property is undefined, the **ROOTDRIVE** property is used to resolve the path.

## Root Source Directory

The value of the DefaultDir column for the root directory entry must be set to the **SourceDir** property.

## Non-root Destination Directories

The Directory value for a non-root directory is also interpreted as the name of a property defining the location of the destination. If the property is defined, the destination directory is resolved to the property's value. If the property is not defined, the destination directory is resolved to a subdirectory beneath the resolved destination directory for the Directory\_Parent entry. The DefaultDir value defines the name of the subdirectory.

## Non-root Source Directories

The source directory for a non-root directory is resolved to a subdirectory of the resolved source directory for the Directory\_Parent entry. Again, the DefaultDir value defines the name of the subdirectory.

## Short or Long File Names

When resolving destination directories, the short file names specified in the DefaultDir column are used if either the **SHORTFILENAME** property is set or the volume the directory is located on does not support long file names. Otherwise, the long file name is used.

Note that when the directories are resolved during the CostFinalize action, the keys in the Directory table become **properties** set to directory paths.

[CreateFolder Table](#)

For creating empty folders during an installation, see [CreateFolder Table](#).

### [Using the Directory Table](#)

For more information about the Directory table, including samples, see [Using the Directory Table](#).

## **Validation**

[ICE03](#)

[ICE06](#)

[ICE07](#)

[ICE30](#)

[ICE32](#)

[ICE38](#)

[ICE46](#)

[ICE48](#)

[ICE56](#)

[ICE57](#)

[ICE64](#)

[ICE88](#)

[ICE90](#)

[ICE91](#)

[ICE99](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DrLocator Table

The DrLocator table holds the information needed to find a file or directory by searching the directory tree.

The DrLocator table has the following columns.

Column	Type	Key	Nullable
Signature_	Identifier	Y	N
Parent	Identifier	Y	Y
Path	AnyPath	Y	Y
Depth	Integer	N	Y

## Columns

### Signature\_

The Signature\_ column is an external key to the first column of the [Signature table](#). This field may represent a unique file signature listed in the Signature table. If the value in this column is absent from the Signature table, then the search is assumed to be for a directory pointed to by the DrLocator table.

### Parent

This column is the signature of the parent directory of the file or directory in the Signature\_ column. If this field is null, and the Path column does not expand to a full path, then all the fixed drives of the user's system are searched by using the Path.

This field is a key into one of the following tables: the [RegLocator](#), the [IniLocator](#), the [CompLocator](#), or the DrLocator tables.

### Path

The Path column contains the path on the user's system. This is either a full path or a relative subpath below the directory specified in the Parent column. See the restrictions on the [AnyPath](#) data type.

### Depth

The depth below the path that the installer searches for the file or directory specified in the Signature\_ column. The value used in the Depth field is based on zero. For example, if the Path field is c:/Program Files/bin, the Depth column must be set to 0 or greater, to detect a file located inside the folder bin. If the Depth field is empty, the depth is assumed to be zero.

## Remarks

This table is used with the [AppSearch Table](#).

This table's columns are generally not localized. If an author decides to search for products in multiple languages, then there must be a separate entry included in the table for each language.

See [Searching for Existing Applications, Files, Registry Entries or .ini File Entries](#).

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE46](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# DuplicateFile Table

The DuplicateFile table contains a list of files that are to be duplicated, either to a different directory than the original file or to the same directory but with a different name. The original file must be a file installed by the [InstallFiles action](#).

The DuplicateFile table has the following columns.

Column	Type	Key	Nullable
FileKey	Identifier	Y	N
Component_	Identifier	N	N
File_	Identifier	N	N
DestName	Filename	N	Y
DestFolder	Identifier	N	Y

## Columns

### FileKey

A primary key, a non-localized token, identifying a unique DuplicateFile record.

### Component\_

An external key to the first column of the [Component table](#). If the component identified by the key is not selected for installation or removal, then no action is taken on this DuplicateFile record.

### File\_

Foreign key into the [File table](#) representing the original file that is to be duplicated.

### DestName

Localizable name to be given to the duplicate file. If this field is blank, then the destination file is given the same name as the original file.

### DestFolder

Name of a property that is the full path to where the duplicate file is to be copied. If this directory is the same as the directory containing the original file and the name for the proposed duplicate file is the same as the original, then no action takes place.

## Remarks

The table is processed by the [DuplicateFiles action](#) and the [RemoveDuplicateFiles action](#).

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE18](#)  
[ICE32](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Environment Table

The Environment table is used to set the values of environment variables.

The Environment table has the following columns.

Column	Type	Key	Nullable
Environment	Identifier	Y	N
Name	Text	N	N
Value	Formatted	N	Y
Component_	Identifier	N	N

## Columns

### Environment

This is the primary key of the table and is a non-localized token.

### Name

This column is the localizable name of the environment variable. The key values are written or removed depending upon which of the characters in the following table are prefixed to the name. There is no effect in the ordering of the symbols used in a prefix.

Prefix	Description
=	Create the environment variable if it does not exist, and then set it during installation. If the environment variable exists, set it during the installation.
+	Create the environment variable if it does not exist, then set it during installation. This has no effect on the value of the environment variable if it already exists.
-	Remove the environment variable when the component is removed. This symbol can be combined with any prefix.
!	Remove the environment variable during an installation. The installer only removes an environment variable during an

	installation if the name and value of the variable match the entries in the Name and Value fields of the Environment table. If you want to remove an environment variable, regardless of its value, use the '!' syntax, and leave the Value field empty.
*	This prefix is used with Windows 2000 to indicate that the name refers to a system environment variable. If no asterisk is present, the installer writes the variable to the user's environment. This symbol can be combined with any prefix. A package that is used for installation in the per-machine <a href="#">installation context</a> should write environment variables to the machine's environment by including * in the Name column. For more information, see Remarks.
=-	The environment variable is set on install and removed on uninstall. This is the usual behavior.
!-	Removes an environment variable during an install or uninstall.
=+ !+ !=	These are not a valid prefixes

If the Value field in the table includes a [~], then the prefix characters apply to only the specified portion of the string. The use of [~] is described below in the Value column section.

The environment variable is removed if the Value field of the table is blank. Therefore, with a blank in the Value field, an = prefix deletes the environment variable on install and a - prefix deletes any current values on uninstall.

## Value

This column contains the localizable value that is to be set as a formatted string. See [Formatted](#). If this field is left blank, the variable is removed. If the field is blank and the string in the Name field is prefixed by the - symbol, the variable is removed only when the component is removed.

To append a value to the end of an existing variable, prefix the string in this field by the Null character [~] and the separator character. For

example, if the semicolon is the chosen separator: [~];*Value*.

To prefix a value to the front of an existing variable, append the string in this field by the separator character and the Null character [~]. For example, if the semicolon is the chosen separator: *Value*;[~] .

If no [~] is present in the field, the string represents the entire value to be set or deleted.

Each row can contain only one value. For example, the entry *Value*;*Value*;[~] is more than one value and should not be used because it causes unpredictable results. The entry *Value*;[~] is just one value.

If Name is prefixed with +, then [~] must not be used in Value column. This is because the meaning of "+" and "[~]" are clearly exclusive of one another.

#### Component\_

An external key to the first column of the [Component table](#). This column references the component that controls the installation of the environment values.

## Remarks

For the installer to set environment variables, the [WriteEnvironmentStrings action](#) and [RemoveEnvironmentStrings action](#) need to be listed in the [InstallExecuteSequence Table](#).

Note that environment variables do not change for the installation in progress when either the [WriteEnvironmentStrings action](#) or [RemoveEnvironmentStrings action](#) are run. On Windows 2000, this information is stored in the registry and a message notifies the system of changes when the installation completes. A new process, or another process that checks for these messages, uses the new environment variables.

When modifying the path environment variable with the Environment table, do not attempt to enter the entire new path explicitly into the Value field. Instead, extend the existing path by prefixing or appending a value and delimiter (;) to [~]. If [~] is not present in the Value field, the existing

path information is lost and installing the .msi file may prevent the computer from booting. The path variable is mostly commonly set using the syntax: [~];Value.

When performing per-machine installations from a terminal server, the installer writes per-user environment variables to **HKU.DefaultEnvironment**. Because Terminal Services does not replicate this section of the registry, the installation does not set the per-user environment variables. A package used for per-machine installations should write environment variables to the computer's environment by including \* in the Name column. If the package can be installed per-user or per-machine, create two components: (1) a per-user component with the Environment table entries authored for user settings, and (2) a per-machine component with the Environment table authored for computer settings. Condition the installation of this component using the **Privileged** property.

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE32](#)  
[ICE46](#)  
[ICE65](#)  
[ICE69](#)  
[ICE80](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Error Table

The Error table is used to look up error message formatting templates when processing errors with an error code set but without a formatting template set (this is the normal situation).

The Error table has the following columns.

Column	Type	Key	Nullable
Error	Integer	Y	N
Message	Template	N	Y

## Columns

### Error

See [Windows Installer Error Messages](#) for a list of the error numbers and messages.

The error number must be a non-negative integer.

The range from 25000 to 30000 is reserved for errors from custom actions. Authors of custom actions may use this range for their custom actions.

### Message

This column contains the localizable error formatting template. The Error table is generated by the initial build process to contain the debug format templates.

The following table lists reserved messages. For a list of ship and internal error codes see [Windows Installer Error Messages](#).

Error Message	Remarks
0	Header prefix for fatal errors (INSTALLMESSAGE_FATALEXIT). Text enclosed in double curly braces <code>{{text}}</code> is only visible in the log file. The text is not displayed to the user in the UI.

1	Error [1].	Header prefix for errors (INSTALLMESSAGE_ERROR)
2	Warning [1].	Header prefix for warnings (INSTALLMESSAGE_WARNING)
3		
4	Info [1].	Header prefix for informational messages (INSTALLMESSAGE_INFO)
5	Internal Error [1]. [2]{, [3]}{, [4]}	Header prefix for internal errors
6		
7	{{Disk full: }}	Header prefix for out of disk space errors (INSTALLMESSAGE_OUTOFDISKSPACE). Text enclosed in double curly braces {{text}} is only visible in the log file. The text is not displayed to the user in the UI.
8	Action [Time]: [1]. [2]	
9	[ProductName]	
10	{{[2]}{, [3]}{, [4]}	
11	Message type: [1], Argument: [2]	
12	=== Logging started: [Date] [Time] ===	
13	=== Logging stopped: [Date] [Time] ===	
14	Action start [Time]: [1]	

15	Action ended [Time]: [1]. Return value [2]	
16	Time remaining: {[1] min }{[2] sec}	
17	Out of memory. Shutdown other applications before retrying	
18	Installer is no longer responding	
19	Installer terminated prematurely	
20	Please wait while Windows configures [ProductName]...	
21	Gathering required information...	
22	Removing older versions of this application...	
23	Preparing to remove older versions of this application...	
32	{[ProductName] }Setup completed successfully.	

33	{{[ProductName]}Setup failed.
----	-------------------------------

## Remarks

The template does not include formatting for the error number in field 1. When processing the error, the installer attaches a header prefix to the template depending on the message type. These headers are also stored in the Error table.

Text enclosed in double curly braces `{{text}}` is only visible in the log file. The text is not displayed to the user in the UI.

You can import a localized Error table into your database by using `Msidb.exe` or [MsiDatabaseImport](#). The SDK includes a localized Error table for each of the languages listed in the [Localizing the Error and ActionText Tables](#) section. If the Error table is not populated, the installer loads localized strings for the language specified by the [ProductLanguage](#) property.

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE40](#)  
[ICE46](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# EventMapping Table

The EventMapping Table lists the controls that subscribe to some control events, and lists the attribute to be changed when the event is published by another control or the Windows Installer.

The EventMapping Table has the following columns.

Column	Type	Key	Nullable
Dialog_	Identifier	Y	N
Control_	Identifier	Y	N
Event	Identifier	Y	N
Attribute	Identifier	N	N

## Columns

### Dialog\_

An external key to the first column of the [Dialog Table](#). This field and the Control\_ field together identify a control.

### Control\_

An external key to the second column of the [Control Table](#). This field and the Dialog\_ field together identify a control.

### Event

This field is an identifier that specifies the type of event that is subscribed to by the control. For more information, see [ControlEvent Overview](#).

### Attribute

The name of the Control\_ attribute that is set when the event in the Event column is received. The Argument of the event is passed as the argument of the attribute call to change this attribute of the control.

## Remarks

The [ControlEvent Table](#) specifies the control events that are started when a user interacts with a [PushButton Control](#), [CheckBox Control](#), or [SelectionTree Control](#). These are the only controls that a user can use to initiate control events.

More than one control on a dialog box can subscribe to the same event.

The following list identifies the typical uses for the EventMapping Table:

- To subscribe a [Text Control](#) to an [ActionText ControlEvent](#), [ActionData ControlEvent](#), [ScriptInProgress ControlEvent](#) or [TimeRemaining ControlEvent](#) published by the Windows Installer.
- To subscribe a [ProgressBar Control](#) or [Billboard Control](#) to a [SetProgress ControlEvent](#).
- To subscribe a [DirectoryCombo Control](#) to an [IgnoreChange ControlEvent](#).
- To automatically disable a [PushButton Control](#) located on the same dialog with a [SelectionTree Control](#). To disable the push button when no features are listed in the [SelectionTree Control](#), use the EventMapping Table to subscribe the PushButton control to a [SelectionNoItems ControlEvent](#). Enter Enable in the Attributes field of the EventMapping Table.
- To display a [Text Control](#) that shows the path to the installation location for the feature that is selected in a [SelectionTree Control](#) on the same dialog. Use the EventMapping Table to subscribe the [Text Control](#) to both a [SelectionPathOn ControlEvent](#) and [SelectionPath ControlEvent](#) published by the [SelectionTree Control](#).
- To display a [Text Control](#) that shows a description of the item highlighted in a [SelectionTree Control](#) located on the same dialog, use the EventMapping Table to subscribe the [Text Control](#) to a [SelectionDescription ControlEvent](#), [SelectionSize ControlEvent](#) or [SelectionAction ControlEvent](#). Enter Text in the Attribute field of the

EventMapping Table.

## **Validation**

[ICE03](#)

[ICE06](#)

[ICE32](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Extension Table

The Extension table contains information about file name extension servers that must be generated as a part of product advertisement. Each row generates a set of registry keys and values.

The Extension table contains the columns shown in the following table.

Column	Type	Key	Nullable
Extension	Text	Y	N
Component_	Identifier	Y	N
ProgId_	Text	N	Y
MIME_	Text	N	Y
Feature_	Identifier	N	N

## Columns

### Extension

The extension associated with the table row. The extension can be up to 255 characters long. Enter the extension in this field without the preceding period.

### Component\_

An external key to the first column of the [Component](#) table. This column references the component that controls the installation of the extension.

### ProgId\_

The Program ID associated with this extension. This is a foreign key into the [ProgId](#) table.

### MIME\_

The Content Type that is to be written for the Extension column.

An external key to the first column of the [MIME](#) table.

### Feature\_

An external key into the first column of the [Feature](#) table specifying

the feature that provides the extension server.

## Remarks

The Extension table is referred to when the [RegisterExtensionInfo](#) action or the [UnRegisterExtensionInfo](#) action is executed.

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE15](#)  
[ICE19](#)  
[ICE32](#)  
[ICE41](#)  
[ICE69](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Feature Table

The Feature Table defines the logical tree structure of features and contains the columns shown in the following table.

Column	Type	Key	Nullable
Feature	Identifier	Y	N
Feature_Parent	Identifier	N	Y
Title	Text	N	Y
Description	Text	N	Y
Display	Integer	N	Y
Level	Integer	N	N
Directory_	Identifier	N	Y
Attributes	Integer	N	N

## Columns

### Feature

The primary key that is used to identify a specific feature record. The value in this field must not exceed a maximum length of 38 characters.

### Feature\_Parent

An optional key of a parent record in the same table.

The key points to the Feature column. If the parent feature is not selected, then this feature is not installed. A null value in this field indicates that this feature does not have a parent and is a root item. The Feature\_Parent column must not equal the Feature column of the same record.

**Note** The maximum depth of any feature is 16. An [error 2701](#) results if a feature that exceeds this maximum depth exists.

### Title

A short string of text that identifies a feature.

This string is listed as an item by the [SelectionTree Control](#) of the [Selection Dialog](#).

#### Description

A longer string of text that describes a feature.

This localizable string is displayed by the [Text Control](#) of the [Selection Dialog](#).

#### Display

The number in this field specifies the order in which the feature is to be displayed in the user interface.

The value also determines whether or not the feature is initially displayed expanded or collapsed. If the value is null or 0 (zero), the record is not displayed.

- If the value is odd, the feature node is expanded initially.
- If the value is even, the feature node is collapsed initially.

#### Level

The initial installation level of this feature. Processing the [Condition Table](#) can modify the level value.

An install level of 0 (zero) disables the item and prevents it from being displayed. A feature with an installation level of 0 (zero) is not installed during any installation, including administrative installations. For more information, see the "Install Level" information in the Remarks section of this topic.

#### Directory\_

The `Directory_` column specifies the name of a directory that can be configured by a [Selection Dialog](#).

Because this field is a key into the [Directory Table](#), the specified directory must be listed in the first column of the Directory Table. You must enter a [Public Property](#) in this column to make the directory configurable, and to display a **Browse** button on the [Selection Dialog](#).

#### Attributes

The remote execution option for features that are not installed and

for which no feature state request is made by using any of the following properties.

- **ADDLOCAL Property**
- **ADDSOURCE Property**
- **ADDDEFAULT Property**
- **COMPADDLOCAL Property**
- **COMPADDSOURCE Property**
- **FILEADDLOCAL Property**
- **FILEADDSOURCE Property**
- **REMOVE Property**
- **REINSTALL Property**
- **ADVERTISE Property**

Add the indicated bits to the total value of this column to include a remote execution option.

- If this field is blank, the value defaults to 0 (zero), msidbFeatureAttributesFavorLocal.
- If the feature install level is 0 (zero), or greater than or equal to the current install level, no change is made in the feature state.

<b>Name</b>	<b>Decimal</b>	<b>Hexadecimal</b>
msidbFeatureAttributesFavorLocal	0	0x0000

msidbFeatureAttributesFavorSource	1	0x0001
msidbFeatureAttributesFollowParent	2	0x0002

msidbFeatureAttributesFavorAdvertise	4	0x0004
msidbFeatureAttributesDisallowAdvertise	8	0x0008

msidbFeatureAttributesUIDisallowAbsent	16	0x0010
msidbFeatureAttributesNoUnsupportedAdvertise	32	0x0020

--	--	--

Some attributes are exclusive of each other. Attempting to set these attributes together on the same feature causes the installation package to fail **Package Validation**.

- Do not use `msidbFeatureAttributesFavorAdvertise` with `msidbFeatureAttributesDisallowAdvertise`.
- Do not use `msidbFeatureAttributesNoUnsupportedAdvertise` with `msidbFeatureAttributesDisallowAdvertise` together.
- Do not use `msidbFeatureAttributesFollowParent` with `msidbFeatureAttributesFavorSource`.
- Note that the `msidbFeatureAttributesFollowParent` and `msidbFeatureAttributesFavorLocal` values are mutually exclusive. If the `msidbFeatureAttributesFollowParent` value is used, the `msidbFeatureAttributesFavorLocal` value is assumed to not exist.

Note that if a child feature is installed, its parent feature is also installed. If a parent feature is installed, its child feature is not necessarily installed unless its `msidbFeatureAttributesFollowParent` and `msidbFeatureAttributesUIDisallowAbsent` attributes are set. This hierarchical relationship of the installation of parent and child features is also used for the GUI installations and installations that use command-line properties.

## Remarks

Several additional temporary columns are added to this table when it is loaded into memory for computations used by costing and user interface (UI) selection.

A component can be shared between two or more features or applications. If two or more features refer to the same component, then

that component is selected for installation if any of the associated features are selected. This can also be the reason child features are not uninstalled when a parent feature is removed. If the child feature consists of components needed by other features or applications, the Windows Installer does not remove the child feature.

For more information, see [Controlling Feature Selection States](#).

Install Level:

- For any installation, there is a defined install level, which is an integral value from 1 to 32,767. The initial value is determined by the [INSTALLLEVEL Property](#), which is set in the [Property Table](#).
- A feature is installed only if the feature level value is less than or equal to the current install level. The UI can be authored so that when the installation is initialized, the Installer allows the user to modify the install level of any feature in the Feature Table. For example, an author can define install level values that represent specific installation options, such as **Custom**, **Typical**, or **Minimum**, and then create a dialog box that uses [SetInstallLevel ControlEvents](#) to enable the user to select one of these states.
- Depending on the state the user selects, the dialog box sets the install level property to the corresponding value. If the author assigns **Typical** a level of 100 and the user selects **Typical**, only those features with a level of 100 or less are installed. In addition, the **Custom** option could lead to another dialog box that contains a [SelectionTree Control](#). The SelectionTree Control then allows the user to individually change whether or not each feature is installed.

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE10](#)  
[ICE14](#)  
[ICE21](#)

ICE32  
ICE41  
ICE45  
ICE47  
ICE50  
ICE57  
ICE59  
ICE62  
ICE67  
ICE79  
ICE86  
ICE94

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# FeatureComponents Table

The FeatureComponents table defines the relationship between features and components. For each feature, this table lists all the components that make up that feature.

The FeatureComponents table has the following columns.

Column	Type	Key	Nullable
Feature_	<a href="#">Identifier</a>	Y	N
Component_	<a href="#">Identifier</a>	Y	N

## Columns

Feature\_

An external key into the first column of the [Feature table](#).

Component\_

An external key into the first column of the [Component table](#).

## Remarks

There is a maximum limit of 1600 components per feature.

Components can be shared by two or more features, that is, the same component can be referred to by more than one feature.

This table is referred to when the [PublishFeatures action](#) or the [UnpublishFeatures action](#) is executed.

## Validation

[ICE03](#)

[ICE06](#)

[ICE21](#)

[ICE22](#)

ICE32  
ICE41  
ICE47  
ICE59  
ICE62  
ICE69

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# File Table

The File Table contains a complete list of source files with their various attributes, ordered by a unique, non-localized, identifier. Files can be stored on the source media as individual files or compressed within a *cabinet file*. For more information, see [Using Cabinets and Compressed Sources](#).

The File Table has the following columns.

Column	Type	Key	Nullable
File	Identifier	Y	N
Component_	Identifier	N	N
FileName	Filename	N	N
FileSize	DoubleInteger	N	N
Version	Version	N	Y
Language	Language	N	Y
Attributes	Integer	N	Y
Sequence	Integer	N	N

## Columns

### File

A non-localized token that uniquely identifies the file. This field is insensitive to case. Do not assign identifiers to different files that differ only by their case.

### Component\_

The external key into the first column of the [Component Table](#). This field identifies the Component that controls the file.

### FileName

The file name used for installation. The name may be localized.

Because some web servers can be case sensitive, FileName should

match the case of the source files exactly to ensure support of Internet downloads.

#### FileSize

The size of the file in bytes. This must be a non-negative number.

#### Version

This field is the version string for a versioned file. This field is blank for non-versioned files. The file version entered into this field must be identical to the version of the file included with the installation package.

The Version field can also be set to contain the primary key of another record in the File table. The referenced file then determines the versioning logic for this file. For more information, see [Companion Files](#). Note that if this file is the key path for its component, it must not be specified as a companion file.

#### Language

A list of decimal language IDs separated by commas.

Font files should not be authored with a language ID, as fonts do not have an embedded language ID resource. Thus this column should be left null for font files.

#### Attributes

The integer that contains bit flags that represent file attributes.

The following table shows the definition of the bit field.

<b>Constant</b>	<b>Hexadecimal</b>	<b>Decimal</b>	<b>Description</b>
msidbFileAttributesReadOnly	0x000001	1	Read-Only
msidbFileAttributesHidden	0x000002	2	Hidden
msidbFileAttributesSystem	0x000004	4	System
msidbFileAttributesVital	0x000200	512	The file is v operation of which it bel of a file wtl msidbFileA fails, the ins rolled back.

			<p>Installer displays a warning message without an Internet connection. If this attribute is set, the installer displays a warning message during installation. If the user selects an Ignore button, the installer continues. If the user selects a Cancel button, the user can check for updates. If the user selects a failure to install button, the installer continues.</p>
msidbFileAttributesChecksum	0x000400	1024	<p>The file contains a checksum. A checksum is a value that is calculated for a file that has been compressed.</p>
msidbFileAttributesPatchAdded	0x001000	4096	<p>This bit must be set if the file is a patch and if the patch is applied by the patching engine.</p>
msidbFileAttributesNoncompressed	0x002000	8192	<p>The file's source is uncompressed. <b>Word Count</b> If neither msidbFileAttributesCompressed or msidbFileAttributesNoncompressed are set, the file is specified as compressed. <b>Summary</b> If both msidbFileAttributesCompressed and msidbFileAttributesNoncompressed are set, the file is specified as compressed.</p>
msidbFileAttributesCompressed	0x004000	16384	<p>The file's source is compressed. <b>Word Count</b> If neither msidbFileAttributesCompressed or msidbFileAttributesNoncompressed are set, the file is specified as uncompressed.</p>

			<b>Summary F</b> both msidbFileAt and msidbFileAt
--	--	--	---

If the msidbFileAttributesVital bit within the Attributes column is set, and if the component to which the file belongs is selected for installation, then the installer must be able to install this file for the installation to be completed successfully. If the installer is unable to install the file for some reason (for example, if the source file cannot be located within the source image), then an error dialog box will appear with the options "Retry" or "Cancel". For a file that does not have msidbFileAttributesVital set, the options in case of an install error will be "Abort", "Retry", and "Ignore" (that is, the user will have the option to complete the install successfully without installing that file).

The msidbFileAttributesChecksum bit within the Attributes column should be set for every executable file in the installation that has a valid checksum stored in the Portable Executable (PE) file header. Only those files that have this bit set will ever be verified for valid checksum during a reinstall. For more information, see the [REINSTALLMODE](#).

### Sequence

Sequence position of this file on the media images. This order must correspond to the order of the files in the cabinet if the files are compressed. The integers in this field must be equal or greater than 1.

The sequence numbers in the Sequence column are used to specify both the order of installation for files and the source media upon which the file is located (in conjunction with the [Media Table](#)). For example, suppose a file has a sequence number of 92. To determine the source disk this file resides on, look in the Media table for the entry with the smallest Last Sequence value that is larger than 92.

Although compressed files are assigned internal sequence numbers within cabinets, those absolute numbers do not need to match the

sequence numbers within the File table. It is, however, important that the sequence of files in the File table be identical to the sequence of the files within the cabinets.

For files that are not compressed, the sequence numbers need not be unique. For instance, if all your files are uncompressed, and all reside on one disk, you could give all the files the same sequence number.

The maximum limit is 32767 files. To create a Windows Installer package with more files, see [Authoring a Large Package](#).

## Remarks

The [InstallFiles](#) and [RemoveFiles](#) actions in the *sequence tables* process the information in this table. For information about using sequence tables, see [Using a Sequence Table](#).

The table is initially generated from the file list, but if cabinet compression is used, the table is regenerated from the output of the compression engine. For more information, see [Cabinet Files](#).

To move an existing file on the user's computer during the installation use the [MoveFiles Action](#) and [MoveFile Table](#). To install a file to multiple locations use the [DuplicateFiles Action](#) and the [DuplicateFile Table](#).

The following table summarizes the possible combinations of values in the Version column and the Language column. For more information, see [File Versioning Rules](#).

Version	Language	Description
1.2.3.4	1033	The version and language.
1.2.3.4	(Null)	The version but no language.
1.2.3.4	0	The version and language are neutral.
Testdb	(Null)	The companion file with no language associated with it.
Testdb	1033	The companion file and language.

(Null)	1033	No version, but has a language associated with it (that is, typelib, helpfile).
--------	------	---

For more information, see the [MsiLockPermissionsEx Table](#) and [LockPermissions Table](#).

## Validation

[ICE02](#)  
[ICE03](#)  
[ICE04](#)  
[ICE06](#)  
[ICE18](#)  
[ICE30](#)  
[ICE32](#)  
[ICE35](#)  
[ICE39](#)  
[ICE42](#)  
[ICE45](#)  
[ICE50](#)  
[ICE51](#)  
[ICE54](#)  
[ICE55](#)  
[ICE57](#)  
[ICE59](#)  
[ICE60](#)  
[ICE67](#)  
[ICE69](#)  
[ICE76](#)  
[ICE91](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# FileSFPCatalog Table

The FileSFPCatalog table associates specified files with the catalog files used by system file protection.

**Windows Vista, Windows Server 2003, Windows XP, and Windows 2000:** Not supported.

The FileSFPCatalog table has the following columns.

Column	Type	Key	Nullable
File_	Identifier	Y	N
SFPCatalog_	Filename	Y	N

## Columns

File\_  
Foreign key to the [File table](#).

SFPCatalog\_  
Foreign key to the [SFPCatalog table](#).

## Remarks

The [InstallSFPCatalogFile action](#) queries the [Component table](#), [File table](#), FileSFPCatalog table and [SFPCatalog table](#).

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE32](#)  
[ICE66](#)  
[ICE76](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Font Table

The Font table contains the information for registering font files with the system.

The Font table has the following columns.

Column	Type	Key	Nullable
File_	Identifier	Y	N
FontTitle	Text	N	Y

## Columns

### File\_

External key into the [File table](#) entry for the font file. It is recommended that the component containing the font file have the FontsFolder specified in the Directory\_ column of the [Component table](#).

### FontTitle

Font name. It is recommended that you leave this column null for TrueType Fonts and TrueType Collections because the installer can register the font after reading the correct font title from the font file. If the font name is entered, it must be identical to font title from the font file. You must specify a title for fonts that do not have embedded names, such as .fon files.

## Remarks

This table is referred to when the [RegisterFonts action](#) or the [UnregisterFonts action](#) is executed.

If the FontTitle field is left Null, the Font name is read directly from the font file specified. If the font name recorded into the FontTitle field differs from the internal font name recorded in the font file, the font is registered twice by the [RegisterFonts action](#).

Font files should not be authored with a language ID, as fonts do not have an embedded language ID resource. Thus the Language column of the [File table](#) should be left null for font files.

Because the installer does not recount font files by default, preexisting font files may be removed with their component when uninstalling an application. To ensure that a font file is not removed, authors may set the `msidbComponentAttributesSharedDllRefCount` or `msidbComponentAttributesPermanent` bit flags in the Attributes column of the Component Table\_ansi\_Component\_Table for the component containing the font file.

## Validation

[ICE03](#)

[ICE06](#)

[ICE07](#)

[ICE32](#)

[ICE51](#)

[ICE60](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Icon Table

This table contains the icon files. Each icon from the table is copied to a file as a part of product advertisement to be used for advertised shortcuts and OLE servers. See [OLE Limitations on Streams](#).

The Icon table has the following columns.

Column	Type	Key	Nullable
Name	Identifier	Y	N
Data	Binary	N	N

## Columns

Name

Name of the icon file.

Data

The binary icon data in PE (.dll or .exe) or icon (.ico) format.

## Remarks

This table is referred to when the [PublishProduct action](#) is executed.

The icons for shortcuts, file name extensions, and CLSIDs must be stored in files that are separate from the target file itself. This is required because the installer should copy only the small icon files to the user's machine when advertising the resource. A developer of an installation package therefore needs to author separate files containing only the icons. These icon files are then stored as binary data in the Icon table.

Icon files that are associated strictly with file name extensions or CLSIDs can have any extension, such as .ico. However, Icon files that are associated with shortcuts must be in the EXE binary format and must be named such that their extension matches the extension of the target. The shortcut will not work if this rule is not followed. For example, if a shortcut

is to point to a resource having the key file Red.bar, then the icon file must also have the extension .bar. Multiple icons can be stuffed into the same icon file as long as all of the target files have the same extension.

## Validation

[ICE03](#)

[ICE06](#)

[ICE29](#)

[ICE32](#)

[ICE36](#)

[ICE50](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IniFile Table

The IniFile table contains the .ini information that the application needs to set in an .ini file.

The IniFile table has the following columns.

Column	Type	Key	Nullable
IniFile	Identifier	Y	N
FileName	FileName	N	N
DirProperty	Identifier	N	Y
Section	Formatted	N	N
Key	Formatted	N	N
Value	Formatted	N	N
Action	Integer	N	N
Component_	Identifier	N	N

## Columns

### IniFile

The key for this table.

### FileName

The localizable name of the .ini file in which to write the information.

### DirProperty

Name of a property having a value that resolves to the full path of the folder containing the .ini file. The property can be the name of a directory in the [Directory table](#), a property set by the [AppSearch table](#), or any other property that represents a full path. If this field is left blank, the .ini file is created in the folder having the full path specified by the [WindowsFolder](#) property.

### Section

The localizable .ini file section.

**Key**

The localizable .ini file key within the section.

**Value**

The localizable value to be written.

**Action**

The type of modification to be made.

<b>Constant</b>	<b>Hexadecimal</b>	<b>Decimal</b>	<b>Modification</b>
msidbIniFileActionAddLine	0x000	0	Creates or updates a .ini entry.
msidbIniFileActionCreateLine	0x001	1	Creates a .ini entry only if the entry does not already exist.
msidbIniFileActionAddTag	0x003	3	Creates a new entry or appends a new comma-separated value to an existing entry.

**Component\_**

External key into the first column of the [Component table](#) referencing the component that controls the installation of the .ini value.

**Remarks**

The .ini file information is written out when the corresponding component has been selected to be installed as local or run from source.

This table is referred to when the [WriteIniValues action](#) or the [RemoveIniValues action](#) is executed.

## Validation

[ICE03](#)

[ICE06](#)

[ICE32](#)

[ICE46](#)

[ICE69](#)

[ICE88](#)

[ICE91](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IniLocator Table

The IniLocator table holds the information needed to search for a file or directory using an .ini file or to search for a particular .ini entry itself. The .ini file must be present in the default Microsoft Windows directory.

The IniLocator table has the following columns.

Column	Type	Key	Nullable
Signature_	Identifier	Y	N
FileName	FileName	N	N
Section	Text	N	N
Key	Text	N	N
Field	Integer	N	Y
Type	Integer	N	Y

## Columns

### Signature\_

An external key into the first column of the [Signature table](#). The Signature\_ represents a unique signature and is also the external key into column one of the Signature table. If this signature is present in the Signature table, then the search is for a file. If this key is absent from the Signature table, and the value of the Type column is msidbLocatorTypeRawValue, the search is for the .ini entry specified by the IniLocator table. Otherwise the search is for a directory specified by the IniLocator table.

### FileName

The .ini file name.

### Section

Section name within the .ini file.

### Key

Key value within the section.

## Field

The field in the .ini line. If Field is Null or 0, then the entire line is read. This must be a non-negative number.

## Type

A value that determines whether the .ini value is a file location, a directory location, or raw .ini value.

The following table lists valid values. If absent, Type is set to be 1.

Constant	Hexadecimal	Decimal	Description
msidbLocatorTypeDirectory	0x000	0	A directory location.
msidbLocatorTypeFileName	0x001	1	A file location.
msidbLocatorTypeRawValue	0x002	2	A raw .ini value.

## Remarks

This table is used with the [AppSearch Table](#).

This table's columns are generally not localized. If an author decides to search for products in multiple languages, then there can be a separate entry included in the table for each language.

Associated localized text for progress display or logging is specified in the [ActionText table](#).

See [Searching for Existing Applications, Files, Registry Entries or .ini File Entries](#).

## Validation

ICE03

ICE06

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# InstallExecuteSequence Table

The InstallExecuteSequence table lists actions that are executed when the top-level [INSTALL action](#) is executed.

Actions in the install sequence up to the [InstallValidate action](#), and any exit dialog boxes, are located in the [InstallUISequence table](#). All actions from the InstallValidate through the end of the install sequence are in the InstallExecuteSequence table. Because the InstallExecuteSequence table needs to stand alone, it has any necessary initialization actions such as the [LaunchConditions](#), [CostInitialize](#), [FileCost](#), and [CostFinalize](#) actions.

[Custom actions](#) requiring a user interface should use [MsiProcessMessage](#) instead of authored dialog boxes created using the [Dialog table](#).

The InstallExecuteSequence table has the following columns.

Column	Type	Key	Nullable
Action	<a href="#">Identifier</a>	Y	N
Condition	<a href="#">Condition</a>	N	Y
Sequence	<a href="#">Integer</a>	N	Y

## Columns

### Action

Name of the action to execute. This is either a built-in action or a custom action.

Primary table key.

### Condition

This field contains a conditional expression. If the expression evaluates to False, then the action is skipped. If the expression syntax is invalid, then the sequence terminates, returning `iesBadActionData`. For information on the syntax of conditional statements, see [Conditional Statement Syntax](#).

## Sequence

Number that determines the sequence position in which this action is to be executed.

A positive value represents the sequence position. A Null value indicates that the action is not executed. The following negative values indicate that this action is to be executed if the installer returns the associated termination flag. Each termination flag (negative value) can be used with no more than one action. Multiple actions can have termination flags, but they must be different flags. Termination flags (negative values) are typically used with [Dialog Boxes](#).

Termination flag	Value	Description
msiDoActionStatusSuccess	-1	Successful completion. Used with <a href="#">Exit</a> dialog boxes.
msiDoActionStatusUserExit	-2	User terminates install. Used with <a href="#">UserExit</a> dialog boxes.
msiDoActionStatusFailure	-3	Fatal exit terminates. Used with a <a href="#">FatalError</a> dialog boxes.
msiDoActionStatusSuspend	-4	Install is suspended.

Zero, all other negative numbers, or a Null value indicate that the action is never run.

## Remarks

Localized text for progress display or logging is specified in the [ActionText table](#).

For an example of a sequence table, see [Using a Sequence Table](#).

## Validation

[ICE03](#)

[ICE06](#)

ICE12  
ICE13  
ICE26  
ICE27  
ICE28  
ICE46  
ICE63  
ICE75  
ICE77  
ICE79  
ICE82  
ICE83  
ICE84  
ICE86

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# InstallUISequence Table

The InstallUISequence table lists actions that are executed when the top-level [INSTALL action](#) is executed and the internal user interface level is set to full UI or reduced UI. The installer skips the actions in this table if the user interface level is set to basic UI or no UI. See [About the User Interface](#).

Actions in the install sequence up to the [InstallValidate action](#), and the exit dialog boxes, are located in the InstallUISequence table. All actions from the InstallValidate through the end of the install sequence are in the [InstallExecuteSequence table](#). Because the InstallExecuteSequence table needs to stand alone, it has any necessary initialization actions such as the [LaunchConditions](#), [CostInitialize](#), [FileCost](#), and the [CostFinalize](#), and [ExecuteAction action](#).

The InstallUISequence table has the following columns.

Column	Type	Key	Nullable
Action	<a href="#">Identifier</a>	Y	N
Condition	<a href="#">Condition</a>	N	Y
Sequence	<a href="#">Integer</a>	N	Y

## Columns

### Action

Name of the action to execute. This is either a built-in action, a custom action, or a user interface wizard.

Primary table key.

### Condition

This field contains a conditional expression. If the expression evaluates to False, then the action is skipped. If the expression syntax is invalid, then the sequence terminates, returning `iesBadActionData`. For information on the syntax of conditional statements, see [Conditional Statement Syntax](#).

## Sequence

The number in this column determines the sequence position in which this action is run.

A positive value represents the sequence position. A Null value indicates that the action is never run. The following negative values indicate that this action is executed if the installer returns the associated termination flag. Each termination flag (negative value) can be used with no more than one action. Multiple actions can have termination flags, but they must be different flags. Termination flags (negative values) are typically used with [Dialog Boxes](#).

Termination flag	Value	Description
msiDoActionStatusSuccess	-1	Successful completion. Used with <a href="#">Exit</a> dialog boxes.
msiDoActionStatusUserExit	-2	User terminates install. Used with <a href="#">UserExit</a> dialog boxes.
msiDoActionStatusFailure	-3	Fatal exit terminates. Used with a <a href="#">FatalError</a> dialog boxes.
msiDoActionStatusSuspend	-4	Install is suspended.

Zero, all other negative numbers, or a Null value indicate that the action is never run.

## Remarks

Associated localized text for progress display or logging is specified in the [ActionText](#) table.

For an example of a sequence table, see [Using a Sequence Table](#).

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE12](#)

ICE13  
ICE20  
ICE26  
ICE27  
ICE28  
ICE46  
ICE75  
ICE79  
ICE82  
ICE86

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# IsolatedComponent Table

Each record of the IsolatedComponent table associates the component specified in the Component\_Application column (commonly an .exe) with the component specified in the Component\_Shared column (commonly a shared DLL). The [IsolateComponents action](#) installs a copy of Component\_Shared into a private location for use by Component\_Application. This isolates the Component\_Application from other copies of Component\_Shared that may be installed to a shared location on the computer. See [Isolated Components](#).

To link one Component\_Shared to multiple Component\_Application, include a separate record for each pair in the IsolatedComponents table. The installer copies the files of Component\_Shared into the directory of each Component\_Application that is installed.

The IsolatedComponent table has the following columns.

Column	Type	Key	Nullable
Component_Shared	Identifier	Y	N
Component_Application	Identifier	Y	N

## Columns

### Component\_Shared

Foreign key into the [Component table](#). The component that contains the shared file, usually a DLL. The DLL should be the key file for this component. This must be a different component than listed in the Component\_Application column.

The shared component controls the registration for all the isolated copies of the component and must have the msidbComponentAttributesSharedDllRefCount flag set in the Attributes column of the Component table. This ensures that the installer can manage the life of the shared component.

### Component\_Application

Foreign key into the [Component table](#). The component that contains

the .exe which loads the shared file. The .exe should be the key file for this component. This must be a different component than listed in the Component\_Shared column.

## Validation

[ICE03](#)

[ICE06](#)

[ICE32](#)

[ICE62](#)

[ICE66](#)

[ICE97](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# LaunchCondition Table

The LaunchCondition table is used by the [LaunchConditions action](#). It contains a list of conditions that all must be satisfied for the installation to begin.

The LaunchCondition table has the following columns.

Column	Type	Key	Nullable
Condition	<a href="#">Condition</a>	Y	N
Description	<a href="#">Formatted</a>	N	N

## Columns

### Condition

Expression that must evaluate to True for installation to begin.

### Description

Localizable text to display when the condition fails and the installation must be terminated.

## Remarks

You cannot guarantee the order in which the launch conditions are evaluated by authoring this table. If it is necessary to control the order in which conditions are evaluated, you should do this by using [Custom Action Type 19](#) custom actions in your installation.

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE46](#)  
[ICE79](#)  
[ICE86](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ListBox Table

The lines of a list box are not treated as individual controls, but they are part of a list box that functions as a control. The ListBox table defines the values for all list boxes.

The ListBox table has the following columns.

Column	Type	Key	Nullable
Property	Identifier	Y	N
Order	Integer	Y	N
Value	Formatted	N	N
Text	Formatted	N	Y

## Columns

### Property

A named property to be tied to this item. All the items tied to the same property become part of the same list box.

### Order

A positive integer used to determine the ordering of the items that appear in a single list box. If the list box is defined as ordered, then all items should have an Order value. The integers do not have to be consecutive. If the list box is defined as unordered, then this column is ignored.

### Value

The value string associated with this item. Selecting the line sets the associated property to this value.

### Text

The localizable, visible text to be assigned to the item. If this entry or the entire column is missing, the text defaults to the corresponding entry in Value.

## Remarks

The contents of the Value and Text fields are formatted by the **MsiFormatRecord** function when the control is created, therefore they can contain any expression that the **MsiFormatRecord** function can interpret. The formatting occurs only when the control is created, and it is not updated if a property involved in the expression is modified during the life of the control.

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE17](#)  
[ICE20](#)  
[ICE46](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Listview Table

The lines of a listview are not treated as individual controls, but they are part of a listview that functions as a control. The ListView table defines the values for all listviews.

The ListView table has the following columns.

Column	Type	Key	Nullable
Property	Identifier	Y	N
Order	Integer	Y	N
Value	Formatted	N	N
Text	Formatted	N	Y
Binary_	Identifier	N	Y

## Columns

### Property

A named property to be tied to this item. All the items tied to the same property become part of the same listview.

### Order

A positive integer used to determine the ordering of the items that appear in a single listview list. The integers do not have to be consecutive. If a listview is defined as ordered, then all the items should have an Ordering value. If the listview is defined as unordered, then this column is ignored.

### Value

The value string associated with this item. Selecting the line sets the associated property to this value.

### Text

The visible, localizable text to be assigned to the item. If this entry or the entire column is missing, then the text defaults to the corresponding entry in Value.

Binary\_

The image data for the icon. This is a foreign key to the [Binary table](#).

## Remarks

The contents of the Value and Text fields are formatted by the [MsiFormatRecord](#) function when the control is created, therefore they can contain any expression that the MsiFormatRecord function can interpret. The formatting occurs only when the control is created, and it is not updated if a property involved in the expression is modified during the life of the control.

## Validation

[ICE03](#)

[ICE06](#)

[ICE17](#)

[ICE32](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# LockPermissions Table

The LockPermissions Table is used to secure individual portions of an application in a locked-down environment. It can be used with the installation of files, registry keys, and created folders.

A package intended for installation in Windows Server 2008 R2 or Windows 7 should use the [MsiLockPermissionsEx Table](#) rather than the LockPermissions Table. Windows Installer versions earlier than Windows Installer 5.0 ignore the MsiLockPermissionsEx Table. Windows Installer 5.0 can install a package that contains the LockPermissions Table. Beginning with Windows Installer 5.0, installation of a package that contains both the MsiLockPermissionsEx Table and the LockPermissions Table fails and returns Windows Installer error message 1941.

The LockPermissions Table has the following columns.

Column	Type	Key	Nullable
LockObject	Identifier	Y	N
Table	Text	Y	N
Domain	Formatted	Y	Y
User	Formatted	Y	N
Permission	DoubleInteger	N	Y

## Columns

### LockObject

This column and the Table column together specify the file, directory, or registry key that is to be secured. The LockObject column is a foreign key that points to the primary key of the table specified by the Table column.

### Table

This column and the LockObject column specify the file, directory, or registry key that is to be secured. In the Table column, enter File, Registry, or CreateFolder to specify a LockObject listed in the [File](#)

[Table](#), [Registry Table](#), or [CreateFolder Table](#).

#### Domain

The column that identifies the domain of the user for which permissions are to be set. This is the name of a stand-alone machine or a domain name. The column data type is [Formatted](#), and you may use the string [%USERDOMAIN] in this field to get the value of the USERDOMAIN environment variable for the current domain. To get any other domain requires using [Custom Actions](#). For more information, see the [Custom Action Table](#).

#### User

The column that identifies the localized name of the user for which permissions are to be set. This name must be located on the machine or domain. The installation fails if the machine or domain controller does not recognize the domain and user combination or if the user's security identifier (SID) cannot be retrieved. Multiple users can be specified for a single LockObject.

The common user names "Everyone" and "Administrators" may be entered in English and are mapped to well-known SIDs.

LocalSystem is given full control in all security descriptors created through the LockPermissions Table. You can use the [ComputerName Property](#), [LogonUser Property](#) or [USERNAME Property](#) in this field to get the current user. A custom action is required to enter the localized name of any other user or group.

You can use multiple records with identical LockObject and Table entries (but different User entries) to specify access control lists for multiple users.

#### Permission

The column that identifies the integer description of system privileges. The following gives the most commonly used values (a complete list exists in Winnt.h).

<b>Privilege</b>	<b>Description</b>
GENERIC_ALL 0X10000000 268435456	Read, write, and execute access

GENERIC_EXECUTE 0X20000000 536870912	Execute access
GENERIC_WRITE 0X40000000 1073741824	Write access

You cannot specify `GENERIC_READ` in the Permission column. Attempting to do so will fail. Instead, you must specify a value such as `KEY_READ` or `FILE_GENERIC_READ`.

Null entered in this column is reserved for future use.

## Remarks

The [InstallFiles](#), [WriteRegistryValues](#), and [CreateFolders](#) actions in *sequence tables* process the information in this table. For information about using sequence tables, see [Using a Sequence Table](#).

Permission can only be set in the LockPermissions Table for users that already exist on the computer or domain. An attempt to set permissions for an unknown user causes the installation to fail, even if that user account is created during the installation by a deferred custom action.

It is recommended that the system administrator's local group be included in all access control lists (ACL). This ensures that the system administrator can access and maintain objects.

Every file, registry key, or directory that is listed in the LockPermissions Table receives an explicit security descriptor, whether it replaces an existing object or not. The Windows Installer attempts to preserve the security on objects that already exist on the system. If an object is not listed in the LockPermissions Table, and replaces an existing object, the replacement gets the security settings of the object that it replaces.

If an object is not listed in the LockPermissions Table, and does not

replace an existing object, it receives no explicit security descriptor. The access to the new object is based on the attributes of its parent or container object. If an object is not listed in the table, and replaces an object with no explicit security descriptor, the access to the new object is based on the attributes of its parent or container object.

The Windows Installer sets the **UserSID** property to the security identifier (SID) or the user running the installation.

## Validation

[ICE03](#)

[ICE06](#)

[ICE46](#)

[ICE55](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Media Table

The Media table describes the set of disks that make up the source media for the installation.

The Media table contains the columns shown in the following table.

Column	Type	Key	Nullable
DiskId	Integer	Y	N
LastSequence	Integer	N	N
DiskPrompt	Text	N	Y
Cabinet	Cabinet	N	Y
VolumeLabel	Text	N	Y
Source	Property	N	Y

## Columns

### DiskId

Determines the sort order for the table. This number must be equal to or greater than 1.

### LastSequence

File sequence number for the last file for this media. The numbers in the LastSequence column specify which of the files in the [File](#) table are found on a particular source disk. Each source disk contains all files with sequence numbers (as shown in the Sequence column of the File table) less than or equal to the value in the LastSequence column, and greater than the LastSequence value of the previous disk (or greater than 0, for the first entry in the Media table). This number must be non-negative; the maximum limit is 32767 files. For more information about creating a Windows Installer package with more files, see [Authoring a Large Package](#).

### DiskPrompt

The disk name, which is usually the visible text printed on the disk.

This localizable text is used to prompt the user when this disk needs to be inserted.

### Cabinet

The name of the cabinet if some or all of the files stored on the media are compressed into a cabinet file. If no cabinets are used, this column must be blank. The name of the cabinet must use the syntax of the [Cabinet](#) data type. Note that to conserve disk space, the installer always removes any cabinets embedded in the .msi file before caching the installation package on the user's computer.

### VolumeLabel

The label attributed to the volume. This is the volume label returned by the **GetVolumeInformation** function. If the [SourceDir](#) property refers to a removable (floppy or CD-ROM) volume, then this volume label is used to verify that the proper disk is in the drive before attempting to install files. The entry in this column must match the volume label of the physical media.

### Source

This field is only used by patching and is otherwise left blank. A patch transform can enter a property here that is the location of the cabinet file containing the patch files or any new files added by the patch. A different source needs to be specified for these files because the source of the patch package can be stored separately from the product's source. If the Cabinet field is empty, the installer ignores the value in this column. If this field is empty, the installer uses the value of the **SourceDir** property as the source of the cabinet.

## Remarks

If the cabinet name is preceded by a number sign (#), then the files referencing this Media table record are packed in a cabinet file that is stored within the database as a separate stream.

For more information about how to add cabinets to the File tables and Media tables, see [Using Cabinets and Compressed Sources](#).

Windows Installer requires that the .msi file be on the first disk of

removable media (CD, DVD, floppy, etc.) used for the product's installation.

## Determining the SourceMode

The **Word Count Summary** property determines the source mode for the current install. If this property is set to 2 or 3, a cabinet install is assumed. In this mode, the cabinet files are assumed to exist in the directory indicated by the **SourceDir** property. If the Source Type value is 0 or 1, all source files are assumed to exist in the tree whose root is indicated by the **SourceDir** property.

Note that this only applies to files in the File table that do not have either the Compressed or Uncompressed bits set in the attributes column. These bits override the value of the **Word Count Summary** property when determining if a particular file is compressed or uncompressed.

## Validation

[ICE03](#)

[ICE04](#)

[ICE06](#)

[ICE35](#)

[ICE58](#)

[ICE71](#)

[ICE81](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MIME Table

The MIME table associates a MIME content type with a file name extension or a CLSID to generate the extension or COM server information required for advertisement of the MIME (Multipurpose Internet Mail Extensions) content.

The MIME table has the following columns.

Column	Type	Key	Nullable
ContentType	Text	Y	N
Extension_	Text	N	N
CLSID	GUID	N	Y

## Columns

### ContentType

This column is an identifier for the MIME content. It is commonly written in the form of type/format.

### Extension\_

This column contains the server extension that is to be associated with the MIME content, without the dot. This column is a foreign key into the [Extension table](#). The Extension table contains file name extension server information which is used as a part of advertisement.

### CLSID

This column contains the COM server CLSID that is to be associated with the MIME content. The CLSID in this column can be a foreign key into the [Class table](#) or it can be a CLSID that already exists on the user's machine. The Class table contains COM server-related information which is used as a part of advertisement.

## Remarks

This table is referred to when the [RegisterMIMEInfo action](#) or the [UnregisterMIMEInfo action](#) is executed. In advertise mode, the RegisterMIMEInfo action registers all MIME information for servers from the MIME table for which the corresponding feature is enabled. Otherwise the action registers MIME information for servers from the MIME table for which the corresponding feature is currently selected to be installed. The [UnregisterMIMEInfo action](#) unregisters MIME-related registry information from the system. The action unregisters MIME information for servers from the MIME table for which the corresponding feature is currently selected to be uninstalled.

## Validation

[ICE03](#)

[ICE06](#)

[ICE15](#)

[ICE32](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MoveFile Table

This table contains a list of files to be moved or copied from a specified source directory to a specified destination directory.

The MoveFile table has the following columns.

Column	Type	Key	Nullable
FileKey	Identifier	Y	N
Component_	Identifier	N	N
SourceName	Text	N	Y
DestName	Filename	N	Y
SourceFolder	Identifier	N	Y
DestFolder	Identifier	N	N
Options	Integer	N	N

## Columns

### FileKey

Primary key that uniquely identifies a particular MoveFile record.

### Component\_

External key into the [Component table](#). If the component referenced by this key is not selected for installation or removal, then no action is taken on this MoveFile entry.

### SourceName

This column contains the localizable name of the source files to be moved or copied. This column may be left blank. See the description of the SourceFolder column. This field must contain a long file name and may contain wildcard characters (\* and ?).

### DestName

This column contains the localizable name to be given to the original file after it is moved or copied. If this field is blank, then the

destination file is given the same name as the source file.

#### SourceFolder

This column contains the name of a property having a value that resolves to the full path to the source directory. If the SourceName column is left blank, then the property named in the SourceFolder column is assumed to contain the full path to the source file itself (including the file name).

#### DestFolder

The name of a property whose value resolves to the full path to the destination directory.

#### Options

Integer value specifying the operating mode.

Constant	Hexadecimal	Decimal	Meaning
(none)	0x000	0	Copy the source file.
msidbMoveFileOptionsMove	0x001	1	Move the source file.

## Remarks

If a "\*" wildcard is entered in the SourceName column of the MoveFile table and a destination file name is specified in the DestName column, all moved or copied files retain the names in the sources.

This table is processed by the [MoveFiles action](#).

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE18](#)  
[ICE32](#)  
[ICE45](#)

ICE85

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiAssembly Table

The MsiAssembly Table specifies Windows Installer settings for Microsoft .NET Framework assemblies and Win32 assemblies. For more information, see [Installation of Assemblies to the Global Assembly Cache](#) and [Installation of Win32 Assemblies](#).

On Windows XP, Windows Installer can install Win32 assemblies as [side-by-side assemblies](#). For more information, see the Side-by-Side Assembly API.

**Windows 2000:** This feature is not supported.

The MsiAssembly Table has the following columns.

Column	Type	Key	Nullable
Component_	Identifier	Y	N
Feature_	Identifier	N	N
File_Manifest	Identifier	N	Y
File_Application	Identifier	N	Y
Attributes	Integer	N	Y

## Columns

### Component\_

The key into the [Component Table](#) that specifies the Windows Installer component that contains this assembly.

The value in this field must not be set to null. The component KeyPath field in the Component Table must not be null.

For Win32 assemblies, the component KeyPath cannot be the manifest file that is specified in File\_Manifest. The manifest can be the keypath for a .NET Framework or policy assembly.

### Feature\_

Key into the [Feature Table](#).

When the assembly must be installed by a feature installation, Windows Installer installs the feature pointed to by this field.

#### File\_Manifest

An external key into the [File Table](#) that specifies the file that contains the manifest for a .NET Framework assembly or Win32 assembly.

For a Win32 assembly, do not specify this file as the component key path file in the KeyPath field of the [Component Table](#).

#### File\_Application

To install the assembly at a private location, enter the key path file for the assembly component in this field.

This is the value that appears in the KeyPath field of the [Component Table](#). The Installer can then install the assembly to the directory structure of the component that is specified in the [Directory Table](#). This field must be null if the assembly is to be installed into the global assembly cache.

#### Attributes

Enter a value of 1 (one) for a Win32 assembly. Enter a value of 0 (zero) for a .NET Framework assembly.

If the Attributes column is NULL, the Installer treats the assembly as a .NET Framework assembly.

## Remarks

If there is at least one entry in the MsiAssembly Table, the [InstallExecuteSequence Table](#) must contain the [MsiPublishAssemblies Action](#), and [MsiUnpublishAssemblies Action](#).

Because assemblies cannot be rolled back after they are committed, Windows Installer uses a two-step installation process. The interfaces to the assemblies are created during the installation operations that are generated by the MsiPublishAssemblies Action.

The assemblies are not committed until successful execution of the [InstallFinalize Action](#). This means that if you author a custom action or resource that relies on the assembly, it must be sequenced after the [InstallFinalize Action](#). For example, if you need to start a service that

depends on an assembly in the Global Assembly Cache (GAC), you must schedule the starting of that service after the [InstallFinalize Action](#). This means you cannot use the [ServiceControl Table](#) to start the service, instead you must use a custom action that is sequenced after InstallFinalize.

## Validation

[ICE03](#)

[ICE06](#)

[ICE32](#)

[ICE66](#)

[ICE83](#)

[ICE94](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiAssemblyName Table

The [MsiAssembly Table](#) and MsiAssemblyName Table specify Windows Installer settings for common language runtime assemblies and Win32 assemblies. For information see, [Installation of Assemblies to the Global Assembly Cache](#) and [Installation of Win32 Assemblies](#).

The MsiAssemblyName Table specifies the schema for the elements of a strong assembly cache name for a .NET Framework or Win32 assembly. The name is constructed by appending all elements with the same Component\_ key. See the following example.

Windows Installer can install Win32 assemblies as [side-by-side assemblies](#). For more information, see the Side-by-Side Assembly API.

The MsiAssemblyName Table has the following columns.

Column	Type	Key	Nullable
Component_	Identifier	Y	N
Name	Text	Y	N
Value	Text	N	N

## Columns

### Component\_

Key into the [Component Table](#) that specifies the Windows Installer component that contains this assembly.

### Name

Name of the attribute associated with the value specified in the Value column.

### Value

Value associated with the name specified in the Name column.

## Remarks

The information authored into the MsiAssemblyName Table must match the information in the manifest file of the assembly. If the information in the manifest and MsiAssemblyName Table do not match, removal of the application can leave the assembly on the computer.

For Win32 assemblies there must be a row in the MsiAssemblyName Table for each of the following entries in the Name field: type, name, version, language, publicKeyToken and processorArchitecture. The corresponding value for each name can be entered into the Value field. The name-value pairs in MsiAssemblyName Table must match the type, name, version, language, publicKeyToken and processorArchitecture attributes in the manifest of the assembly.

For private common language runtime assemblies (.NET Framework versions 1.0 and 1.1), the MsiAssemblyName Table must include a row for each of the following entries in the Name field: Name, Version, and Culture. The corresponding value for each Name can be entered into the Value field.

For global common language runtime assemblies (.NET Framework versions 1.0 and 1.1), the MsiAssemblyName Table must include a row for each of the following entries in the Name field: Name, Version, Culture, and PublicKeyToken. The corresponding value for each Name can be entered into the Value field.

The .NET Framework version 1.1 is the minimum version that can be used to perform an in-place update of a global common language runtime assembly. You can check the [MsiNetAssemblySupport](#) property for the version. The MsiAssemblyName Table must also have a FileVersion field because this type of assembly update only changes the FileVersion. For more information, see [Updating Assemblies](#).

For example, the assembly manifest for ComponentA might have an assemblyIdentity section as follows for a Win32 assembly.

```
<assemblyIdentity type="win32" name="ms-sxstest-simple" vers
```

In this case, populate the MsiAssemblyName Table as follows.

Component	Name	Value
ComponentA	type	win32

ComponentA	name	ms-sxstest-simple
ComponentA	version	1.0.0.0
ComponentA	language	en
ComponentA	publicKeyToken	1111111111222222
ComponentA	processorArchitecture	x86

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE32](#)  
[ICE66](#)  
[ICE83](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiDigitalCertificate Table

The MsiDigitalCertificate table stores certificates in binary stream format and associates each certificate with a primary key. The primary key is used to share certificates among multiple digitally signed objects. A digital certificate is a credential that provides a means to verify identity. For more information, see [Digital Certificates in the Cryptography section of the Microsoft Windows Software Development Kit \(SDK\)](#).

The [MsiDigitalSignature](#) and MsiDigitalCertificate tables are available starting with Windows Installer version 2.0.

Windows Installer can use digital signatures as a means to detect corrupted resources. Windows Installer version 2.0 can only verify the digital signatures of external cabinets, and only by the use of the MsiDigitalSignature and MsiDigitalCertificate tables.

Beginning with Windows Installer version 3.0, the Windows Installer can verify the digital signatures of patches (.msp files) by using the [MsiPatchCertificate](#) and MsiDigitalCertificate tables. For more information, see [Guidelines for Authoring Secure Installations](#) and [User Account Control \(UAC\) Patching](#).

The MsiDigitalCertificate table has the following columns.

Column	Type	Key	Nullable
DigitalCertificate	<a href="#">Identifier</a>	Y	N
CertData	<a href="#">Binary</a>	N	N

## Columns

### DigitalCertificate

Identifies the digital signature certificate. Primary key of table.

### CertData

The binary representation of the digital certificate. The CertData column contains the encoded byte array of a certificate context. This is the **pbCertEncoded** member of the **CERT\_CONTEXT** structure.

The certificate context can be obtained by calling **WinVerifyTrust**, **MsiGetFileSignatureInformation**, or by importing a .cer file.

## Validation

[ICE03](#)

[ICE06](#)

[ICE29](#)

[ICE32](#)

[ICE66](#)

[ICE81](#)

## See Also

[MsiGetFileSignatureInformation](#)

[MsiDigitalSignature](#) table

[Digital Signatures and Windows Installer](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiDigitalSignature Table

The MsiDigitalSignature table contains the signature information for every digitally signed object in the installation database.

The MsiDigitalSignature and [MsiDigitalCertificate](#) tables are available starting with Windows Installer version 2.0.

Windows Installer version can use digital signatures as a means to detect corrupted resources. Windows Installer 2.0 can only verify the digital signatures of external cabinets, and only by the use of the MsiDigitalSignature and MsiDigitalCertificate tables.

Beginning with Windows Installer 3.0, the Windows Installer can verify the digital signatures of patches (.msp files) by using the [MsiPatchCertificate](#) and MsiDigitalCertificate tables. For more information, see [Guidelines for Authoring Secure Installations](#) and [User Account Control \(UAC\) Patching](#).

The MsiDigitalSignature table has the following columns.

Column	Type	Key	Nullable
Table	<a href="#">Identifier</a>	Y	N
SignObject	<a href="#">Text</a>	Y	N
DigitalCertificate_	<a href="#">Identifier</a>	N	N
Hash	<a href="#">Binary</a>	N	Y

## Columns

### Table

With the Windows Installer version 2.0, the entry in this field must be "Media" for the [Media table](#). The installer only verifies the digital signatures on external cabinet media entries. This column and the SignObject column together specify the resource that is digitally signed.

### SignObject

A foreign key into the primary key of the table specified by the Table column. This column and the Table column together specify the resource that is digitally signed.

#### DigitalCertificate\_

A foreign key into the [MsiDigitalCertificate table](#). This identifies the certificate that must exist on the file for the associated action to succeed. The resource (or object) is always required to match this certificate in the MsiDigitalCertificate table.

#### Hash

In this field enter the reference hash of the resource (or object) that is to be checked against the actual hash of the resource (or object) obtained at run-time. If only the certificate needs to be verified, the Hash field may be null. Note that the format of the hash depends on the type of the resource (or object) being signed.

The Hash column contains the binary representation of the hash. The actual content is the **pbData** member of the **CRYPT\_HASH\_BLOB** structure, which is part of the **CRYPTOAPI\_BLOB** structure. This may be obtained by calling WinVerifyTrust or [MsiGetFileSignatureInformation](#).

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE29](#)  
[ICE32](#)  
[ICE66](#)  
[ICE81](#)

## See Also

### [MsiGetFileSignatureInformation](#)

[MsiDigitalCertificate table](#)

[Digital Signatures and Windows Installer](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiEmbeddedChainer Table

Use this table to author a [multiple-package installation](#). Each row in the MsiEmbeddedChainer table references a different user-defined function that can be used to install multiple Windows Installer packages from a single package. The [executable files](#) for the user-defined functions are stored inside the Windows Installer package.

**Windows Installer 4.0 or earlier:** Not supported. This table is available beginning with Windows Installer 4.5.

To install multiple packages from a single package, one of the user-defined functions listed in the MsiEmbeddedChainer table must have a conditional statement in the Condition field that evaluates to run the action. If more than one function has a condition that evaluates to run, only one function can run. This case is an error, and it cannot be guaranteed which function will run. If other custom actions are needed by the installation, these should be authored into the [CustomAction table](#) and sequence tables.

The functions must join the current installation by calling the [MsiJoinTransaction](#) function and must call the [MsiEndTransaction](#) function to commit the installation of multiple packages. If the functions return before calling [MsiEndTransaction](#), the installer rolls back all installations.

The MsiEmbeddedChainer table has the following columns.

Column	Type	Key	Nullable
MsiEmbeddedChainer	<a href="#">Identifier</a>	Y	N
Condition	<a href="#">Condition</a>	N	Y
CommandLine	<a href="#">Formatted</a>	N	Y
Source	<a href="#">CustomSource</a>	N	N
Type	<a href="#">Integer</a>	N	N

## Columns

## MsiEmbeddedChainer

The primary key for the table. This value is a unique identifier for the user-defined function described by this row.

## Condition

A conditional statement for running the user-defined function. You can enable or disable the functions listed in the MsiEmbeddedChainer table using a transform that modifies property values evaluated by this field. For more information, see [Using Properties in Conditional Statements](#).

## CommandLine

The value in this field is a part of the command line string passed to the executable file identified in the Source column. The installer appends the value in this field to the transaction handle to generate the command line. If the value in this column is null, the command line consists of only the transaction handle.

## Source

The location of the executable file for the user-defined function. If the value in the Type column is 2, this column can contain an external key into the [Binary table](#). If the value in the Type column is 18, this column can contain an external key into the [File table](#). If the value in the Type column is 50, this column can contain an external key into the [Property table](#).

## Type

The functions listed in the MsiEmbeddedChainer table are described using the following custom action numeric types. This column can contain the values for the following three numeric types only; any other combination of custom action flags is ignored.

<b>Custom action type</b>	<b>Custom action flags</b>	<b>Hexadecimal</b>	<b>Decimal</b>
<a href="#">Custom Action Type 2</a>	msidbCustomActionTypeExe + msidbCustomActionTypeBinaryData	0x002	2
<a href="#">Custom Action</a>	msidbCustomActionTypeExe + msidbCustomActionTypeSourceFile	0x012	18

Type 18			
Custom Action Type 50	msidbCustomActionTypeExe + msidbCustomActionTypeProperty	0x032	50

## Remarks

The Windows Installer does not prevent the user-defined functions in this table from running during the advertisement of the application. You can use a conditional statement in the Condition column to prevent a function from being run during advertisement.

The Windows Installer also provides a non-embedded external UI handler to build a rich user interface on top of the Windows Installer package. For more information about using an external UI handler with the Windows Installer, see [Monitoring an Installation Using MsiSetExternalUI](#).

The [MsiPackageCertificate Table](#) lists digital signature certificates used to verify the identity of the installation packages that make a multiple-package installation. You can use this table to reduce the number of times your multiple-package installation displays a *User Account Control* (UAC) prompt that requires a response by an administrator.

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiEmbeddedUI Table

The MsiEmbeddedUI table defines a user interface embedded in the Windows Installer package.

**Windows Installer 4.0 or earlier:** Not supported. This table is available beginning with Windows Installer 4.5.

The MsiEmbeddedUI table has the following columns.

Column	Type	Key	Nullable
MsiEmbeddedUI	Identifier	Y	N
FileName	Text	N	N
Attributes	Integer	N	N
MessageFilter	DoubleInteger	N	Y
Data	Binary	N	N

## Columns

### MsiEmbeddedUI

The primary key for the table.

### FileName

The name of the file that receives the binary information in the Data column. The name of the file is required to include an extension. For example, the name *embeddedui.dll* is acceptable, but *embeddedui* is unacceptable. The name may be localized. This field can contain a short file name or a long file name, but it cannot contain both. The format of this field is like the [Filename](#) column data type except that the vertical bar (|) separator for the short file name/long file name syntax is not available. Because some web servers can be case sensitive, FileName should match the case of the source files exactly to ensure support of Internet downloads.

### Attributes

Information about the data in the Data column. The value in this field

can contain one or more of the following constants.

<b>Constant</b>	<b>Hexadecimal</b>	<b>Decimal</b>	<b>Meaning</b>
None	0x00	0	The file is not the DLL file for the user interface. It may be a resource file used by the user interface.
msidbEmbeddedUI	0x01	1	The primary DLL file for the user interface. No more than one row in the table can be marked with this attribute. If multiple rows are marked with this attribute, it is an error and it cannot be guaranteed which DLL is used.
msidbEmbeddedHandlesBasic	0x02	2	Enables the installer to invoke the embedded UI during a basic UI level installation. The installer ignores this attribute if it is not combined with the msidbEmbeddedUI attribute.

### MessageFilter

Specifies the types of messages that are sent to the user interface

DLL. This column is only relevant for rows with the msidbEmbeddedUI attribute. This field should be null if a row references a resource file and the value of Attributes is null. If a row references a user interface DLL, the value in this column should not be null.

The value in this column can be a combination of the following values. The installer ignores any other values.

<b>Constant</b>	<b>Hexadecimal</b>	<b>Decimal</b>
INSTALLLOGMODE_FATALEXIT	0x00001	1
INSTALLLOGMODE_ERROR	0x00002	2
INSTALLLOGMODE_WARNING	0x00004	4
INSTALLLOGMODE_USER	0x00008	8
INSTALLLOGMODE_INFO	0x00010	16
INSTALLLOGMODE_FILESINUSE	0x00020	32
INSTALLLOGMODE_RESOLVESOURCE	0x00040	64
INSTALLLOGMODE_OUTOFDISKSPACE	0x00080	128
INSTALLLOGMODE_ACTIONSTART	0x00100	256
INSTALLLOGMODE_ACTIONDATA	0x00200	512
INSTALLLOGMODE_PROGRESS	0x00400	1024

INSTALLLOGMODE_COMMONDATA	0x00800	2048
INSTALLLOGMODE_INITIALIZE	0x01000	4096
INSTALLLOGMODE_TERMINATE	0x02000	8192
INSTALLLOGMODE_SHOWDIALOG	0x04000	16384
INSTALLLOGMODE_RMFILESINUSE	0x02000000	33554432
INSTALLLOGMODE_INSTALLSTART	0x04000000	67108864
INSTALLLOGMODE_INSTALLEND	0x08000000	134217728

--	--	--

## Data

This column contains binary information. If the Attribute field is marked with the msidbEmbeddedUI attribute, the information in this field must be a DLL. If the Attribute field is not the msidbEmbeddedUI attribute, the information in this field can be a resource file in any format.

## Remarks

To use an embedded user interface, the setup developer must author this functionality into the Windows Installer package. The MsiEmbeddedUI table defines the embedded user interface. The DLL for the embedded UI should export the [InitializeEmbeddedUI](#), [EmbeddedUIHandler](#), and [ShutdownEmbeddedUI](#) functions. Packages that do not support an embedded user interface can use the Windows Installer internal user interface.

To run [Debugging Tools for Windows](#) on an embedded user interface, use the techniques described in [Debugging Custom Actions](#). Set the value of MsiBreak to MsiEmbeddedUI.

For an example of an embedded custom UI see [Using an Embedded UI](#).

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiFileHash Table

The **MsiFileHash** table is used to store a 128-bit hash of a source file provided by the Windows Installer package. The hash is split into four 32-bit values and stored in separate columns of the table.

Windows Installer can use file hashing as a means to detect and eliminate unnecessary file copying. A file hash stored in the **MsiFileHash** table may be compared to a hash of an existing file on the user's computer obtained by calling **MsiGetFileHash**. The **MsiFileHash** table can only be used with unversioned files.

The **MsiFileHash** table has the following columns.

Column	Type	Key	Nullable
File_	Identifier	Y	N
Options	Integer	N	N
HashPart1	DoubleInteger	N	N
HashPart2	DoubleInteger	N	N
HashPart3	DoubleInteger	N	N
Hashpart4	DoubleInteger	N	N

## Columns

File\_

Foreign key to [File table](#). 72 char string.

Options

This column must be 0 and is reserved for future use.

HashPart1

First 32 bits of hash. The file hash entered in this field must be obtained by calling **MsiGetFileHash** or the **FileHash method**. Do not use other methods.

HashPart2

Second 32 bits of hash. The file hash entered in this field must be

obtained by calling **MsiGetFileHash** or the **FileHash method**. Do not use other hashing methods.

#### HashPart3

Third 32 bits of hash. The file hash entered in this field must be obtained by calling **MsiGetFileHash** or the **FileHash method**. Do not use other methods.

#### HashPart4

Fourth 32 bits of hash. The file hash entered in this field must be obtained by calling **MsiGetFileHash** or the **FileHash method**. Do not use other methods.

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE32](#)  
[ICE60](#)  
[ICE66](#)

## See Also

[MsiGetFileHash](#)

[Default File Versioning](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiLockPermissionsEx Table

The MsiLockPermissionsEx Table can be used to secure services, files, registry keys, and created folders.

A package should not contain both the MsiLockPermissionsEx Table and the [LockPermissions Table](#).

**Windows Installer 4.5 or earlier:** Not supported. This table is recommended for packages intended for installation with Windows Installer 5.0 or later.

The MsiLockPermissionsEx Table has the following columns.

Column	Type	Key	Nullable
MsiLockPermissionsEx	Text	Y	N
LockObject	Identifier	N	N
Table	Text	N	N
SDDLText	FormattedSDDLText	N	N
Condition	Condition	N	Y

## Columns

### MsiLockPermissionsEx

This is the primary key of this table.

### LockObject

This column and the Table column together specify the file, directory, registry key, or service that is to be secured. The LockObject column is a foreign key that points to the primary key of the table specified by the Table column.

### Table

This column and the LockObject column specify the file, directory, registry key, or service that is to be secured. In the Table column, enter File, Registry, CreateFolder, or ServiceInstall to specify a

LockObject listed in the [File Table](#), [Registry Table](#), [CreateFolder Table](#), or [ServiceInstall Table](#).

#### SDDLText

Enter the SDDL string to indicate permissions to apply to selected object. The SDDL must be provided in Security Descriptor String Format.

#### Condition

This column contains a conditional expression used to determine whether to apply the specified permission. If the condition evaluates to FALSE, the permission is not applied. If the condition evaluates to TRUE, the permission is applied.

## Remarks

See [Securing Resources](#) for more information about securing services, files, registry keys, and created folders.

Use the MsiLockPermissionsEx Table to secure objects for a user account that is being created during the installation. The user account must already exist when the installation secures the object. Create the user account before installing the file, registry key, folder or service being secured.

If a LockObject and Table pair in this table has more than one conditional expression that evaluates to true, the installation fails and Windows Installer returns an error message 1942.

If the FormattedSDDLText string in the SDDLText field cannot be resolved into a valid SDDL string, the installation fails and Windows Installer returns an error message 1943.

If the user does not have sufficient privileges to set the security descriptor specified by the SDDLText field on a file or folder, the installation fails and Windows Installer returns an error message 1926.

If the user does not have sufficient privileges to set the security descriptor specified by the SDDLText field on a registry key, the installation fails and Windows Installer returns an error message 1401.

If the user does not have sufficient privileges to set the security descriptor

specified by the SDDLText field on a service, the installation fails and Windows Installer returns an error message 1944.

## **Validation**

ICE104

ICE03

ICE06

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiPackageCertificate Table

The MsiPackageCertificate Table lists digital signature certificates used to verify the identity of the installation packages that make this [Multiple-Package Installation](#).

Use this table to author a multiple-package installation for a product containing multiple Windows Installer packages. If the first package is digitally signed, and contains a MsiPackageCertificate Table specifying digital certificates for all the remaining packages in the product, the administrator needs only accept the [User Account Control](#) (UAC) prompt displayed for the first package. After accepting the UAC's prompt for the first package, the user-defined functions in the [MsiEmbeddedChainer table](#) can then join the remaining packages to the multiple-package installation without displaying a UAC prompt and requiring an administrator response for each package.

If one or more of the functions in the MsiEmbeddedChainer table request an unsigned package, another UAC prompt requiring administrator interaction is displayed for each unsigned package. If the administrator accepts this UAC prompt, the multi-package installation continues. Once an administrator has provided credentials for a package, no UAC prompt will again be displayed for that package during this multi-package installation. If the administrator rejects a UAC prompt for a package, the Windows installer rolls back the multi-package installation before it commits to install any packages belonging to the product.

**Windows Installer 4.0 or earlier:** Not supported. This table is available beginning with Windows Installer 4.5.

The MsiPackageCertificate Table has the following columns:

Column	Type	Key	Nullable
PackageCertificate	Identifier	Y	N
DigitalCertificate_	Identifier	N	N

## Columns

## PackageCertificate

The unique identifier for this row in the MsiPackageCertificate Table.

## DigitalCertificate

An external key into the first column of the [MsiDigitalCertificate Table](#). The row indicated in the MsiDigitalCertificate Table contains the binary representation of the signer certificate.

## Validation

[ICE39](#)

[ICE81](#)

## See Also

[MsiEmbeddedChainer](#)

[MsiDigitalCertificate Table](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiPatchCertificate Table

The MsiPatchCertificate Table identifies the possible signer certificates used to digitally sign patches. The MsiPatchCertificate Table contains the information that is needed to enable [User Account Control \(UAC\) Patching](#) for an application.

The MsiPatchCertificate Table has the following columns:

Column	Type	Key	Nullable
PatchCertificate	Identifier	Y	N
DigitalCertificate_	Identifier	N	N

## Columns

### PatchCertificate

The unique identifier for this row in the MsiPatchCertificate Table.

### DigitalCertificate

An external key into the first column of the [MsiDigitalCertificate Table](#). The row indicated in the MsiDigitalCertificate Table contains the binary representation of the signer certificate.

## Remarks

Patches are always evaluated against the MsiPatchCertificate Table that is current at the time the patch is applied. A patch can modify the MsiPatchCertificate Table by adding or removing entries. This enables a patch to modify the evaluation of future patches that are applied later in the patching sequence. Multiple certificates can exist in the table, and the patch must match at least one certificate to be applied.

## Validation

ICE03

[ICE06](#)  
[ICE32](#)  
[ICE81](#)

## **See Also**

[DisableLUAPatching](#)

[User Account Control \(UAC\) Patching](#)

**[MSIDISABLELUAPATCHING](#)**

[Digital Signatures and Windows Installer](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiPatchHeaders Table

The MsiPatchHeaders table holds the binary patch header streams used for patch validation.

The MsiPatchHeaders table is used when long [File table](#) keys result in a failure to generate the patch header stream in the [Patch table](#). This can be due to the stream name limitation described in [OLE Limitations on Streams](#). In this case, the Patch table can reference the MsiPatchHeaders table to create the patch header stream.

The MsiPatchHeaders table has the following columns.

Column	Type	Key	Nullable
StreamRef	<a href="#">Identifier</a>	Y	N
Header	<a href="#">Binary</a>	N	N

## Columns

### StreamRef

The primary key for the table that uniquely identifies a particular patch header.

### Header

This column is the binary stream patch header used for patch validation.

## Remarks

This table is processed by the [PatchFiles action](#). This table is usually added to the install package by a transform from a patch package. It is usually not authored directly into an installation package.

## Validation

[ICE03](#)

ICE06  
ICE29

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiPatchMetadata Table

The MsiPatchMetadata Table contains information about a Windows Installer patch that is required to remove the patch and that is used by **Add/Remove Programs**.

Patches installed without this table present in the patch database (.msp file) cannot be removed, and are missing some information from **Add/Remove Programs**. The table must be in the database of the patch file and not in a transform in the patch.

The MsiPatchMetadata Table has the following columns.

Column	Type	Key	Nullable
Company	Identifier	Y	Y
Property	Identifier	Y	N
Value	Text	N	N

## Columns

### Company

The name of the company. An empty field (a Null value) indicates that the row contains one of the standard metadata properties of the Windows Installer. For more information, see the Remarks section of this topic.

By adding a row to the table and entering a company name in this field, you can add any company to extend the property set.

### Property

The name of a metadata property.

### Value

The value of the metadata property. This can never be Null or an empty string.

## Remarks

Available in Windows Installer 3.0 and later.

Rows in the MsiPatchMetadata Table that contain a Null value in the CompanyName field refer to one of the following standard Windows Installer metadata properties.

Property	Description
AllowRemoval	Indicates whether or not the patch is an <b>Uninstallable Patch</b> . If the value field contains 0 (zero), the patch cannot be removed. If the value field contains one (1), the patch is an Uninstallable Patch. This property is registered and its value can be obtained by using the <b>MsiGetPatchInfoEx</b> function.
ManufacturerName	Name of the manufacturer of the application.
MinorUpdateTargetRTM	Indicates that the patch targets the RTM version of the product or the most recent major upgrade patch. Author this optional property in minor upgrade patches that contain sequencing information to indicate that the patch removes of all patches up to the RTM version of the product, or up to the most recent major upgrade patch. This property is available in Windows Installer 3.1 and later.
TargetProductName	Name of the application or target application suite.
MoreInfoURL	A URL that provides information specific to this patch. This property is registered and its value can be obtained by using the <b>MsiGetPatchInfoEx</b> function. Beginning with Windows XP with Service Pack 2 (SP2), this value can be the support link for the patch displayed in <b>Add/Remove Programs</b> .

CreationTimeUTC	Creation time of the .msp file in the form of mm-dd-yy HH:MM (month-day-year hour:minute).
DisplayName	A title for the patch that is okay for public display. This property is registered, and its value can be obtained by using the <b>MsiGetPatchInfoEx</b> function. Beginning with Windows XP with SP2, this value is the name of the patch that is displayed in <b>Add/Remove Programs</b> .
Description	Brief description of the patch.
Classification	A string value that contains the arbitrary category of updates as defined by the author of the patch. For example, patch authors can specify that each patch be classified as a Hotfix, Security Rollup, Critical Update, Update, Service Pack, or Update Rollup. This property is required.
OptimizeCA	Indicates whether the Windows Installer should skip custom actions when applying the patch. This can reduce the time required to apply the patch.  The OptimizeCA property can have one of the following values:

- 0 - Do not skip any custom actions.
- 1 - Skip property and directory assignment custom actions. [Custom Action Type 35](#) and [Custom Action Type 51](#) can be property and directory assignment custom actions.
- 2 - Skip immediate custom actions that do not fall into the property or directory assignments. The immediate custom actions do not include msidbCustomActionTypeInScript option in the Type column of the [CustomAction Table](#).
- 4 - Skip custom actions that run within the script.

The value of OptimizeCA must be the same for all patches that are being installed or no custom actions are skipped. For example, if two patches

are being installed, and OptimizeCA is set to the values 1 and 2 respectively, no custom actions are skipped.

The values of OptimizeCA can be combined when processing multiple new patches. If all patches have a 1 (one) included in the values, then all property and directory assignment custom actions are skipped. If one patch has the value 3 (three) for the property, and one patch has the value 1 (one) for the property, the property and directory assignment custom actions are skipped. However, the other immediate custom actions run, because not all of the patches requested are skipped.

**OptimizedInstallMode** If this property is set to 1 (one) in all the patches to be applied in a transaction, an application of the patch is optimized if possible. For more information, see [Patch Optimization](#). Available beginning with Windows Installer 3.1.

## Validation

[ICE03](#)

[ICE06](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiPatchOldAssemblyFile Table

The MsiPatchOldAssemblyFile table relates a file in the [File table](#) to an assembly name in the [MsiPatchOldAssemblyName table](#). Multiple old assembly names can be associated with a single file.

The MsiPatchOldAssemblyFile table has the following columns.

Column	Type	Key	Nullable
File_	Identifier	Y	N
Assembly_	Identifier	Y	N

## Columns

File\_

Foreign key to the File table that specifies the assembly to be patched. This column is part of the primary key.

Assembly\_

Foreign key to the MsiPatchOldAssemblyName table that identifies one of the old assembly names for the assembly. This column is part of the primary key.

## Remarks

Windows Installer uses the MsiPatchOldAssemblyFile table and MsiPatchOldAssemblyName table when patching assemblies installed to the Global Assembly Cache (GAC). When releasing a newer version of an assembly, the strong name of the assembly is changed. The two tables together identify the old assembly name for an updated assembly. This allows the Installer to use the old assembly name to find the original file in the GAC and apply a binary patch. Without this information, the installer may have to access the original installation source in order to patch an assembly installed in the GAC.

The MsiPatchOldAssemblyFile table and MsiPatchOldAssemblyName

table are not generated automatically by [PatchWiz](#). The update package specified in the [UpgradedImages table](#) is required to contain these tables for the patch to have this information.

## Validation

[ICE03](#)

[ICE06](#)

[ICE32](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiPatchOldAssemblyName Table

The MsiPatchOldAssemblyName table specifies the old name for an assembly.

The MsiPatchOldAssemblyName table has the following columns.

Column	Type	Key	Nullable
Assembly	Identifier	Y	N
Name	Text	Y	N
Value	Text	N	N

## Columns

### Assembly

Unique identifier for the old assembly name. This key is used as a mapping between this and the [MsiPatchOldAssemblyFile table](#).

### Name

Name of the attribute associated with the value specified in the Value column.

### Value

Value associated with the name specified in the Name column.

## Remarks

Windows Installer uses the MsiPatchOldAssemblyFile table and MsiPatchOldAssemblyName table when patching assemblies installed to the Global Assembly Cache (GAC). When releasing a newer version of an assembly, the strong name of the assembly is changed. The two tables together identify the old assembly name for an updated assembly. This allows the Installer to use the old assembly name to find the original file in the GAC and apply a binary patch. Without this information, the installer may have to access the original installation source in order to

patch an assembly installed in the GAC.

The MsiPatchOldAssemblyFile table and MsiPatchOldAssemblyName table are not generated automatically by [PatchWiz](#). The update package specified in the [UpgradedImages table](#) is required to contain these tables for the patch to have this information.

## Validation

[ICE03](#)

[ICE06](#)

[ICE32](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiPatchSequence Table

The MsiPatchSequence table contains all the information the installer requires to determine the sequence of application of a [small update](#) patch relative to all other patches. The table must be in the database of the patch file and not in a transform in the patch. The installer ignores this table when applying a [major upgrade](#) patch. When applying a [minor upgrade](#) patch, the installer only uses this table to identify superseded patches that must not be sequenced.

The **MsiPatchSequence table** has the following columns.

Column	Type	Key	Nullable
PatchFamily	Identifier	Y	N
ProductCode	GUID	Y	Y
Sequence	Version	N	N
Attributes	Integer	N	Y

## Columns

### PatchFamily

Specifies that the patch is a member of the patch family named in this field. Patches in the same patch family that target the same product version are sorted by the values in the Sequence column. The patches within the patch family are applied to the target product in the order of increasing sequence. The PatchFamily is also used to determine which patches are to be superseded. A patch may be listed in multiple rows and belong to multiple patch families if it applies to more than one product or includes multiple fixes.

The Windows Installer does not interpret the PatchFamily value in any way other than comparisons for equality against other PatchFamily values. A PatchFamily value must be unique within the ProductCode targeted by the set of patches. In the complex patching scenarios the PatchFamily identifier may need to be globally unique.

## ProductCode

A value in this field is optional. If a [product code](#) GUID is entered in this field and the patch is being applied to the specified product, the patch is sorted and applied as a member of the specified PatchFamily. If a product code GUID is entered in this field and the patch is not being applied to the product specified by ProductCode, this row is ignored. If the value in ProductCode is NULL, the patch is sorted and applied as a member of PatchFamily for all targets of the patch regardless of the product code.

A patch can have multiple rows in the same PatchFamily and a different ProductCode for each product targeted by the patch. One row for the PatchFamily can specify NULL for ProductCode. If the target product matches a row with a non-NULL ProductCode, the installer uses the matching row and ignores the row with the NULL ProductCode. If none of the specified product codes match the target, the patch is sorted and applied as a member of PatchFamily for all targets of the patch regardless of the product code.

## Sequence

The value in the Sequence column specifies the sequence of this patch within the specified PatchFamily. The value in Sequence is expressed in the format of [Version](#) data. The value contains between 1 and 4 fields and each field has a range of 0 to 65535. Members of PatchFamily are sorted and applied to the target product in the order of increasing Sequence values. For example, the following six values are increasing: 1, 1.1, 1.2, 2.01, 2.01.1, 2.01.1.1.

## Attributes

The presence of the `msidbPatchSequenceSupersedeEarlier` attribute in a row indicates that the small update patch supersedes the updates provided by all patches with lesser Sequence values in the same PatchFamily. This patch contains all fixes provided by earlier patches in the specified PatchFamily. This attribute does not mean that this patch supersedes the earlier patches in all cases because the earlier patches can belong to multiple patch families.

A small update patch cannot supersede a minor upgrade or major upgrade patch under any circumstances, even if the `msidbPatchSequenceSupersedeEarlier` is set.



Name	Value	Meaning
	0x00	Indicates a simple sequencing value.
msidbPatchSequenceSupersedeEarlier	0x01	Indicates a patch that supersedes earlier patches in this family.

## Validation

[ICE03](#)

[ICE06](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiServiceConfig Table

The MsiServiceConfig table configures a service that is installed or being installed by the current package.

**Windows Installer 4.5 or earlier:** Not supported. This table is available beginning with Windows Installer 5.0.

The MsiServiceConfig table has the following columns.

Column	Type	Key	Nullable
MsiServiceConfig	Identifier	Y	N
Name	Formatted	N	N
Event	Integer	N	N
ConfigType	Integer	N	N
Argument	Formatted	N	Y
Component_	Identifier	N	N

## Columns

### MsiServiceConfig

This is the primary key of this table.

### Name

This column contains the name of a service that is a part of this package or that is already installed.

### Event

This column specifies when to change the service configuration. The following values can be combined to represent multiple operations. Any values included other than these are ignored.

Constant	Description
msidbServiceConfigEventInstall 1	Takes the action during installation of the component.

msidbServiceConfigEventUninstall 2	Takes the action during uninstallation of the component.
msidbServiceConfigEventReinstall 4	Takes the action during reinstallation of the component.

## ConfigType

The value in this field, combined with the value in the Arguments field, specify what change to make to the service configuration. The specified change takes effect the next time the system is started.

Config	Description
SERVICE_CONFIG_DELAYED_AUTO_START 3	Configure the ti service.  Enter 1 in the A service after oth time delay.  Enter 0 in the A the auto-start se  Applies only to or services insta SERVICE_AUT StartType field c
SERVICE_CONFIG_REQUIRED_PRIVILEGES_INFO 6	Change the list o the service.  Enter a list of re Argument field. value in the Arg <b>Privilege Const</b> privileges. You o the <b>Formatted</b> st Separate the pri by [~].

<p>SERVICE_CONFIG_SERVICE_SID_INFO 5</p>	<p>Add a service SID token containing</p> <p>Enter in the Argument the SID type for the structure: SERVICE_SID_INFO (0x00), SERVICE_SID_NAME (0x03), or SERVICE_SID_NAME (0x01).</p>
<p>SERVICE_CONFIG_PRESHUTDOWN_INFO 7</p>	<p>Configure the length of the Service Control Manager delay before proceeding with operations. The Argument is the time of time after sending SERVICE_CONFIG_PRESHUTDOWN_INFO notification to the service.</p> <p>Enter the time delay in milliseconds, in the Argument field. The default delay is 30000 milliseconds.</p>
<p>SERVICE_CONFIG_FAILURE_ACTIONS_FLAG 4</p>	<p>Configure when the service has failed for this service. The Argument is the actions that the service has failed.</p> <p>Enter 0 to run the service terminate action. The structure is SERVICE_STATUS_FAILURE_ACTIONS.</p> <p>Enter 1 to run the service terminates report action. The structure is SERVICE_STATUS_FAILURE_ACTIONS and the dwWinlogMsg is ERROR_SUCCESS. The service terminates with the dwWinlogMsg is ERROR_SUCCESS.</p>

### Argument

The value in this field, combined with the value in the ConfigType field, specify what change to make to the service configuration. The specified change takes effect the next time the system is started.

### Component\_

External key to the Component column of the [Component Table](#).

## Validation

[ICE102](#)

[ICE03](#)

[ICE06](#)

[ICE32](#)

[ICE45](#)

[ICE46](#)

[ICE69](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiServiceConfigFailureActions Table

The MsiServiceConfigFailureActions table lists operations to be run after a service fails. The operations specified in this table run the next time the system is started.

**Windows Installer 4.5 or earlier:** Not supported. This table is available beginning with Windows Installer 5.0.

The MsiServiceConfigFailureActions table has the following columns.

Column	Type	Key	Nullable
MsiServiceConfigFailureActions	Identifier	Y	N
Name	Formatted	N	N
Event	Integer	N	N
ResetPeriod	Integer	N	Y
RebootMessage	Formatted	N	Y
Command	Formatted	N	Y
Actions	Text	N	Y
DelayActions	Text	N	Y
Component_	Identifier	N	N

## Columns

### MsiServiceConfigFailureActions

This is the primary key of this table that identifies a failure action.

### Name

This column contains the name of a service that is a part of this package or that is already installed.

### Event

This column specifies when to change the service's configuration. The following values are bit fields that can be combined to represent multiple operations. Any other bit field values are ignored.

Constant	Description
msidbServiceConfigEventInstall 1	Change during installation of the component.
msidbServiceConfigEventUninstall 2	Change during uninstallation of the component.
msidbServiceConfigEventReinstall 4	Change during re-installation of the component.

#### ResetPeriod

The reset period in seconds of service's failure count. The Service Control Manager (SCM) counts the number of times each service has failed since the system was last restarted. The count is reset to zero if the service does not fail for the reset period. When the service fails for the Nth time, the system performs the action specified in the element [N-1] of the array specified in the Actions field.

Leave ResetPeriod field empty to indicate that failure count should never be reset.

#### RebootMessage

The message sent to users before restarting the computer in response to a SC\_ACTION\_REBOOT action specified in the Actions column. You can use an empty string, "", to send the current message unchanged. You can use [~] syntax of the [Formatted](#) datatype to delete the current message and send no message.

#### Command

The command line run by the process created by the **CreateProcess** function in response to a SC\_ACTION\_RUN\_COMMAND action specified in the Actions column. The new process runs under the same account as the service and only if the Action field is SC\_ACTION\_RUN\_COMMAND. You can use an empty string, "", to

use the current command line unchanged. You can use [~] syntax of the [Formatted](#) datatype to delete the current command line and run no operation when the service fails.

### Actions

This field contains an array of integer values that specify the actions taken by the SCM if the service fails. Separate the values in the array by [~]. The integer value in the Nth element of the array specifies the action performed when the service fails for the Nth time. Each member of the array is one of following integer values.

Constant	Description
SC_ACTION_NONE 0	No action.
SC_ACTION_REBOOT 2	Restart the computer.
SC_ACTION_RESTART 1	Restart the service.
SC_ACTION_RUN_COMMAND 3	Run a command.

### DelayActions

This field contains an array of integer values that specify the time in milliseconds to wait before performing the action specified in the Action column. Separate the values in the array by [~]. The number of elements in the DelayActions array must be equal to the number of elements in the Actions array. The Nth element of the DelayActions array specifies the time delay for the nth element of the Actions array.

### Component\_

External key to column one of the [Component Table](#).

## Validation

ICE102  
ICE03  
ICE06  
ICE32  
ICE45  
ICE46  
ICE69

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiSFCEBypass Table

The MsiSFCEBypass Table contains a list of files that should bypass Windows File Protection when the files are installed on Microsoft Windows Me.

The MsiSFCEBypass Table has the following column.

Column	Type	Key	Nullable
File_	Identifier	Y	N

## Columns

File\_

The foreign key to the [File Table](#) that specifies the file that should bypass Windows File Protection when the file is installed on Windows Me.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# MsiShortcutProperty Table

The MsiShortcutProperty table enables Windows Installer to set properties for shortcuts that are also Windows Shell objects. Beginning with Windows Vista and Windows Server 2008 the Windows Shell provides an IPropertyStore Interface for shell objects such as shortcuts. A Windows Installer 5.0 package running on Windows Server 2008 R2 or Windows 7 can set these properties when the shortcut is installed.

**Windows Installer 4.5 or earlier:** Not supported. This table is available beginning with Windows Installer 5.0.

The MsiShortcutProperty table has the following columns.

Column	Type	Key	Nullable
MsiShortcutProperty	Identifier	Y	N
Shortcut_	Identifier	N	N
PropertyKey	Formatted	N	N
PropVariantValue	Formatted	N	N

## Columns

### MsiShortcutProperty

Unique identifier for this row of the MsiShortcutProperty table.

### Shortcut\_

A key into the [Shortcut](#) table that identifies the shortcut having a property set.

### PropertyKey

A string value that provides information for the **PROPERTYKEY** structure.

### PropVariantValue

A string value that provides information for the **PROPVARIANT** structure.

Multiple properties can be set on a shortcut. If the same property is set multiple times on the same shortcut the value is set in an unspecified order.

Windows Installer can set shortcut properties only when the shortcut is installed or reinstalled. A patch that does not reinstall a shortcut that has already been installed does not update the shortcut's properties. A patch can update the properties by including a Shortcut table in the patch package and reinstalling the shortcut.

## **Validation**

ICE03

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ODBCAttribute Table

The ODBCAttribute table contains information about the attributes of Open Database Connectivity (ODBC) drivers and translators.

The ODBCAttribute table has the following columns.

Column	Type	Key	Nullable
Driver_	Identifier	Y	N
Attribute	Text	Y	N
Value	Formatted	N	Y

## Columns

Driver\_

Internal token name for a driver. A primary key for the table. A foreign key into the [ODBCDriver table](#).

Attribute

Name of the driver attribute. A primary key for the table.

Value

Localizable string value for attribute.

## Remarks

The [InstallODBC](#) and [RemoveODBC](#) actions in *sequence tables* process the information in this table. For information about using sequence tables, see [Using a Sequence Table](#).

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE32](#)

ICE46

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ODBCDataSource Table

The ODBCDataSource table lists the data sources belonging to the installation.

The ODBCDataSource table has the following columns.

Column	Type	Key	Nullable
DataSource	Identifier	Y	N
Component_	Identifier	N	N
Description	Text	N	N
DriverDescription	Text	N	N
Registration	Integer	N	N

## Columns

### DataSource

Internal token name for data source. A primary key for the table.

### Component\_

External key into the [Component table](#).

### Description

The description registered for this data source. This value cannot be localized. ODBC data source descriptions cannot exceed SQL\_MAX\_DSN\_LENGTH.

### DriverDescription

Associated driver which may be pre-existing. This value cannot be localized.

### Registration

This field specifies the type of registration for the data source.

Constant	Hexadecimal	Decimal
msidbODBCDataSourceRegistrationPerMachine	0x000	0

msidbODBCDataSourceRegistrationPerUser	0x001	1

## Remarks

The [InstallODBC](#) and [RemoveODBC](#) actions in *sequence tables* process the information in this table. For information about using sequence tables, see [Using a Sequence Table](#).

## Validation

[ICE03](#)

[ICE06](#)

[ICE32](#)

[ICE45](#)

[ICE98](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ODBCDriver Table

The ODBCDriver table lists the ODBC drivers belonging to the installation.

The ODBCDriver table has the following columns.

Column	Type	Key	Nullable
Driver	Identifier	Y	N
Component_	Identifier	N	N
Description	Text	N	N
File_	Identifier	N	N
File_Setup	Identifier	N	Y

## Columns

### Driver

Internal token name for driver. A primary key for the table

### Component\_

External key into the Component table.

### Description

The description registered for this ODBC driver. This value cannot be localized.

### File\_

The DLL file for drivers listed in the Driver column. The File\_ column is an external key into the [File table](#). The filename entered in the Filename column of that File table record must be in the short filename format. The SFN|LFN syntax cannot be used.

### File\_Setup

The setup DLL file for the driver if it is different than from Driver. The File\_ column is an external key into the [File table](#). The filename entered in the Filename column of that File table record must be in the short filename format. The SFN|LFN syntax cannot be used.

## Remarks

The [InstallODBC](#) and [RemoveODBC](#) actions in *sequence tables* process the information in this table. For information about using sequence tables, see [Using a Sequence Table](#).

## Validation

[ICE03](#)

[ICE06](#)

[ICE32](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ODBCSourceAttribute Table

The ODBCSourceAttribute table contains information about the attributes of data sources.

The ODBCSourceAttribute table has the following columns.

Column	Type	Key	Nullable
DataSource_	Identifier	Y	N
Attribute	Text	Y	N
Value	Formatted	N	Y

## Columns

DataSource\_

Internal token name for data source. A primary key for the table.

Attribute

Name of data source attribute. A primary key for the table.

Value

The localizable string value for the attribute.

## Remarks

The [InstallODBC](#) and [RemoveODBC](#) actions in *sequence tables* process the information in this table. For information about using sequence tables, see [Using a Sequence Table](#).

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE32](#)  
[ICE46](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ODBCTranslator Table

The ODBCTranslator table lists the ODBC translators belonging to the installation.

The ODBCTranslator table has the following columns.

Column	Type	Key	Nullable
Translator	Identifier	Y	N
Component_	Identifier	N	N
Description	Text	N	N
File_	Identifier	N	N
File_Setup	Identifier	N	Y

## Columns

### Translator

Internal token name for translator. A primary key for the table.

### Component\_

External key into the Component table.

### Description

The description registered for this ODBC driver translator. This value cannot be localized.

### File\_

The DLL file for the transfer listed in the Translator column. The File\_ column is an external key into the [File table](#). The filename entered in the Filename column of that File table record must be in the short filename format. The SFN|LFN syntax cannot be used.

### File\_Setup

The setup DLL file for the translator if it is different from the Translator column. The File\_ column is an external key into the [File table](#). The filename entered in the Filename column of that File table record must be in the short filename format. The SFN|LFN syntax

cannot be used.

## Remarks

The [InstallODBC](#) and [RemoveODBC](#) actions in *sequence tables* process the information in this table. For information about using sequence tables, see [Using a Sequence Table](#).

## Validation

[ICE03](#)

[ICE06](#)

[ICE32](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patch Table

The Patch table specifies the file that is to receive a particular patch and the physical location of the patch files on the media images.

The Patch table has the following columns.

Column	Type	Key	Nullable
File_	Identifier	Y	N
Sequence	Integer	Y	N
PatchSize	DoubleInteger	N	N
Attributes	Integer	N	N
Header	Binary	N	Y
StreamRef_	Identifier	N	Y

## Columns

### File\_

The patch is applied to the file specified by the identifier in this column. This is a primary key for the table and it is a foreign key to the [File table](#).

### Sequence

This is the position of the patch file in the sequence order of files on the media images. The sequence order must correspond to the order of the files in the patch package cabinet file. This is a primary key for this table. The maximum limit is 32767 files, to create a Windows Installer package with more files, see [Authoring a Large Package](#).

### PatchSize

This column gives the size of the patch in bytes written as a long integer.

### Attributes

Integer containing bit flags representing patch attributes. Insert a

value of 1 in this column to indicate that the failure to apply this patch is not a fatal error.

Constant	Hexadecimal	Decimal	Description
(none)	0x000	0	Failure to apply this patch is a fatal error.
msidbPatchAttributesNonVital	0x001	1	Indicates that the failure to apply this patch is not a fatal error.

#### Header

This column is the binary stream patch header used for patch validation. This column should be null if the StreamRef\_ column is not null. In this case, the patch header stream is stored in the [MsiPatchHeaders table](#) to overcome the stream name limitation described in [OLE Limitations on Streams](#).

#### StreamRef\_

External key into the MsiPatchHeaders table specifying the row that contains the patch header stream.

### Remarks

This table is processed by the [PatchFiles action](#). It is usually added to the install package by a transform from a patch package. It is usually not authored directly into an install package.

### Validation

[ICE03](#)  
[ICE06](#)  
[ICE29](#)

ICE45

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PatchPackage Table

The PatchPackage table describes all patch packages that have been applied to this product. For each patch package, the unique identifier for the patch is provided along with information about the media image the on which the patch is located.

The PatchPackage table has the following columns.

Column	Type	Key	Nullable
PatchId	GUID	Y	N
Media_	Integer	N	N

## Columns

### PatchId

This column contains a unique identifier for this particular patch.

### Media\_

This column is a foreign key to the DiskId column of the [Media Table](#) and indicates the disk containing the patch package.

## Remarks

The PatchPackage table is required in every patch package. If the table is missing, attempts to install the patch fail with "Error 2768: Transform in patch package is invalid." This table is usually added to the install package by a transform from a patch package. It is usually not authored directly into an install package.

## Validation

[ICE03](#)

[ICE06](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ProgId Table

The ProgId table contains information for program IDs and version independent program IDs that must be generated as a part of the product advertisement.

The ProgId table has the following columns.

Column	Type	Key	Nullable
ProgId	Text	Y	N
ProgId_Parent	Text	N	Y
Class_	GUID	N	Y
Description	Text	N	Y
Icon_	Identifier	N	Y
IconIndex	Integer	N	Y

## Columns

### ProgId

The program ID or version independent program ID. ProgIds listed in the ProgId table are registered if the CLSID listed in the Class\_ column of this table is scheduled to be advertised or installed. When the ProgId is selected for registration, all ProgIds that refer to this row through the ProgId\_Parent column are also selected for registration.

### ProgId\_Parent

Defined only for version independent program IDs. This field is a foreign key into the ProgId column. To define a version independent program ID, enter the corresponding ProgId into the ProgId\_Parent column. When the ProgId is selected for installation, the corresponding version-independent ProgIds associated through the ProgId\_Parent column are also selected for registration.

### Class\_

An optional foreign key into the [Class table](#). This column must be Null for a version independent ProgId. If the Class\_value for a ProgId is null, the ProgId is registered when it appears in the ProgId column of a row in the [Extension table](#) and the extension has at least one Verb associated with it in the [Verb table](#). ProgIds selected for registration in this manner do not install other ProgIds that reference the current ProgId through the ProgId\_Default value.

#### Description

An optional localized description of the associated program ID.

#### Icon\_

An optional foreign key into the [Icon table](#) that specifies the icon file associated with this ProgId. This is written under the DefaultIcon key associated with this ProgId. This column must be Null for a version independent ProgId.

#### IconIndex

The Icon index into the icon file. This column must be Null for a version independent ProgId.

## Remarks

The [RegisterProgIdInfo](#) and [UnregisterProgIdInfo](#) actions in [sequence tables](#) process the information in this table. For information about using sequence tables, see [Using a Sequence Table](#).

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE32](#)  
[ICE36](#)  
[ICE89](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Property Table

The Property table contains the property names and values for all defined properties in the installation. Properties with Null values are not present in the table.

The Property table has the following columns.

Column	Type	Key	Nullable
Property	Identifier	Y	N
Value	Text	N	N

## Columns

### Property

The name of a property. See [Properties](#).

### Value

A localizable string value for the property. This may never be Null or an empty string.

## Remarks

Note that you cannot use the Property table to set a property to the value of another property. The installer does nothing to the text string entered in the Value column before setting the property in the Property column. If FirstProperty is entered into the Property column and [SecondProperty] in the Value column, the value of FirstProperty is set to the text string "[SecondProperty]" and not to the value of the SecondProperty property. This is necessary to prevent creating circular references in the Property table. Instead, you can set one property to another by using a [Custom Action Type 51](#).

Note that the [AdminProperties](#) property can be used to override the values in the Property table during an [Administrative Installation](#).

Do not use properties for passwords or other information that must remain secure. The installer may write the value of a property authored into the Property table, or created at runtime, into a log or the system registry.

## **Validation**

[ICE03](#)

[ICE05](#)

[ICE06](#)

[ICE16](#)

[ICE24](#)

[ICE31](#)

[ICE34](#)

[ICE36](#)

[ICE40](#)

[ICE46](#)

[ICE48](#)

[ICE52](#)

[ICE61](#)

[ICE73](#)

[ICE74](#)

[ICE80](#)

[ICE87](#)

[ICE88](#)

[ICE91](#)

[ICE99](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PublishComponent Table

The PublishComponent table associates components listed in the [Component table](#) with a qualifier text-string and a category ID GUID. Components with parallel functionality that have been grouped together in this way are referred to as qualified components. See [Qualified Components](#). This provides the installer with a method for single-level indirection when referring to components. See [Using Qualified Components](#).

The PublishComponent table has the following columns.

Column	Type	Key	Nullable
ComponentId	GUID	Y	N
Qualifier	Text	Y	N
Component_	Identifier	Y	N
AppData	Text	N	Y
Feature_	Identifier	N	N

## Columns

### ComponentId

A string [GUID](#) that represents the category of components being grouped together. Note that this column's title is misleading. This is the GUID for the category of qualified components and is not the same GUID appearing in the ComponentId column of the [Component table](#). Here it refers to a server that provides the functionality of a component to external clients rather than the component itself.

### Qualifier

A text string that qualifies the value in the ComponentId column. A qualifier is used to distinguish multiple forms of the same component, such as a component that is implemented in multiple languages. These are the qualifier text-strings returned by

## **MsiEnumComponentQualifiers.**

### Component\_

External key into column one of the [Component table](#). This identifier refers to the qualified component's record in the Component table.

### AppData

An optional localizable text describing the qualified component of this record. The string is commonly parsed by the application and can be displayed to the user. It should describe the qualified component. This can be retrieved with

[MsiEnumComponentQualifiers](#).

### Feature\_

External key into column one of the [Feature table](#). This is the feature using this qualified component.

## **Remarks**

This table is referred to when the [PublishComponents action](#) or the [UnpublishComponents action](#) is executed.

Note that the name of this table is misleading. This table is not required in order to author advertisement. See the Attributes column of the [Component table](#) and [Feature table](#) for information on how to set the installation state of components to advertise.

## **Validation**

[ICE03](#)

[ICE06](#)

[ICE19](#)

[ICE22](#)

[ICE32](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RadioButton Table

Radio buttons are not treated as individual controls, but they are part of a radio button group that functions as a [RadioButtonGroup control](#). The RadioButton table lists the buttons for all the groups.

The RadioButton table has the following columns.

Column	Type	Key	Nullable
Property	Identifier	Y	N
Order	Integer	Y	N
Value	Formatted	N	N
X	Integer	N	N
Y	Integer	N	N
Width	Integer	N	N
Height	Integer	N	N
Text	Formatted	N	Y
Help	Text	N	Y

## Columns

### Property

A named property to be tied to this radio button. All the buttons tied to the same property become part of the same group.

### Order

A positive integer used to determine the ordering of the items within one list. The integers do not have to be consecutive.

### Value

The value string associated with this button. Selecting the button sets the associated property to this value.

### X

The horizontal coordinate within the group of the upper-left corner of

the bounding rectangle of the radio button. This must be a non-negative number.

Y

The vertical coordinate within the group of the upper-left corner of the bounding rectangle of the radio button. This must be a non-negative number.

Width

The width of the button. This must be a non-negative number.

Height

The height of the button. This must be a non-negative number.

Text

The localizable, visible title to be assigned to the radio button. If the text is too long to be fit on the control, then it is truncated. If the button displays an icon or a bitmap, then this column contains the name of the picture, which is a key into the [Binary table](#). There is no way to show both a picture and text on a button.

Help

The Help strings used with the button. The text is optional and is localizable. The string is divided into two parts separated by a character (|). The first part of the string is used as ToolTip text. This text is shown by screen readers for controls that contain a picture. The second part is used for context-sensitive Help, although context-sensitive help has not yet been implemented. The separator character is required even if only one of the two kinds of text is present.

## Remarks

The integer values for x, y, width, and height are in the [installer units](#), not dialog units. An installer unit is equal to one-twelfth the height of the 10-point MS Sans Serif font size. Coordinates for the controls are relative to the billboard.

The coordinates of the buttons are given relative to the group. If the coordinates of the group are changed, then the buttons within the group

remain in the same relative position to each other.

The contents of the Value and Text fields are formatted by the **MsiFormatRecord** function when the control is created, therefore they can contain any expression that the **MsiFormatRecord** function can interpret. The formatting occurs only when the control is created, and it is not updated if a property involved in the expression is modified during the life of the control.

Every RadioButtonGroup control is associated with a property. The default value for this property must be initialized in the [Property table](#). Within each RadioButtonGroup specified in the RadioButton table, there may be one radio button that has a value in the Value field that matches the default value for this property. This is the default button for the RadioButtonGroup control. The default button is initially shown as selected in the control.

Note that the user cannot change focus in a dialog box by pressing the TAB key to a RadioButtonGroup Control until one of the buttons in the group has been selected. To make the focus move to this button group by pressing the TAB key, specify one of the buttons as a default button for the group.

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE17](#)  
[ICE34](#)  
[ICE46](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Registry Table

The Registry table holds the registry information that the application needs to set in the system registry.

The Registry table has the following columns.

Column	Type	Key	Nullable
Registry	Identifier	Y	N
Root	Integer	N	N
Key	RegPath	N	N
Name	Formatted	N	Y
Value	Formatted	N	Y
Component_	Identifier	N	N

## Columns

### Registry

Primary key used to identify a registry record.

### Root

The predefined root key for the registry value. Enter a value of -1 in this field to make the root key dependent on the type of installation. Enter one of the other values in the following table to force the registry value to be written under a particular root key.

Constant	Hexadecimal	Decimal	Root key
(none)	- 0x001	-1	If this is a per- the registry va under <b>HKEY_CUR</b> If this is a per- installation, th is written unde <b>HKEY_LOC</b> .

			Note that a per installation is : setting the <b>AL</b> property to 1.
msidbRegistryRootClassesRoot	0x000	0	<b>HKEY_CLASSES_ROOT</b> The installer writes the value from <b>HKCU\Software\installatic</b> during installation for the current user.  The installer writes the value from <b>HKLM\Software\installatic</b> during per-machine installations.
msidbRegistryRootCurrentUser	0x001	1	<b>HKEY_CURRENT_USER</b>
msidbRegistryRootLocalMachine	0x002	2	<b>HKEY_LOCAL_MACHINE</b>
msidbRegistryRootUsers	0x003	3	<b>HKEY_USERS</b>

Note that it is recommended that registry entries written to the **HKCU** hive reference a component having the RegistryKeyPath bit set in the Attributes column of the [Component table](#). This ensures that the installer writes the necessary registry entries when there are multiple users on the same computer.

#### Key

The localizable key for the registry value.

#### Name

This column contains the registry value name (localizable). If this is Null, then the data entered into the Value column are written to the default registry key.

If the Value column is Null, then the strings shown in the following table in the Name column have special significance.

#### **String Meaning**

+	The key is to be created, if absent, when the component is installed.
-	The key is to be deleted, if present, with all of its values and subkeys, when the component is uninstalled.
*	The key is to be created, if absent, when the component is installed. Additionally, the key is to be deleted, if present, with all of its values and subkeys, when the component is uninstalled.

Note that the [RemoveRegistry table](#) must be used if an installed registry key is to be deleted, with its values and subkeys, when the component is installed.

## Value

This column is the localizable registry value. The field is [Formatted](#). If the value is attached to one of the following prefixes (i.e. *##value*) then the value is interpreted as described in the table. Note that each prefix begins with a number sign (#). If the value begins with two or more consecutive number signs (##), the first # is ignored and value is interpreted and stored as a string.

<b>Prefix</b>	<b>Meaning</b>
#x	The value is interpreted and stored as a hexadecimal value (REG_BINARY).
#%	The value is interpreted and stored as an expandable string (REG_EXPAND_SZ).
#	The value is interpreted and stored as an integer (REG_DWORD).

- If the value contains the sequence tilde [~], then the value is interpreted as a Null-delimited list of strings (REG\_MULTI\_SZ). For example, to specify a list containing the three strings a, b and c, use "a[~]b[~]c".
- The sequence [~] within the value separates the individual

strings and is interpreted and stored as a Null character.

- If a [~] precedes the string list, the strings are to be appended to any existing registry value strings. If an appending string already occurs in the registry value, the original occurrence of the string is removed.
- If a [~] follows the end of the string list, the strings are to be prepended to any existing registry value strings. If a prepending string already occurs in the registry value, the original occurrence of the string is removed.
- If a [~] is at both the beginning and the end or at neither the beginning nor the end of the string list, the strings are to replace any existing registry value strings.
- Otherwise, the value is interpreted and stored as a string (REG\_SZ).

Component\_

External key into the first column of the [Component table](#) referencing the component that controls the installation of the registry value.

## Remarks

The [WriteRegistryValues](#) and [RemoveRegistryValues](#) actions in [sequence tables](#) process the information in this table. For information about using sequence tables, see [Using a Sequence Table](#).

The registry information is written out to the system registry when the corresponding component has been selected to be installed locally or run from source.

Note that the installer removes a registry key after removing the last value or subkey under the key. To prevent an empty registry key from being removed when uninstalling, write a dummy value under the key you need to keep and enter + in the Name column. If \* is in the Name column, the key is deleted, with all of its values and subkeys, when the component is removed.

A custom action can be used to add rows to the Registry table during an installation, uninstallation, or repair transaction. These rows do not persist in the Registry table and the information is only available during the current transaction. The custom action must therefore be run in every installation, uninstallation, or repair transaction that requires the information in these additional rows. The custom action must come before the RemoveRegistryValues and WriteRegistryValues actions in the action sequence.

For information on how to secure a registry key see the [MsiLockPermissionsEx Table](#) and [LockPermissions Table](#).

## Validation

[ICE02](#)  
[ICE03](#)  
[ICE06](#)  
[ICE32](#)  
[ICE38](#)  
[ICE43](#)  
[ICE46](#)  
[ICE49](#)  
[ICE53](#)  
[ICE55](#)  
[ICE57](#)  
[ICE70](#)  
[ICE80](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RegLocator Table

The RegLocator table holds the information needed to search for a file or directory using the registry, or to search for a particular registry entry itself. This table has the following columns.

Column	Type	Key	Nullable
Signature_	Identifier	Y	N
Root	Integer	N	N
Key	RegPath	N	N
Name	Formatted	N	Y
Type	Integer	N	Y

## Columns

### Signature\_

The value in the Signature\_ field represents a unique signature that is an external key into column one of the [Signature](#) table. If this signature is present in the Signature table, the search is for a file. If this signature is absent from the Signature table, and the value of the Type column is msidbLocatorTypeRawValue, the search is for the registry key name pointed to by the RegLocator table. Otherwise the search is for a directory pointed to by the RegLocator table.

### Root

The predefined root key for the registry value.

Constant	Hexadecimal	Decimal	Root key
msidbRegistryRootClassesRoot	0x000	0	HKEY_CLAS
msidbRegistryRootCurrentUser	0x001	1	HKEY_CURF
msidbRegistryRootLocalMachine	0x002	2	HKEY_LOCA
msidbRegistryRootUsers	0x003	3	HKEY_USER

## Key

The key for the registry value.

## Name

The registry value name. If this value is null, then the value from the key's unnamed or default value, if any, is retrieved.

## Type

A value that determines if the registry value is a file name, a directory location, or raw registry value.

The following table lists valid values. Set one of the first three values and `msidbLocatorType64bit` if necessary. If the entry in this field is absent, `Type` is set to be 1.

<b>Constant</b>	<b>Hexadecimal</b>	<b>Decimal</b>	<b>Description</b>
<code>msidbLocatorTypeDirectory</code>	0x000	0	Key path is a directory.
<code>msidbLocatorTypeFileName</code>	0x001	1	Key path is a file name.
<code>msidbLocatorTypeRawValue</code>	0x002	2	Key path is a registry value.
<code>msidbLocatorType64bit</code>	0x010	16	Set this bit to have the installer search the 64-bit portion of the registry. Do not set this bit to have the installer search the 32-bit portion of the registry.

## Remarks

Note that if the value in the Type field is `msidbLocatorTypeRawValue`, the installer sets the value of the property specified in the Property field of the [AppSearch](#) table to the registry value. The installer adds a prefix to the registry value that identifies the type of registry value. For more information about types of registry values, see [Registry Value Types](#).

Registry type	Prefix added by Installer
REG_SZ	None, but if the first character of the registry value is #, the installer escapes the character by prefixing a another #.
DWORD	"#" optionally followed by '+' or '-'
REG_EXPAND_SZ	"#%"
REG_MULTI_SZ	Null. The installer sets the property to a value beginning with a null and ending with a null.
REG_BINARY	"#x" In case of REG_BINARY, the installer converts and saves each hexadecimal digit (nibble) as an ASCII character prefixed by "#x".

Typically, the columns in this table are not localized. If an author decides to search for products in multiple languages, then there must be a separate entry included in the table for each language.

Note that it is not possible to use the `RegLocator` table to check only for the presence of the key. However, you can search for the default value of a key and retrieve its value if it is not empty.

For more information, see [Searching for Existing Applications, Files, Registry Entries or .ini File Entries](#).

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE46](#)  
[ICE80](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RemoveFile Table

The RemoveFile table contains a list of files to be removed by the [RemoveFiles action](#). Setting the FileName column of this table to Null supports the removal of empty folders.

The RemoveFile table has the following columns.

Column	Type	Key	Nullable
FileKey	Identifier	Y	N
Component_	Identifier	N	N
FileName	WildcardFilename	N	Y
DirProperty	Identifier	N	N
InstallMode	Integer	N	N

## Columns

### FileKey

Primary key used to identify this particular table entry.

### Component\_

External key the first column of the [Component table](#). This field references the component that controls the file to be removed.

### FileName

This column contains the localizable name of the file to be removed. If this column is null, then the specified folder will be removed if it is empty. All of the files that match the wildcard will be removed from the specified directory.

### DirProperty

Name of a property whose value is assumed to resolve to the full path to the folder of the file to be removed. The property can be the name of a directory in the [Directory table](#), a property set by the [AppSearch table](#), or any other property that represents a full path.

### InstallMode

Must be one of the following values.

Constant	Hexadecimal	Decimal	Description
msidbRemoveFileInstallModeOnInstall	0x001	1	Remove associated files when installing (msiInstallModeOnInstall)
msidbRemoveFileInstallModeOnRemove	0x002	2	Remove associated files when removing (msiInstallModeOnRemove)
msidbRemoveFileInstallModeOnBoth	0x003	3	Remove associated files when installing and removing (msiInstallModeOnBoth)

## Remarks

The file references in this table are processed by the [RemoveFiles action](#).

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE18](#)  
[ICE32](#)  
[ICE45](#)  
[ICE64](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RemoveIniFile Table

The RemoveIniFile table contains the information an application needs to delete from a .ini file.

The RemoveIniFile table has the following columns.

Column	Type	Key	Nullable
RemoveIniFile	Identifier	Y	N
FileName	FileName	N	N
DirProperty	Identifier	N	Y
Section	Formatted	N	N
Key	Formatted	N	N
Value	Formatted	N	Y
Action	Integer	N	N
Component_	Identifier	N	N

## Columns

### RemoveIniFile

The key for this table.

### FileName

The .ini file name in which to delete the information.

### DirProperty

Name of a property whose value is assumed to resolve to the full path to the folder of the .ini file to be removed. The property can be the name of a directory in the [Directory table](#), a property set by the [AppSearch table](#), or any other property that represents a full path.

### Section

The localizable .ini file section.

### Key

The localizable .ini file key below the section.

## Value

The localizable value to be deleted. The value is required when Action is 4.

## Action

The type of modification to be made.

Constant	Hexadecimal	Decimal	Meaning
msidbIniFileActionRemoveLine	0x002	2	Deletes .ini entry.
msidbIniFileActionRemoveTag	0x004	4	Deletes a tag from a .ini entry.

## Component\_

External key into first column of the [Component table](#) referencing the component that controls the deletion of the .ini value.

## Remarks

The .ini file information is deleted when the corresponding Component has been selected to be installed, either locally or run from source.

This table is referred to when the [RemoveIniValues action](#) is executed.

If the Directory\_ column is specified as null, the ini file location is the standard Windows ini location which is the Windows directory by default.

Removing the last value from a section deletes that section. There is no other way to delete an entire section other than removing all its values.

## Validation

[ICE03](#)

[ICE06](#)

[ICE32](#)

[ICE40](#)

[ICE46](#)

ICE69

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# RemoveRegistry Table

The RemoveRegistry table contains the registry information the application needs to delete from the system registry.

The RemoveRegistry table has the following columns.

Column	Type	Key	Nullable
RemoveRegistry	Identifier	Y	N
Root	Integer	N	N
Key	RegPath	N	N
Name	Formatted	N	Y
Component_	Identifier	N	N

## Columns

RemoveRegistry

The key for this table.

Root

The predefined root key for the registry value.

Constant	Hexadecimal	Decimal	Root key
(none)	- 0x001	-1	<b>HKEY_CUR</b> Installer sets tl doing a per-us
(none)	-0x001	-1	<b>HKEY_LOC</b> Installer sets tl doing an all-us with <b>ALLUSI</b>
msidbRegistryRootClassesRoot	0x000	0	<b>HKEY_CLA</b> The installer r value from the

			<b>HKCU\Softw</b> during installa user and per-n <a href="#">installation co</a>
msidbRegistryRootCurrentUser	0x001	1	<b>HKEY_CUR</b>
msidbRegistryRootLocalMachine	0x002	2	<b>HKEY_LOC.</b>
msidbRegistryRootUsers	0x003	3	<b>HKEY_USEI</b>

### Key

The localizable key for the registry value.

### Name

The localizable registry value name.

The following string in the Name column has special significance.

<b>String Meaning</b>	
"-"	The key is to be deleted, if present, with all of its values and subkeys, when the component is installed.

Note that the [Registry table](#) should be used to create or remove a registry key when the component is removed.

### Component\_

External key into the first column of the [Component table](#) referencing the component that controls the deletion of the registry value.

## Remarks

The registry information is deleted from the system registry when the corresponding component has been selected to be installed locally or run from source.

This table is referred to when the [RemoveRegistryValues action](#) is executed.

## Validation

[ICE03](#)

[ICE06](#)

[ICE32](#)

[ICE46](#)

[ICE69](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ReserveCost Table

The ReserveCost table is an optional table that allows the author to reserve an amount of disk space in any directory that depends on the installation state of a component.

The ReserveCost table has the following columns.

Column	Type	Key	Nullable
ReserveKey	Identifier	Y	N
Component_	Identifier	N	N
ReserveFolder	Identifier	N	Y
ReserveLocal	DoubleInteger	N	N
ReserveSource	DoubleInteger	N	N

## Columns

### ReserveKey

Primary key that uniquely identifies a ReserveCost table entry.

### Component\_

External key to column one of the [Component](#) table. Reserves a specified amount of space if this component is to be installed.

### ReserveFolder

This column contains the name of a property that is the full path to the destination directory. This property name is typically the name of a directory in the [Directory](#) table or the name of a property set obtained using the [Appsearch](#) action. This adds the amount of disk space specified in ReserveLocal or ReserveSource to the volume cost of the device containing the directory.

### ReserveLocal

The number of bytes of disk space to reserve if the linked component is installed to run locally.

### ReserveSource

The number of bytes of disk space to reserve if the linked component is installed to run from source.

## Remarks

Reserving cost in this way could be useful for authors who want to ensure that a minimum amount of disk space will be available after the installation is completed. For example, this disk space might be reserved for user documents or for application files (such as index files) that are created only after the application is started following installation.

You can use the ReserveCost table to enable custom actions to specify an approximate cost for any files, registry entries, or other items that the custom action might install. Custom actions that add entries to the ReserveCost table should be sequenced between the [CostInitialize](#) and [FileCost](#) actions. This is necessary for the FileCost action to correctly initialize the costing of all components affected by entries in the ReserveCost table.

## Validation

[ICE03](#)

[ICE06](#)

[ICE32](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SelfReg Table

The SelfReg table contains information about modules that need to be self registered. The installer calls the **DllRegisterServer** function during installation of the module; it calls **DllUnregisterServer** during uninstallation of the module. The installer does not self register EXE files.

The SelfReg table has the following columns.

Column	Type	Key	Nullable
File_	Identifier	Y	N
Cost	Integer	N	Y

## Columns

File\_

External key into the first column of the [File table](#) indicating the module that needs to be registered.

Cost

The cost of registering the module in bytes. This must be a non-negative number.

## Remarks

Installation package authors are strongly advised against using self registration. Instead they should register modules by authoring one or more tables provided by the installer for this purpose. For more information, see [Registry Tables Group](#). Many of the benefits of having a central installer service are lost with self registration because self-registration routines tend to hide critical configuration information. Reasons for avoiding self registration include:

- Rollback of an installation with self-registered modules cannot be safely done using **DllUnregisterServer** because there is no way of

telling if the self-registered keys are used by another feature or application.

- The ability to use advertisement is reduced if Class or extension server registration is performed within self-registration routines.
- The installer automatically handles HKCR keys in the registry tables for both per-user or per-machine installations. **DIIRegisterServer** routines currently do not support the notion of a per-user HKCR key.
- If multiple users are using a self-registered application on the same computer, each user must install the application the first time they run it. Otherwise the installer cannot easily determine that the proper HKCU registry keys exist.
- The **DIIRegisterServer** can be denied access to network resources such as type libraries if a component is both specified as run-from-source and is listed in the SelfReg table. This can cause the installation of the component to fail to during an administrative installation.
- Self-registering DLLs are more susceptible to coding errors because the new code required for **DIIRegisterServer** is commonly different for each DLL. Instead use the registry tables in the database to take advantage of existing code provided by the installer.
- Self-registering DLLs can sometimes link to auxiliary DLLs that are not present or are the wrong version. In contrast, the installer can register the DLLs using the registry tables with no dependency on the current state of the system.

**Note** You cannot specify the order in which the installer registers or unregisters self-registering DLLs by using the [SelfRegModules](#) and [SelfUnRegModules](#) actions. See [Specifying the Order of Self Registration](#).

## Validation

ICE03

ICE06  
ICE32

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ServiceControl Table

The ServiceControl table is used to control installed or uninstalled services.

**Note** Services that rely on the presence of an [assembly](#) in the Global Assembly Cache (GAC) cannot be installed or started using the [ServiceInstall](#) and ServiceControl tables. If you need to start a service that depends on an assembly in the GAC, you must use a custom action sequenced after the [InstallFinalize action](#) or a [commit custom action](#). For information about installing assemblies to the GAC see [Installation of Assemblies to the Global Assembly Cache](#).

The ServiceControl table has the following columns.

Column	Type	Key	Nullable
ServiceControl	Identifier	Y	N
Name	Formatted	N	N
Event	Integer	N	N
Arguments	Formatted	N	Y
Wait	Integer	N	Y
Component_	Identifier	N	N

## Columns

### ServiceControl

This is the primary key of this table.

### Name

This column is the string naming the service. This column can be used to control a service that is not installed.

### Event

This column contains the operations to be performed on the named service. Note that when stopping a service, all services that depend on that service are also stopped. When deleting a service that is

running, the installer stops the service.

The values in this field are bit fields that can be combined into a single value that represents several operations.

The following values are only used during an installation.

<b>Constant</b>	<b>Hexadecimal</b>	<b>Decimal</b>	<b>Description</b>
msidbServiceControlEventsStart	0x001	1	Starts the service during the <a href="#">StartServices</a> action.
msidbServiceControlEventsStop	0x002	2	Stops the service during the <a href="#">StopServices</a> action.
(none)	0x004	4	<reserved>
msidbServiceControlEventsDelete	0x008	8	Deletes the service during the <a href="#">DeleteServices</a> action.

The following values are only used during an uninstall.

<b>Constant</b>	<b>Hexadecimal</b>	<b>Decimal</b>	<b>Description</b>
msidbServiceControlEventsUninstallStart	0x010	16	Starts service during the <a href="#">StartServices</a> action.
msidbServiceControlEventsUninstallStop	0x020	32	Stops service during the <a href="#">StopServices</a> action.

			<a href="#">action</a>
(none)	0x040	64	<rese
msidbServiceControlEventUninstallDelete	0x080	128	Delet servic the <a href="#">Delet</a> <a href="#">action</a>

## Arguments

A list of arguments for starting services. The arguments are separated by null characters [~]. For example, the list of arguments One, Two, and Three are listed as: One[~]Two[~]Three.

## Wait

Leaving this field null or entering a value of 1 causes the installer to wait a maximum of 30 seconds for the service to complete before proceeding. The wait can be used to allow additional time for a critical event to return a failure error. A value of 0 in this field means to wait only until the service control manager (SCM) reports that this service is in a pending state before continuing with the installation.

## Component\_

External key to column one of the [Component Table](#).

## Remarks

The StartServices, StopServices, and DeleteServices actions in [sequence tables](#) process the information in this table. For information about using sequence tables, see [Using a Sequence Table](#).

Use the Name column to start, stop, or delete services that are being replaced by the installation or that are dependent upon a new service that is being installed. For example, entering MyService into the ServiceControl column can tie this service to MyComponent in the Component\_ column. If the bit field in the Event column is set for start while installing, then the installer starts MyService when installing MyComponent.

## Validation

[ICE03](#)

[ICE06](#)

[ICE32](#)

[ICE45](#)

[ICE46](#)

[ICE69](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ServiceInstall Table

The ServiceInstall table is used to install a service and has the following columns.

Column	Type	Key	Nullable
ServiceInstall	Identifier	Y	N
Name	Formatted	N	N
DisplayName	Formatted	N	Y
ServiceType	DoubleInteger	N	N
StartType	DoubleInteger	N	N
ErrorControl	DoubleInteger	N	N
LoadOrderGroup	Formatted	N	Y
Dependencies	Formatted	N	Y
StartName	Formatted	N	Y
Password	Formatted	N	Y
Arguments	Formatted	N	Y
Component_	Identifier	N	N
Description	Formatted	N	Y

## Columns

### ServiceInstall

This is the primary key for the table.

### Name

This column is the string that gives the service name to install. The string has a maximum length of 256 characters. The service control manager database preserves the case of the characters in the service name, but comparisons of service names are case insensitive. Forward-slash (/) and back-slash (\) are invalid service

name characters.

#### DisplayName

This column is the localizable string that user interface programs use to identify the service. The string has a maximum length of 256 characters. The service control manager preserves the case of the display name, but display name comparisons are case insensitive.

#### ServiceType

This column is a set of bit flags that specify the type of service. One of the following service types must be specified in this column.

Type of service	Value	Description
SERVICE_WIN32_OWN_PROCESS	0x00000010	A Microsoft Win32 service that runs its own process.
SERVICE_WIN32_SHARE_PROCESS	0x00000020	A Win32 service that shares a process.
SERVICE_INTERACTIVE_PROCESS	0x00000100	A Win32 service that interacts with the desktop. This value cannot be used alone and must be added to one of the two previous types. The StartName column must be set to LocalSystem or null when using this flag.

The following types of service are unsupported.

Type of service	Value	Description
SERVICE_KERNEL_DRIVER	0x00000001	A driver service.
SERVICE_FILE_SYSTEM_DRIVER	0x00000002	A file system driver service.

### StartType

This column is a set of bit flags that specify when to start the service. One of the following types of service start must be specified in this column.

Type of service start	Value	Description
SERVICE_AUTO_START	0x00000002	A service start during startup of the system.
SERVICE_DEMAND_START	0x00000003	A service start when the service control manager calls the <b>StartService</b> function.
SERVICE_DISABLED	0x00000004	Specifies a service that can no longer be started.

The Windows Installer cannot use the SERVICE\_BOOT\_START and SERVICE\_SYSTEM\_START options.

### ErrorControl

This column specifies the action taken by the startup program if the service fails to start during startup. These values affect the ServiceControl StartService events for installed services. One of the following error control flags must be specified in this column.

Adding the constant msidbServiceInstallErrorControlVital (value = 0x08000) to the flags in the following table specifies that the overall install should fail if the service cannot be installed into the system.

**Startup program's**

<b>Error control flag</b>	<b>Value</b>	<b>action</b>
SERVICE_ERROR_IGNORE	0x00000000	Logs the error and continues with the startup operation.
SERVICE_ERROR_NORMAL	0x00000001	Logs the error, displays a message box and continues the startup operation.
SERVICE_ERROR_CRITICAL	0x00000003	Logs the error if it is possible and the system is restarted with the last configuration known to be good. If the last-known-good configuration is being started, the startup operation fails.

### LoadOrderGroup

This column contains the string that names the load ordering group of which this service is a member. Specify null or an empty string if the service does not belong to a group.

### Dependencies

This column is a list of names of services or load ordering groups that the system must start before this service. Separate names in the list by Nulls. If the service has no dependencies, then specify Null or an empty string. Use the syntax [~] to insert a Null. Dependency on a group means that this service can run if at least one member of the group is running after an attempt to start all members of the group.

For example, to require that the system start service1 and service2, before starting the service listed in the ServiceInstall column, enter service1[~]service2[~][~] into the Dependencies column. The identifiers service1 and service2 must either occur in the primary key of the table or be the name of the service that is already installed.

You must prefix group names with + so that they can be

distinguished from a service name. To require that the system start service1 and at least one member of the ordering group MyGroup before starting the service listed in the ServiceInstall column, enter service1[~]+MyGroup[~][~].

#### StartName

The service is logged on as the name given by the string in this column. If the service type is SERVICE\_WIN32\_OWN\_PROCESS use an account name in the form: DomainName\UserName. If the account belongs to the built-in domain it is permitted to specify .\UserName. The LocalSystem account must be used if the type of service is SERVICE\_WIN32\_SHARE\_PROCESS or SERVICE\_INTERACTIVE\_PROCESS. The **CreateService** function uses the LocalSystem account if StartName is specified as null and most services therefore leave this column blank.

#### Password

This string is the password to the account name specified in the StartName column. Note that the user must have permissions to log on as a service. The service has no password if StartName is null or an empty string. The Startname of LocalSystem is null, and therefore the password in this instance is null, so most services leave this column blank.

Note that after deleting a service that was installed with a user name and password, the installer cannot rollback the service without first using a custom action to get the password. The installer can acquire all the necessary information about the service except the password, which is stored in a protected part of the system. The custom action acquires the password by prompting the user, reading a property from the database, or reading a file. The custom action must then call **ChangeServiceConfig**, to supply the password, before reinstalling the service.

Windows Installer does not write the value entered into the Password field into the log file.

#### Arguments

This column contains any command line arguments or properties required to run the service.

#### Component\_

External key to column one of the [Component Table](#). Note that to install this service using the InstallService table, the KeyPath for this component must be the executable file for the service.

## Description

This column contains a localizable description for the service being configured. If this column is left blank the installer uses the existing description of the service if one exists. For more information, see SERVICE\_DESCRIPTION in the Microsoft Windows Software Development Kit (SDK). To erase an existing description enter "[~]" in this column. This results in a blank description for either a new or existing service.

## Remarks

The [InstallServices](#) action in [sequence tables](#) processes the information in this table. For information about using sequence tables, see [Using a Sequence Table](#).

This table has most of the parameters to the Win32 **CreateService** function.

Although it is possible to use the user interface to specify that a service be installed as run-from-source, the installer does not actually support this type of installation. Services that run with the privilege level of the local system must be installed to run from the local hard drive. Avoid installing services that impersonate the privileges of a particular user because this may write security data into a log or the system registry. This can potentially create a security problem, password conflict, or the loss of configuration data when the system is restarted.

To delete a service during an uninstallation, there must be a corresponding record for the service in the [ServiceControl table](#) and the msidbServiceControlEventUninstallDelete flag must appear in the Event column. The installer does not delete a service in the ServiceInstall table during uninstallation without this entry in the ServiceControl table.

For information on how to secure a service see the [MsiLockPermissionsEx Table](#).

## Validation

[ICE03](#)

[ICE06](#)

[ICE32](#)

[ICE45](#)

[ICE46](#)

[ICE66](#)

[ICE69](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# SFPCatalog Table

The SFPCatalog table contains the catalogs used by Windows Me.

The SFPCatalog table has the following columns.

Column	Type	Key	Nullable
SFPCatalog	Filename	Y	N
Catalog	Binary	N	N
Dependency	Formatted	N	Y

## Columns

### SFPCatalog

The short file name for the catalog. Because catalogs have no version, the catalog specified in this column can overwrite a catalog that has the same name and is already installed on the local system. See the file versioning topic [Neither File Has a Version](#).

### Catalog

Binary data for the catalog.

### Dependency

The catalog specified in this field is the parent of the dependent catalog specified in the SFPCatalog field. Enter the short file name of the parent catalog into the Dependency field. This field is a key back into the SFPCatalog column. The dependency match is case sensitive.

If the Dependency field references another record in this SFPCatalog table, the parent catalog is installed before the dependent catalog.

If the Dependency field references a parent catalog that is not present in the SFPCatalog field of this table, and if the parent catalog is not already installed, the installation fails.

## Remarks

The [InstallSFPCatalogFile](#) action queries the [Component](#) table, [File](#) table, [FileSFPCatalog](#) table and [SFPCatalog](#) table.

## Validation

[ICE03](#)

[ICE06](#)

[ICE29](#)

[ICE32](#)

[ICE46](#)

[ICE66](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Shortcut Table

The Shortcut table holds the information the application needs to create shortcuts on the user's computer.

The Shortcut table has the following columns.

Column	Type	Key	Nullable
Shortcut	Identifier	Y	N
Directory_	Identifier	N	N
Name	Filename	N	N
Component_	Identifier	N	N
Target	Shortcut	N	N
Arguments	Formatted	N	Y
Description	Text	N	Y
Hotkey	Integer	N	Y
Icon_	Identifier	N	Y
IconIndex	Integer	N	Y
ShowCmd	Integer	N	Y
WkDir	Identifier	N	Y
DisplayResourceDLL	Formatted	N	Y
DisplayResourceId	Integer	N	Y
DescriptionResourceDLL	Formatted	N	Y
DescriptionResourceId	Integer	N	Y

## Columns

Shortcut

The key value for this table.

## Directory\_

The external key into the first column of the [Directory table](#). This column specifies the directory in which the Shortcut file is created.

## Name

The localizable name of the shortcut to be created.

## Component\_

The external key into the first column of the [Component table](#). The installer uses the installation state of the component specified in this column to determine whether the shortcut is created or deleted. This component must have a valid key path for the shortcut to be installed. If the Target column contains the name of a feature, the file launched by the shortcut is the key file of the component listed in this column.

## Target

The shortcut target.

For an advertised shortcut, this column must be an external key into the first column of the [Feature table](#). The installer evaluates the entry in the Target field as an [Identifier](#) and the entry must be a valid foreign key into the [Feature Table](#). The file launched by the shortcut in this case is the key file of the component listed in the [Component\\_](#) column. When the shortcut is activated, the installer verifies that all the components in the feature are installed before launching this file.

For a non-advertised shortcut, the installer evaluates this field as a [Formatted](#) string. The field should contain a property identifier enclosed by square brackets ([ ]), that is expanded into the file or a folder pointed to by the shortcut. For more information, see the [CreateShortcuts action](#).

## Arguments

The command-line arguments for the shortcut.

Note that the resolution of properties in the Arguments field is limited. A property formatted as *[Property]* in this field can only be resolved if the property already has the intended value when the component that owns the shortcut is installed. For example, to resolve to the correct value for the argument "[#MyDoc.doc]", the same process must be installing the file MyDoc.doc and the component that owns the shortcut.

## Description

The localizable description of the shortcut.

## Hotkey

The hotkey for the shortcut. The low-order byte contains the virtual-key code for the key, and the high-order byte contains modifier flags. This must be a non-negative number. Authors of installation packages are generally recommended not to set this option, because the setting of this option can add duplicate hotkeys to a user's desktop. In addition, the practice of assigning hotkeys to shortcuts can be problematic for users using hotkeys for [accessibility](#).

## Icon\_

The external key to column one of the [Icon table](#).

## IconIndex

The icon index for the shortcut. This must be a non-negative number.

## ShowCmd

The Show command for the application window.

The following values may be used. The values are as defined for the Windows API function ShowWindow.

Value	Meaning
1	SW_SHOWNORMAL
3	SW_SHOWMAXIMIZED
7	SW_SHOWMINNOACTIVE

## WkDir

The name of the property that has the path of the working directory for the shortcut. The value can use the Windows format to reference environment variables, for example %USERPROFILE%. The references are resolved to an actual path when the installer resolves the working directory to create the shortcut.

## DisplayResourceDLL

This field contains a [Formatted](#) string value for the full path to the language-neutral portable executable (LN file) that contains the resource configuration (RC Config) data. The formatted string can use the [#filekey] convention. If this field contains a value, the Name column is ignored. If this field is empty, the installer uses the value in the Name column. When this field contains a value, the DisplayResourceId field is also required to contain a value, or the installation fails.

This column of the Shortcut table is used only when running on Windows Vista or Windows Server 2008 and is otherwise ignored. This column is available with versions not earlier than Windows Installer 4.0.

For information about how to add shortcuts to Shortcut table for use with MUI resources see [A MUI Shortcut Example](#).

#### DisplayResourceId

The display name index for the shortcut. This must be a non-negative number. When this field contains a value, the DisplayResourceDLL field is required to also contain a value or the installation fails.

This column of the Shortcut table is used only when running on Windows Vista or Windows Server 2008 and is otherwise ignored. This column is available with versions not earlier than Windows Installer 4.0.

#### DescriptionResourceDLL

This field contains a [Formatted](#) string value for the full path to the language-neutral portable executable (LN file) that contains the resource configuration (RC Config) data. The formatted string can use the [#filekey] convention. If this field contains a value, the Name column is ignored. If this field is empty, the installer uses the value in the Description column. When this field contains a value, the DescriptionResourceId field is also required to contain a value, or the installation fails.

This column of the Shortcut table is used only when running on Windows Vista or Windows Server 2008 and is otherwise ignored. This column is available with versions not earlier than Windows Installer 4.0.

For information about how to add shortcuts to Shortcut table for use with MUI resources see [A MUI Shortcut Example](#).

#### DescriptionResourceId

The description name index for the shortcut. This must be a non-negative number. When this field contains a value, the DescriptionResourceDLL field is required to also contain a value or the installation fails.

This column of the Shortcut table is used only when running on Windows Vista or Windows Server 2008 and is otherwise ignored. This column is available with versions not earlier than Windows Installer 4.0.

## Remarks

The enabling of a feature creates an advertised shortcut only if the system's IShellLink interface supports installer descriptor resolution. This is supported by Microsoft Windows 2000 and systems running Microsoft Internet Explorer 4.01. If unsupported, the installer creates a non-advertised shortcut upon the installation of the feature's component, either locally or run from source.

Note that advertised shortcuts always point at a particular application, identified by a **ProductCode**, and should not be shared between applications. Advertised shortcuts only work for the most recently installed application, and are removed when that application is removed.

This table is referred to when the [CreateShortcuts action](#) and the [RemoveShortcuts action](#) is executed.

See also the **DISABLEADVTSHORTCUTS** property.

## Validation

[ICE03](#)

[ICE06](#)

[ICE19](#)

[ICE32](#)

[ICE36](#)

ICE46  
ICE50  
ICE57  
ICE59  
ICE67  
ICE69  
ICE80  
ICE90  
ICE91  
ICE94

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Signature Table

The Signature table holds the information that uniquely identifies a file signature. For more information regarding signatures see [Digital Signatures and Windows Installer](#).

The Signature table has the following columns.

Column	Type	Key	Nullable
Signature	Identifier	Y	N
FileName	Text	N	N
MinVersion	Text	N	Y
MaxVersion	Text	N	Y
MinSize	DoubleInteger	N	Y
MaxSize	DoubleInteger	N	Y
MinDate	DoubleInteger	N	Y
MaxDate	DoubleInteger	N	Y
Languages	Text	N	Y

## Columns

### Signature

The Signature column is a unique file signature.

### FileName

The name of the file.

### MinVersion

The minimum version of the file, with a language comparison. If this field is specified, then the file must have a version that is at least equal to MinVersion. If the file has an equal version to the MinVersion field value but the language specified in the Languages column differs, the file does not satisfy the signature filter criteria.

**Note** The language specified in the Languages column is used in

the comparison and there is no way to ignore language. If you want a file to meet the MinVersion field requirement regardless of language, you must enter a value in the MinVersion field that is one less than the actual value. For example, if the minimum version for the filter is 2.0.2600.1183, use 2.0.2600.1182 to find the file without matching the language information.

#### MaxVersion

The maximum version of the file. If this field is specified, then the file must have a version that is at most equal to MaxVersion.

#### MinSize

The minimum size of the file. If this field is specified, then the file under inspection must have a size that is at least equal to MinSize. This must be a non-negative number.

#### MaxSize

The maximum size of the file. If this field is specified, then the file under inspection must have a size that is at most equal to MaxSize. This must be a non-negative number.

#### MinDate

The minimum modification date and time of the file. If this field is specified, then the file under inspection must have a modification date and time that is at least equal to MinDate. This must be a non-negative number. The format of this field is two packed 16-bit values of type **WORD**. The high order **WORD** value specifies the date in MS-DOS date format. The low order **WORD** value specifies the time in MS-DOS time format. A value of 0 for the time value represents midnight. See the Remarks section.

#### MaxDate

The maximum creation date of the file. If this field is specified, then the file under inspection must have a creation date that is at most equal to MaxDate. This must be a non-negative number. The format of this field is two packed 16-bit values of type **WORD**. The high order **WORD** value specifies the date in MS-DOS date format. The low order **WORD** value specifies the time in MS-DOS time format. A value of 0 for the time value represent midnight. See the Remarks section.

#### Languages

The languages supported by the file.

## Remarks

This table is used with the [AppSearch Table](#).

The signature is searched for using the [RegLocator table](#), the [IniLocator table](#), the [CompLocator table](#), and the [DrLocator table](#). This table's columns are generally not localized. If an author decides to search for products in multiple languages, then there can be a separate entry included in the table for each language.

The Signature table generally follows the Windows Installer [File Versioning Rules](#). Languages specified in the Languages column of the Signature table are not evaluated unless the file versions are equivalent. The Languages column will ensure that a file is of a particular language if it is of the requested version. There is no method available to ignore the Languages column. A NULL value entered in the Languages column is treated as a file without a language and does not match the file signature of a file with a language appearing in the Signature table. The following example searches for a particular version of MSI.DLL.

DrLocator table

Signature_	Parent	Path	Depth
MsiDll	{null}	c:\windows\system32	0

AppSearch Table

Property	Signature_
MSIDLL	MsiDll

Signature table

Signature	FileName	MinVersion	MaxVersion	MinSize	MaxSize	MinD
MsiDll	msi.dll	2.0.2600.1106	{null}	{null}	{null}	{null}

In this case, and on Windows XP SP1, the [AppSearch action](#) sets MSIDLL to c:\windows\system32\msi.dll because MSI.DLL is a language neutral file. If the value of the Languages column is changed from 0 to 1033, then the AppSearch action fails to find the matching msi.dll and the MSIDLL property is undefined.

You cannot use the Signature table to query on languages alone. To search for different language versions of a file, you must have a separate entry in the Signature table for each language version. If multiple languages are provided in the Languages column, then the search is for a file that supports all of those languages.

The format of MinDate and MaxDate columns are two packed 16-bit values of type **WORD**.

#### Date **WORD**

Bits	Content
0-4	Day of the month (1-31)
5-8	Month (1 = January, 2 = February, and so on)
9-15	Year offset from 1980 (add 1980 to get actual year)

#### Time **WORD**

Bits	Content
0-4	Seconds divided by 2
5-10	Minutes (0-59)
11-15	Hour(0-23 on 24 hour clock)

The formula for calculating the MinDate and MaxDate field values is:

$( (\text{Year} - 1980) * 512 + \text{Month} * 32 + \text{Day} ) * 65536 + \text{Hours} * 2048 + \text{Minutes} * 32 + \text{Seconds}/2$

## Validation

ICE03  
ICE06

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# TextStyle Table

The TextStyle table lists different font styles used in controls having text.

The TextStyle table has the following columns.

Column	Type	Key	Nullable
TextStyle	Identifier	Y	N
FaceName	Text	N	N
Size	Integer	N	N
Color	DoubleInteger	N	Y
StyleBits	Integer	N	Y

## Columns

### TextStyle

This column is the name of the font style. This name can be embedded in the text string to indicate a style change. Note that the font style name used in this field must not end with the characters: `_UL`. See [Adding Controls and Text](#).

### FaceName

A string indicating the name of the font. The string must be no more than 31 characters long.

### Size

The font size measured in points. This must be a non-negative number.

### Color

This column specifies the text color displayed by a [Text Control](#). All other types of controls always use the default text color. The value put in this column should be computed using the following formula:  $65536 * \text{blue} + 256 * \text{green} + \text{red}$ , where red, green, and blue are each in the range of 0-255. The value must not exceed 16777215, which is the value for white. The value is 0 for black, 255 for red,

65280 for green, 16711680 for blue and 8421504 for grey. Leaving the field empty specifies the default color.

Do not place transparent [Text controls](#) on top of colored bitmaps. The text may not be visible if the user changes the display color scheme. For example, text may become invisible if the user sets the high contrast parameter for accessibility.

## StyleBits

A combination of bits indicating the formatting for the text.

The individual style bits have the following values.

<b>Constant</b>	<b>Hexadecimal</b>	<b>Decimal</b>	<b>Style</b>
msidbTextStyleStyleBitsBold	0x001	1	Bold
msidbTextStyleStyleBitsItalic	0x002	2	Italic
msidbTextStyleStyleBitsUnderline	0x004	4	Underline
msidbTextStyleStyleBitsStrike	0x008	8	Strike out

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE31](#)  
[ICE45](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# TypeLib Table

The TypeLib table contains the information that needs to be placed in the registry registration of type libraries.

The TypeLib table has the following columns.

Column	Type	Key	Nullable
LibID	GUID	Y	N
Language	Integer	Y	N
Component_	Identifier	Y	N
Version	DoubleInteger	N	Y
Description	Text	N	Y
Directory_	Identifier	N	Y
Feature_	Identifier	N	N
Cost	DoubleInteger	N	Y

## Columns

### LibID

The GUID that identifies the library.

### Language

The language of the type library. This must be a non-negative number.

### Component\_

External key into the first column of the [Component table](#). This column identifies the component belonging to Feature\_ whose key file is the type library being registered.

### Version

This is the version of the library. The major and minor versions are encoded in the four byte integer value. The minor version is in the lower eight bits. The major version is in the middle sixteen bits.

## Description

A localizable description of the library.

## Directory\_

External key into the first column of the [Directory table](#). This column identifies the Help path for the type library. This column is ignored during advertising.

## Feature\_

External key into the first column of the [Feature table](#). This column specifies the feature that must be installed for the type library to be operational.

## Cost

The cost associated with the registration of the type library in bytes. This must be a non-negative number or null.

## Remarks

This table is referred to when the [RegisterTypeLibraries action](#) or the [UnregisterTypeLibraries action](#) is executed.

The installer writes all type library registration information into the HKEY\_LOCAL\_MACHINE (HKLM) registry location. This is the case even for per-user installations. Type libraries cannot be registered in per-user locations (HKCU).

Installation package authors are strongly advised against using the TypeLib table. Instead, they should register type libraries by using the [Registry](#) table. Reasons for avoiding self registration include:

- If an installation using the TypeLib table fails and must be rolled back, the rollback may not restore the computer to the same state that existed prior to the rollback. Type libraries registered prior to rollback may not be registered after rollback.

## Validation

[ICE03](#)

ICE06  
ICE19  
ICE32

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UIText Table

The UIText table contains the localized versions of some of the strings used in the user interface. These strings are not part of any other table. The UIText table is for strings that have no logical place in any other table.

The UIText table has the following columns.

Column	Type	Key	Nullable
Key	Identifier	Y	N
Text	Text	N	Y

## Columns

### Key

A unique key that identifies the particular string.

### Text

The localized version of the string.

## Remarks

Some controls use the UIText table as a source of strings. For information about which strings are required in this table for each particular control, see the individual controls in the [User Interface Reference](#).

## Validation

[ICE03](#)

[ICE06](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Upgrade Table

The Upgrade table contains information required during [major upgrades](#). To fully enable the installer's upgrade capabilities, every package should have an [UpgradeCode](#) property and an Upgrade table. Each record in the Upgrade table gives a characteristic combination of upgrade code, product version, and language information used to identify a set of products affected by the upgrade. When the [FindRelatedProducts](#) action detects an affected product installed on the system, it appends the product code to a property specified in the ActionProperty column. The [RemoveExistingProducts](#) action and the [MigrateFeatureStates](#) action only remove or migrate products listed in the ActionProperty column.

The Upgrade table contains the columns shown in the following table.

Column	Type	Key	Nullable
UpgradeCode	<a href="#">GUID</a>	Y	N
VersionMin	<a href="#">Text</a>	Y	Y
VersionMax	Text	Y	Y
Language	Text	Y	Y
Attributes	<a href="#">Integer</a>	Y	N
Remove	<a href="#">Formatted</a>	N	Y
ActionProperty	<a href="#">Identifier</a>	N	N

## Columns

### UpgradeCode

The [UpgradeCode](#) property in this column specifies the upgrade code of all products that are to be detected by the [FindRelatedProducts](#) action.

### VersionMin

Lower boundary of the range of product versions detected by [FindRelatedProducts](#). Enter

msidbUpgradeAttributesVersionMinInclusive in Attributes to include VersionMin in the range. If VersionMin equals an empty string ("") it is evaluated the same as 0. If VersionMin is null, FindRelatedProducts ignores msidbUpgradeAttributesVersionMinInclusive and detects all previous versions. VersionMin and VersionMax must not both be null.

VersionMin must be a valid product version as described for the **ProductVersion** property. Note that Windows Installer uses only the first three fields of the product version. If you include a fourth field in your product version, the installer ignores the fourth field.

#### VersionMax

Upper boundary of the range of product versions detected by the FindRelatedProducts action. Enter msidbUpgradeAttributesVersionMaxInclusive in Attributes to include VersionMax in the range. If VersionMax is an empty string (""), it is evaluated the same as 0. If VersionMax is null, FindRelatedProducts ignores msidbUpgradeAttributesVersionMaxInclusive and detects all product versions greater than (or greater than or equal to) the lower boundary specified by VersionMin and msidbUpgradeAttributesVersionMinInclusive. VersionMin and VersionMax must not both be null.

VersionMax must be a valid product version as described for the **ProductVersion** property. Note that Windows Installer uses only the first three fields of the product version. If you include a fourth field in your product version, the installer ignores the fourth field.

#### Language

The set of languages detected by FindRelatedProducts. Enter a list of numeric language identifiers (LANGID) separated by commas. Enter msidbUpgradeAttributesLanguagesExclusive in Attributes to detect all languages exclusive of those listed in Language. If Language is null or an empty string (""), FindRelatedProducts ignores msidbUpgradeAttributesLanguagesExclusive and detects all languages.

#### Attributes

This column contains bit flags specifying attributes of the Upgrade table.

---

Bit flag name	Decimal	Hexadecimal	A
msidbUpgradeAttributesMigrateFeatures	1	0x001	M st th M a
msidbUpgradeAttributesOnlyDetect	2	0x002	D a n
msidbUpgradeAttributesIgnoreRemoveFailure	4	0x004	C ir fæ p a
msidbUpgradeAttributesVersionMinInclusive	256	0x100	D v th V
msidbUpgradeAttributesVersionMaxInclusive	512	0x200	D v th V
msidbUpgradeAttributesLanguagesExclusive	1024	0x400	D læ th ir c

## Remove

The installer sets the **REMOVE** property to features specified in this column. The features to be removed can be determined at run time. The **Formatted** string entered in this field must evaluate to a comma-delimited list of feature names. For example: [Feature1],[Feature2],[Feature3]. No features are removed if the field contains formatted text that evaluates to an empty string (""). The installer sets

REMOVE=ALL only if the Remove field is empty. Note the difference between an empty string and empty field. If the field is empty, it is null.

### ActionProperty

When the [FindRelatedProducts](#) action detects a related product installed on the system, it appends the product code to the property specified in this field. The property specified in this column must be a public property and the package author must add the property to the [SecureCustomProperties](#) property. Each row in the Upgrade table must have a unique ActionProperty value. After FindRelatedProducts, the value of this property is a list product codes, separated by semicolons (;), detected on the system.

## Validation

[ICE03](#)  
[ICE06](#)  
[ICE46](#)  
[ICE61](#)  
[ICE66](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Verb Table

The Verb table contains command-verb information associated with file name extensions that must be generated as a part of product advertisement. Each row generates a set of registry keys and values.

The Verb table has the following columns.

Column	Type	Key	Nullable
Extension_	Text	Y	N
Verb	Text	Y	N
Sequence	Integer	N	Y
Command	Formatted	N	Y
Argument	Formatted	N	Y

## Columns

### Extension\_

The extension associated with the table row. This column is an external key to the first column of the [Extension table](#).

### Verb

The verb for the command.

### Sequence

The sequence of the commands. Only verbs for which the Sequence column is non-Null are used to prepare an ordered list for the default value of the shell key. The Verb with the lowest value in this column becomes the default verb.

### Command

The localizable text displayed on the context menu.

### Argument

Value for the command arguments.

Note that the resolution of properties in the Argument field is limited. A property formatted as [*Property*] in this field can only be resolved if

the property already has the intended value when the component owning the verb is installed. For example, for the argument "[#MyDoc.doc]" to resolve to the correct value, the same process must be installing the file MyDoc.doc and the component that owns the verb.

## Remarks

This table is referred to when the [RegisterExtensionInfo action](#) or the [UnregisterExtensionInfo action](#) is executed.

## Validation

[ICE03](#)

[ICE06](#)

[ICE32](#)

[ICE46](#)

[ICE69](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Columns Table

The `_Columns` table is a read-only system table that contains the column catalog. It lists the columns for all the tables. You can query this table to find out if a given column exists.

The `_Columns` table has the following columns.

Column	Type	Key	Nullable
Table	Text	Y	N
Number	Integer	Y	N
Name	Text	N	N

## Columns

### Table

The name of the table that contains the column.

### Number

The order of the column within the table.

### Name

The name of the column.

## Remarks

Because the `_Columns` table is a system table that cannot be modified through SQL queries, you cannot obtain the primary keys with the [MsiDatabaseGetPrimaryKeys](#) function or the [PrimaryKeys property](#).

Only persistent columns are stored in the `_Columns` table. To determine if a temporary column exists one would need to create a view using a `SELECT *` statement against the table, then loop through all fields in a record returned by the [MsiViewGetColumnInfo](#) function with the `MSICOLINFO_NAMES` option.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# **\_Storages Table**

The `_Storages` table lists embedded OLE data storages. This is a temporary table, created only when referenced by a SQL statement.

<b>Column</b>	<b>Type</b>	<b>Key</b>	<b>Nullable</b>
Name	Text	Y	N
Data	Binary	N	Y

## **Columns**

### **Name**

A unique key that identifies the storage. The maximum length of Name is 31 characters.

### **Data**

The unformatted binary data.

## **Remarks**

To add an OLE storage to a database, create a new record in the `_Storages` table and enter the name of the storage into the Name column. Use [MsiRecordSetStream](#) to copy data into the Data column of this record. Finally, use [MsiViewModify](#) to insert the record into the `_Storages` table.

Data cannot be read from the `_Storages` table. However, the `_Storages` table can be queried to check for the existence of a specific storage. This means that it is not possible to move an OLE storage from one database to another. You must instead import the original storage file into the new database. To delete an OLE storage, fetch the record containing the binary data, set the Data column in the `_Storages` table to null, and then update the record. An alternative method is to simply delete the record using either [MsiViewModify](#) or a plain SQL query.

To rename an OLE storage, update the Name column of the record.

If a hold is placed on this table using SQL (ALTER TABLE <table> HOLD) or a column is added with HOLD, the table must be released using FREE. Storages are not written until the table has been released or committed.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# \_Streams Table

The \_Streams table lists embedded OLE data streams. This is a temporary table, created only when referenced by a SQL statement.

<b>Column</b>	<b>Type</b>	<b>Key</b>	<b>Nullable</b>
Name	Text	Y	N
Data	Binary	N	Y

## Columns

### Name

A unique key that identifies the stream. The maximum length of Name is 62 characters.

### Data

The unformatted binary data.

## Remarks

To copy an OLE data stream (for example, an object of the [Binary](#) data type) from a file into a database, create a record in the \_Streams table and enter the name of the data stream into the Name column of this record and copy the data from the file into the Data column using [MsiRecordSetStream](#). Use [MsiViewModify](#) to insert the new record into the table.

To read a binary data stream embedded in a database, use a SQL query to find and to fetch the record containing the binary data. Use [MsiRecordReadStream](#) to read the binary data into a buffer.

To move a binary data stream from one database to another, first export the data to a file. Use a SQL query to find the data stream in the file and use [MsiRecordSetStream](#) to copy the data from the file into Data column of \_Streams table of the second database. This ensures that each database has its own copy of the binary data. You cannot move

binary data from one database to another simply by fetching a record with the data from the first database and inserting it into the second database.

To delete a data stream, fetch the record and set the Data column to null before updating the record. Another method is to remove the record from the table, deleting it using either [MsiViewModify](#) or a plain SQL query. A stream should not be fetched into a record if the stream is deleted from the table.

To rename an OLE data stream, update the 'Name' column of the record.

If a hold is placed on this table using SQL (`ALTER TABLE <table> HOLD`) or a column is added with HOLD, the table must be released using `FREE`. Streams are not written until the table has been released or committed.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# **\_Tables Table**

The `_Tables` table is a read-only system table that lists all the tables in the database. Query this table to find out if a table exists.

The `_Tables` Table has the following column.

<b>Column</b>	<b>Type</b>	<b>Key</b>	<b>Nullable</b>
Name	<a href="#">Text</a>	Y	N

## **Column**

Name

Name of one of the tables.

## **Remarks**

Because the `_Tables` table is a system table that cannot be modified through SQL queries, you cannot obtain the primary keys with the [MsiDatabaseGetPrimaryKeys](#) function or the [PrimaryKeys property](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# **\_TransformView Table**

This is a read-only temporary table used to view transforms with the transform view mode. This table is never persisted by the installer.

To invoke the transform view mode, obtain a handle and open the reference database. See [Obtaining a Database Handle](#). Call **MsiDatabaseApplyTransform** with MSITRANSFORM\_ERROR\_VIEWTRANSFORM. This stops the transform from being applied to the database and dumps the transform contents into the \_TransformView table. The data in the table can be accessed using SQL queries. See [Working with Queries](#).

The \_TransformView table is not cleared when another transform is applied. The table reflects the cumulative effect of successive applications. To view the transforms separately you must release the table.

The \_TransformView Table has the following columns.

<b>Column</b>	<b>Type</b>	<b>Key</b>	<b>Nullable</b>
Table	<a href="#">Identifier</a>	Y	N
Column	<a href="#">Text</a>	Y	N
Row	<a href="#">Text</a>	Y	Y
Data	<a href="#">Text</a>	N	Y
Current	<a href="#">Text</a>	N	Y

## **Column**

Table

Name of an altered database table.

Column

Name of an altered table column or INSERT, DELETE, CREATE, or DROP.

Row

A list of the primary key values separated by tabs. Null primary key values are represented by a single space character. A Null value in this column indicates a schema change.

#### Data

Data, name of a data stream, or a column definition.

#### Current

Current value from reference database, or column a number.

## Remarks

The `_TransformView` is held in memory by a lock count, that can be released with the following SQL command.

"ALTER TABLE `_TransformView` FREE".

The data in the table can be accessed using SQL queries. The SQL language has two main divisions: Data Definition Language (DDL) that is used to define all the objects in an SQL database, and Data Manipulation Language (DML) that is used to select, insert, update, and delete data in the objects defined using DDL.

The Data Manipulation Language (DML) transform operations are indicated as follows. Data Manipulation Language (DML) are those statements in SQL that manipulate, as opposed to define, data.

<b>Transform operation</b>	<b>SQL result</b>
Modify data	{table} {column} {row} {data} {current value}
Insert row	{table} "INSERT" {row} NULL NULL
Delete row	{table} "DELETE" {row} NULL NULL

The Data Definition Language (DDL) transform operations are indicated as follows. Data Definition Language (DDL) are those statements in SQL that define, as opposed to manipulate, data.

<b>Transform operation</b>	<b>SQL result</b>
Add column	{table} {column} NULL {defn} {column number}

Add table	{table} "CREATE" NULL NULL NULL
Drop table	{table} "DROP" NULL NULL NULL

When the application of a transform adds this table, the Data field receives text that can be interpreted as a 16-bit integer value. The value describes the column named in the Column field. You can compare the integer value to the constants in the following table to determine the definition of the altered column.

Bit	Description
Bits 0–7	Hexadecimal: 0x0000–0x0100 Decimal: 0–255 Column width
Bit 8	Hexadecimal: 0x0100 Decimal: 256 A persistent column. Zero means a temporary column.
Bit 9	Hexadecimal: 0x0200 Decimal: 1023 A localizable column. Zero means the column cannot be localized.
Bits 10–11	Hexadecimal: 0x0000 Decimal: 0 Long integer Hexadecimal: 0x0400 Decimal: 1024 Short integer Hexadecimal: 0x0800

	Decimal: 2048 Binary object Hexadecimal: 0x0C00 Decimal: 3072 String
Bit 12	Hexadecimal: 0x1000 Decimal: 4096 Nullable column. Zero means the column is non-nullable.
Bit 13	Hexadecimal: 0x2000 Decimal: 8192 Primary key column. Zero means this column is not a primary key.
Bits 14–15	Hexadecimal: 0x4000–0x8000 Decimal: 16384–32768 Reserved

For a script sample that demonstrates the `_TransformView` table, see [View a Transform](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Validation Table

The `_Validation` table is a system table that contains the column names and the column values for all of the tables in the database. It is used during the database validation process to ensure that all columns are accounted for and have the correct values. This table is not shipped with the installer database.

The `_Validation` table has the following columns.

Column	Type	Key	Nullable
Table	Identifier	Y	N
Column	Identifier	Y	N
Nullable	Text	N	N
MinValue	DoubleInteger	N	Y
MaxValue	DoubleInteger	N	Y
KeyTable	Identifier	N	Y
KeyColumn	Integer	N	Y
Category	Text	N	Y
Set	Text	N	Y
Description	Text	N	Y

## Columns

### Table

Used to identify a particular table. This key and the Column key form the primary key of the `_Validation` table.

### Column

Used to identify a particular column of the table. This key and the Table key form the primary key of the `_Validation` table.

### Nullable

Identifies whether the column may contain a Null value.

This column may have one of the following values.

String	Meaning
Y	Yes, the column may have a Null value.
N	No, the column may not have a Null value.

#### MinValue

This field applies to columns having numeric value. The field contains the minimum permissible value. This can be the minimum value for an integer or the minimum value for a date or version string.

#### MaxValue

This field applies to columns having numeric value. The field is the maximum permissible value. This may be the maximum value for an integer or the maximum value for a date or version string.

#### KeyTable

This field applies to columns that are external keys. The field identified in Column must link to the column number specified by KeyColumn in the table named in KeyTable. This can be a list of tables separated by semicolons.

#### KeyColumn

This field applies to table columns that are external keys. The field identified in Column must link to the column number specified by KeyColumn in the table named in KeyTable. The permissible range of the KeyColumn field is 1-32.

#### Category

This is the type of data contained by the database field specified by the Table and Column columns of the \_Validation table. If this is a type having a numeric value, such as [Integer](#), [DoubleInteger](#) or [Time/Date](#), then enter null into this field and specify the value's range using the MinValue and MaxValue columns. Use the Category column to specify the non-numeric data types described in [Column Data Types](#).

## Set

This is a list of permissible values for this field separated by semicolons. This field is usually used for enumerations.

## Description

A description of the data that is stored in the column.

## Validation

[ICE03](#)

[ICE06](#)

[ICE32](#)

## Remarks

The Category field of this table only applies to string data. If the Column field refers to a column with binary data, the binary data type must be specified in the Category field. Integer data Column types ignore the Category field during validation.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Released Versions, Tools, and Redistributables

The following topics provide more information about released versions, tools, and redistributables of Windows Installer.

- [Windows SDK Components for Windows Installer Developers](#) explains where to obtain the installer redistributable components, documentation, database validation tool, database table editor, database schema, development tools, VBScript tools, sample product, and code samples.
- [Windows Installer Redistributables](#) provides information on where to obtain the installer redistributable components.
- [Released Versions of Windows Installer](#) lists the released versions of the installer that have shipped with different operating systems.
- [Windows Installer Development Tools](#) contains reference pages for using the installer development tools.
- [Instmsi.exe](#) describes which Instmsi is compatible on which operating system. InstMsi is the redistributable package for installing the installer.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Windows SDK Components for Windows Installer Developers

The components of the Windows Installer Software Development Kit are included in the Microsoft Windows Software Development Kit (SDK). The Windows SDK includes redistributable components, documentation, installer database validation tool, database table editor, database schema, development tools, Visual Basic Scripting Edition (VBScript) tools, sample product, and code samples.

For the download page for the Windows SDK, see [Microsoft Windows Software Development Kit \(SDK\)](#).

## See Also

[Released Versions, Tools, and Redistributables](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Windows Installer Redistributables

The Windows Installer redistributable is a software update package. See the section [Released Versions of Windows Installer](#) to determine which products shipped versions of the Windows Installer. The redistributable update package for a version is made available after the release of the product that ships with a specific Windows Installer version.

## Obtaining the Windows Installer Redistributable

- You can find all the available Windows Installer redistributables at the [Microsoft Download Center](#).
- The download for the [Windows Installer 4.5 redistributable package](#) is available at: <http://go.microsoft.com/fwlink/?LinkId=101159>.
- The name of the redistributable that installs Windows Installer 4.5 on x86-based computers running Windows Vista, Windows Vista with Service Pack 1 (SP1), and Windows Server 2008 is Windows6.0-KB942288-v2-x86.MSU.
- The name of the redistributable that installs Windows Installer 4.5 on x64-based computers running Windows Vista, Windows Vista with SP1, and Windows Server 2008 is Windows6.0-KB942288-v2-x64.MSU.
- The name of the redistributable that installs Windows Installer 4.5 on Itanium-Based Systems computers running Windows Vista, Windows Vista with SP1, and Windows Server 2008 is Windows6.0-KB942288-v2-ia64.MSU.
- The name of the redistributable that installs Windows Installer 4.5 on x86-based computers running Windows XP with Service Pack 2 (SP2) and Windows XP with Service Pack 3 (SP3) is WindowsXP-KB942288-v3-x86.exe.
- The name of the redistributable that installs Windows Installer 4.5 on x86-based computers running Windows Server 2003 with Service

Pack 1 (SP1) and Windows Server 2003 with Service Pack 2 (SP2) is WindowsServer2003-KB942288-v4-x86.exe.

- The name of the redistributable that installs Windows Installer 4.5 on x64-based computers running Windows Server 2003 with SP1 and Windows Server 2003 with SP2 is WindowsServer2003-KB942288-v4-x64.exe.
- The name of the redistributable that installs Windows Installer 4.5 on Itanium-Based Systems computers running Windows Server 2003 with SP1 and Windows Server 2003 with SP2 is WindowsServer2003-KB942288-v4-ia64.exe.
- There is no redistributable that installs Windows Installer 4.0. This version of the Windows Installer ships with Windows Vista.
- The name of the redistributable that installs Windows Installer 3.1 is WindowsInstaller-KB893803-v2-x86.exe. The download for the [Windows Installer 3.1 Redistributable \(v2\)](#) package is available at: <http://www.microsoft.com/downloads/details.aspx?FamilyID=889482fc-5f56-4a38-b838-de776fd4138c>.

**Note** If you upgraded to Windows Installer 3.1 by installing Windows Server 2003 with SP1, or an earlier version of this redistributable, you may also need to install the [Update for Windows Server 2003 Service Pack 1 \(KB898715\)](#) to obtain all the updates available in [Windows Installer 3.1 Redistributable \(v2\)](#).

- The redistributable that installs Windows Installer 3.0 is WindowsInstaller-KB884016-v2-x86.exe. The download for the [Windows Installer 3.0 Redistributable](#) is available at: <http://www.microsoft.com/downloads/details.aspx?FamilyID=5fbc5470-b259-4733-a914-a956122e08e8>.
- The Windows Installer 2.0 used a previous naming convention for the redistributable: [Instmsi.exe](#). The redistributable for installing or upgrading to Windows Installer 2.0 on Windows 2000 should not be

used to install or upgrade Windows Installer 2.0 on Windows Server 2003 and Windows XP.

The download for the [Windows Installer 2.0 Redistributable for Windows NT 4.0 and Windows 2000](http://www.microsoft.com/downloads/details.aspx?FamilyID=4b6140f9-2d36-4977-8fa1-6f8a0f5dca8f) is available at <http://www.microsoft.com/downloads/details.aspx?FamilyID=4b6140f9-2d36-4977-8fa1-6f8a0f5dca8f>.

## Installing the Windows Installer Redistributable

The Windows Installer 4.5 redistributable is provided for Windows Vista and Windows Server 2008 operating systems as a .msu file and should be installed using the [Windows Update Stand-alone Installer](#) (Wusa.exe.)

The Windows Installer 4.5 redistributable for Windows XP and Windows Server 2003 operating systems can be installed using the following command line syntax and options.

The Windows Installer 3.1 and Windows Installer 3.0 redistributables can be installed using the following command line syntax and options.

### Syntax

Use the following syntax to install the redistributables for Windows Installer 4.5 on Windows XP and Windows Server 2003.

*<Name of the Redistributable>[<options>]*

### Command-Line Options

The Windows Installer redistributable software update packages use the following case-insensitive command-line options.

Option	Description
/norestart	Prevents the redistributable package from asking the user to reboot even if it had to replace files that were in use during the installation. If the update package is invoked with this option, it returns ERROR_SUCCESS_REBOOT_REQUIRED if it had to replace

	<p>files that were in use.</p> <p>If it did not have to replace files that were in use, it returns ERROR_SUCCESS. See the remarks section for additional information on delayed reboots.</p>
/quiet	<p>For use by applications that redistribute the Windows Installer as part of a bootstrapping application.</p> <p>A user interface (UI) is not presented to the user. The bootstrapping application should check the return code to determine whether a reboot is needed to complete the installation of the Windows Installer.</p>
/help	<p>Displays help on all the available options.</p>

## Delayed Restart on Windows Vista and Windows Server 2008

The /norestart command-line option prevents wusa.exe from restarting the computer. However, if a file being updated by the MSU package is in use, then the package is not applied to the computer until the user restarts the computer. This means that applications that use the Windows Installer 4.5 redistributable for Windows Vista and Windows Server 2008 cannot use the Windows Installer 4.5 functionality until the computer is restarted.

## Delayed Restart on Windows XP and Windows Server 2003

It is recommended that the Windows Installer service be stopped when using the update package. When the package is run in full UI mode it detects if the Windows Installer service is running and requests the user to stop the service. If the user continues without stopping the service, the update replaces Windows Installer.

[Bootstrapping](#) applications that use the redistributable package to install the Windows Installer with another application can require an extra system reboot in addition to reboots needed to install the application. The delayed reboot option is only recommended for cases where it is

necessary to eliminate an extra reboot caused by installing files that are in use. Developers should do the following in their setup application to use the delayed reboot option.

- Call the redistributable package with the /norestart command-line option.
- Treat the return of either ERROR\_SUCCESS or ERROR\_SUCCESS\_REBOOT\_REQUIRED as meaning success.
- Invoke Msiexec on the application's package and run other setup code specific to the application. If the setup application uses **MsiInstallProduct**, then the application must load MSI.DLL from the system directory. If no reboot occurs and if the redistributable returned ERROR\_SUCCESS\_REBOOT\_REQUIRED, then prompt the user for a reboot to complete the setup of the Windows Installer binaries. If a reboot occurs, no additional steps are required.

**Note** Applications that call **LoadLibrary** on the new MSI.DLL after the redistributable package returns success must ensure that an older version of MSI.DLL has not already been loaded within the process. If an older version of MSI.DLL was loaded, it must be unloaded from the process address space prior to calling **LoadLibrary** for the new MSI.DLL.

For more information, see [Windows Installer Bootstrapping](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Released Versions of Windows Installer

The table in this topic identifies the released versions of the Windows Installer. For more information, see [Operating System Property Values](#).

Release	Version	Description
Windows Installer 2.0	2.0.2600.0	Released with Windows XP.
Windows Installer 2.0	2.0.2600.1	Released with Windows 2000 Server with Service Pack 3 (SP3).
Windows Installer 2.0	2.0.2600.1183	Released with Windows 2000 Server with Service Pack 4 (SP4).
Windows Installer 2.0	2.0.2600.2	Released as a redistributable.
Windows Installer 2.0	2.0.2600.1106	Released with Windows XP with Service Pack 1 (SP1).
Windows Installer 2.0	2.0.3790.0	Released with Windows Server 2003.
Windows Installer 3.0	3.0.3790.2180	Released with Windows XP with Service Pack 2 (SP2). Released as a redistributable.
Windows Installer 3.1	3.1.4000.1823	Released as a redistributable. This version is has the same functionality as version 3.1.4000.2435.
Windows Installer 3.1	3.1.4000.1830	Released with Windows Server 2003 with Service Pack 1 (SP1) and Windows XP Professional x64 Edition. Update this version to version 3.1.4000.2435 to address the issue discussed in <a href="#">KB898628</a> .
Windows Installer 3.1	3.1.4000.3959	Released with Windows Server 2003 with Service Pack 2 (SP2).
Windows	3.1.4000.2435	Released with a fix to address the issue

Installer 3.1		discussed in <a href="#">KB898628</a> . This is the latest version of Windows Installer 3.1.
Windows Installer 3.1	3.1.4001.5512	Released with Windows XP with Service Pack 3 (SP3).
Windows Installer 4.0	4.0.6000.16386	Released with Windows Vista.
Windows Installer 4.0	4.0.6001.18000	Released with Windows Vista with Service Pack 1 (SP1) and Windows Server 2008.
Windows Installer 4.5	4.5.6002.18005	Released with Windows Vista with Service Pack 2 (SP2) and Windows Server 2008 with Service Pack (SP2.)
Windows Installer 4.5	4.5.6000.20817	Released as a redistributable for Windows Vista.
Windows Installer 4.5	4.5.6001.22162	Released as a redistributable for Windows Server 2008 and Windows Vista with SP1.
Windows Installer 4.5	4.5.6001.22159	Released as a redistributable for Windows XP with Service Pack 2 (SP2) and later, and Windows Server 2003 with SP1 and later.

Windows Installer 5.0 will be released with Windows Server 2008 R2 and Windows 7.

For information about how to determine the Windows Installer version, see [Determining the Windows Installer Version](#).

For a list of changes in each of the Windows Installer versions, see [What's New in Windows Installer](#).

For information about obtaining the latest Windows Installer redistributable, see [Windows Installer Redistributables](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Windows Installer Development Tools

The following tools are only available in the [Windows SDK Components for Windows Installer Developers](#).

Utility	Description
<a href="#">Instmsi.exe</a>	Redistributable package for installing the Windows Installer on Windows operating systems earlier than Windows Me.
<a href="#">Msicert.exe</a>	Populates the MsiDigitalSignature table and MsiDigitalCertificate table with the digital signature information belonging to external cabinet files in the Media table.
<a href="#">Msidb.exe</a>	Imports and exports database tables and streams, merges databases, and applies transforms.
<a href="#">Msifiler.exe</a>	Populates the File table with file versions, languages, and sizes based upon a source directory. It can also update the MsiFileHash table with file hashes.
<a href="#">Msiinfo.exe</a>	Edits or displays <a href="#">summary information stream</a> .
<a href="#">Msimerg.exe</a>	Merges one database into another.
<a href="#">Msimsp.exe</a>	Patch creation tool. The recommended method for generating a patch package is to use a patch creation tool such as Msimsp.exe with PATCHWIZ.DLL.
<a href="#">Msistuff.exe</a>	Displays or configures the resources in the Setup.exe bootstrap executable.
<a href="#">Msitool.mak</a>	Makefile that can be used to make tools and custom actions.
<a href="#">Msitrans.exe</a>	Generates a transform or applies a transform file to a database.
<a href="#">Msival2.exe</a>	Runs one or a suite of <a href="#">Internal Consistency Evaluators - ICEs</a> .

<a href="#">Msizap.exe</a>	Removes Windows Installer information for a product or all products installed on a machine.
<a href="#">Orca.exe</a>	Database editor. Creates and edits .msi files and merge modules.
<a href="#">PATCHWIZ.DLL</a>	Generates a Windows Installer <a href="#">patch package</a> from a patch creation properties file (.pcp file). The recommended method for generating a patch package is to use a patch creation tool such as Msimsp.exe with PATCHWIZ.DLL.
<a href="#">Willogutl.exe</a>	Assists the analysis of log files from a Windows Installer installation and displays suggested solutions to errors.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Instmsi.exe

Instmsi.exe is the redistributable package for installing Windows Installer 2.0, and earlier versions of Windows Installer. See [Windows Installer Redistributables](#) for the redistributables for Windows Installer 3.0 and later versions.

For more information about which version of the Windows Installer was shipped with your operating system, see [Released Versions of Windows Installer](#).

Some redistributables should not be run on certain versions of the operating system. The following table describes which Instmsi is compatible on which operating system.

<b>If Instmsi.exe installs this version of the Windows Installer</b>	<b>Instmsi.exe can be run on these operating systems</b>	<b>Instmsi.exe must not be run on these operating systems</b>
Windows Installer version 1.0	Windows 95, Windows 98, Windows NT 4.0+SP3	Windows Me, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008
Windows Installer version 1.1	Windows 95, Windows 98, Windows NT 4.0+SP3	Windows Me, Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008
Windows Installer version 1.2	Windows 95, Windows 98, Windows Me, Windows NT 4.0+SP3	Windows 2000, Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008
Windows Installer version 2.0	Windows 95, Windows 98, Windows Me, Windows NT 4.0+SP6, Windows 2000	Windows XP, Windows Server 2003, Windows Vista, Windows Server 2008

For example, an application that redistributes Windows Installer version 1.1 should check that the operating system is Windows NT 4.0 SP3 or Windows 98/95 before running the redistributable package. Applications using the redistributable package should also ensure that the ANSI version of the Windows Installer is installed on Windows 98/95, and that the Unicode version is installed on Windows NT or Windows 2000. Note that some applications rename the Unicode version to InstMsiW.

## Syntax

*instmsi options*

## Command Line Options

The command line options are case-insensitive.

Option	Description
/q	For use by applications that redistribute the Windows Installer as part of a bootstrapping application. No UI is presented to the user. The bootstrapping application should check the return code to determine whether a reboot is needed to complete the installation of the Windows Installer.
/t	Used for debugging purposes only.
/c:"msiinst /delayreboot"	The delayed reboot option. Prevents Instmsi from prompting the user for a reboot even if it had to replace files that were in use during the installation. If Instmsi is invoked with this option, it returns <code>ERROR_SUCCESS_REBOOT_REQUIRED</code> if it had to replace files that were in use. If it did not have to replace files that were in use, it returns <code>ERROR_SUCCESS</code> . Available with Instmsi for Windows Installer 2.0 or later. See the remarks section for additional information on delayed reboots.
/c:"msiinst	The quiet version of the delayed reboot option. It does not

/delayrebootq"	present any UI to the user. Otherwise the behavior is identical to the previous option. Available with Instmsi for Windows Installer 2.0 or later. See the remarks section for additional information on delayed reboots.
/?	Displays help.

## Remarks

**Bootstrapping** applications that use Instmsi.exe to install the Windows Installer with another application may require an extra system reboot. This is potentially an extra reboot in addition to any reboots that are needed to install the application.

The delayed reboot option is only recommended for setup developers who want to eliminate an extra reboot caused by using Instmsi.exe with a setup application that installs files that are in use.

Developers should do the following in their setup application to use the delayed reboot option. This option is not available with Instmsi.exe versions that install Window Installer versions earlier than version 2.0:

### ► To use the delayed reboot option

1. Call Instmsi.exe with one of the delayed reboot command line options.
2. Treat the return of either `ERROR_SUCCESS` or `ERROR_SUCCESS_REBOOT_REQUIRED` as meaning success.
3. Get the path to the folder containing the newly installed Windows Installer binaries from the `InstallerLocation` value under:  
**HKEY\_LOCAL\_MACHINE\Software\Microsoft\Windows\Curren**  
This value is of type **REG\_SZ**.
4. Set the current directory to the path obtained in step 3.
5. Invoke Msiexec on the application's package and run other setup code specific to the application. If the setup application uses

**MsiInstallProduct**, then the application must load MSI.DLL from the location obtained in step 3.

**Note** Applications that call **LoadLibrary** on the new MSI.DLL in the location obtained in step 3 must ensure that an older version of MSI.DLL has not already been loaded within the process. If an older version of MSI.DLL was loaded within the process, it must be unloaded from the process address space prior to the **LoadLibrary** call for the new MSI.DLL.

6. If step (5) does not require a reboot and if Instmsi.exe had returned ERROR\_SUCCESS\_REBOOT\_REQUIRED in step (1), prompt the user for a reboot to complete the setup of the Windows Installer binaries on the system. However, if a reboot occurs in step (5), no additional steps are required.

Instmsi.exe is available in the [Windows SDK Components for Windows Installer Developers](#).

## See Also

[Bootstrapping](#)

[Internet Download Bootstrapping](#)

[Released Versions, Tools, and Redistributables](#)

[Windows Installer Development Tools](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Msicert.exe

Windows Installer can use digital signatures as a means to detect corrupted resources. A signer certificate may be compared to the signer certificate of an external resource to be installed by the package. For more information, see [Digital Signatures and Windows Installer](#).

MsiCert.exe is a command line utility that can be used to populate the [MsiDigitalSignature table](#) and [MsiDigitalCertificate table](#) with the digital signature information of an external cabinet file. The cabinet file must be digitally signed and listed in the [Media table](#). MsiCert.exe uses the signer certificate information from the digitally signed cabinet and will create and add the MsiDigitalSignature and MsiDigitalCertificate tables to the database if they do not already exist.

## Syntax

```
msicert -d {database} -m {media entry} -c {cabinet} [-h]
```

## Command Line Options

The command line options are case-insensitive and slash delimiters may be used instead of a dash.

Option	Parameter	Description
-d	<database>	The database (.msi file) that is being updated.
-m	<media Id>	The entry in the DiskId field of the <a href="#">Media table</a> in the record for the cabinet file.
-c	<cabinet>	The path to the digitally signed cabinet file.
-h		Include the hash of the digital signature. This is optional.

This tool is only available in the [Windows SDK Components for Windows Installer Developers](#).

## See Also

[Windows Installer Development Tools  
Released Versions, Tools, and Redistributables](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Msidb.exe

Msidb.exe uses **MsiDatabaseImport** and **MsiDatabaseExport** to import and export **database tables** and streams.

If the mode, folder, database and table list are specified on the command line, Msidb.exe does not bring up any user interface and operates as a silent command-line utility suitable for build script.

## Syntax

**MsiDb** {option}...{option}... {table}...{table}

## Command Line Options

Msidb.exe uses the following case-insensitive command line options. A slash delimiter may also be used in place of a dash.

Option	Description
-i	Import text archive files from folder into database. Table names for import are file names 8 characters long with an ".idt" extension. Longer names are truncated to 8 characters if supplied by command for import. Standard wild card specifications may be used.
-e	Export selected tables from database into text archive files in folder. Table names for export are table names. Only the wildcard specification, "*", may be used. Tables may be exported from a read only database.
-c	Creates a new database file and imports tables. Overwrites an existing database file.
-f	Specifies the folder containing the text archive files for tables and streams. If the folder containing the text archive files is not specified, the utility prompts the user for the folder.
-d	Fully-qualified path to the database file.
-m	Fully-qualified path to the database that is to be merged in. This option is available only in silent command line mode. Multiple instances of this option may occur to a maximum of 10. If the

	database is not specified on the command-line, the utility prompts the user for the database.
-t	Fully-qualified path to the transform to be applied. This option is available only in silent command line mode. Multiple instances of this option may occur to a maximum of 10.
-j	Name of storage to remove from the database. This option is available only in silent command line mode. Multiple instances of this option may occur to a maximum of 10.
-k	Name of stream to remove from the database. This option available only in silent command line mode. Multiple instances of this option may occur to a maximum of 10.
-x	Name of stream to save to a disk file in the current directory. This option is available only in silent command line mode. Binary data streams are stored as separate files with the extension ".ibd". Binary filename used is primary key data for the row containing the stream.
-w	Name of storage to save to a disk file in the current directory. This option is available only in silent command line mode.
-a	Name of file to add to the database as a stream. This option is available only in silent command line mode. Multiple instances of this option may occur to a maximum of 10. Binary data streams are stored as separate files with the extension ".ibd". Binary filename used is primary key data for the row containing the stream.
-r	Name of storage to add to the database as a substorage. This option available only in silent command line mode. Multiple instances of this option may occur to a maximum of 10.
-s	Truncate table names to 8 characters on export to an .idt. The table name is truncated to 8 characters and the extension ".idt" is added.
-?	Displays the command line help dialog

**Note** When using long filenames with spaces, use quotes around them. For example, for a database that is in the "My Documents" folder, specify it as "c:\my documents".

This tool is only available in the [Windows SDK Components for Windows](#)

[Installer Developers.](#)

## **See Also**

[Windows Installer Development Tools  
Released Versions, Tools, and Redistributables](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Msifiler.exe

MsiFiler.exe populates the [File table](#) with file versions, languages, and sizes based upon a source directory. It can also update the [MsiFileHash table](#) with file hashes.

## Syntax

```
msifiler.exe -d {database} [-v] [-h] [-s ALTSOURCE]
```

## Command Line Options

MsiFiler.exe uses the following case-insensitive command line options. A slash delimiter may also be used in place of a dash.

Option	Parameter	Description
-d	<i>database</i>	The database (.msi file) that is to be updated.
-v		Use verbose mode.
-h		Populate the <a href="#">MsiFileHash table</a> . This creates the table if it does not already exist. The MsiFileHash table can only be used with unversioned files.
-s	<i>ALTSOURCE</i>	ALTSOURCE specifies an alternative directory to find the files.

This tool is only available in the [Windows SDK Components for Windows Installer Developers](#).

## See Also

[Windows Installer Development Tools](#)  
[Using the Directory Table](#)  
[Released Versions, Tools, and Redistributables](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Msiinfo.exe

Msiinfo.exe is a command line utility that uses [Database Functions](#) and [Installer Functions](#) to edit or display the [summary information stream](#) of a database.

## Syntax

**MsiInfo** {*database*}[[/B] /D]{*option*}{*data*}

- The database cannot be a read-only database.
- If no data follows an option, the corresponding property is removed.
- A maximum of 20 options may be specified on the command line. The same properties can be specified multiple times.
- If the data contains a space, enclose the data with quote marks. For example, such as /T "MY TITLE".

## Command Line Options

Msiinfo.exe uses the following case-insensitive command line options. A slash delimiter may also be used in place of a dash. For more information see [Summary Information Stream Property Set](#).

Option	Description	Property ID	PID
<i>none</i>	If no options are specified, the utility displays the current values of the Summary Information properties.		
-b	Display information about every string in the string pool. The -b option is only valid if -d is also used and -b must come before the -d option.		
-d	Display information about the string pool. Validates the string pool and		

	provides information about the codepage of the database. Note that the codepage of the database is different than the codepage of the Summary Information stream (PID_CODEPAGE). This option also checks every string for characters that are invalid in the codepage of the database.		
-i	Not applicable. Reserved.	PID_DICTIONARY	0
-c	Sets the <b>Codepage Summary Property</b> .	PID_CODEPAGE	1
-t	Sets the <b>Title Summary Property</b> .	PID_TITLE	2
-j	Sets the <b>Subject Summary Property</b> .	ID PID_SUBJECT	3
-a	Sets the <b>Author Summary Property</b> .	PID_AUTHOR	4
-k	Sets the <b>Keywords Summary Property</b> .	PID_KEYWORDS	5
-o	Sets the <b>Comments Summary Property</b> .	PID_COMMENTS	6
-p	Sets the <b>Template Summary Property</b> .	PID_TEMPLATE	7
-l	Sets the <b>Last Saved By Summary Property</b> .	PID_LASTAUTHOR	8
-v	Sets the <b>Revision Number Summary Property</b> .	PID_REVNUMBER	9
-e	Not applicable. Reserved.	PID_EDITTIME	10
-s	Sets the <b>Last Printed Summary Property</b> . To specify the year, month, day, hour, minute, and second, use the format "yyyy/mm/dd hh:mm:ss." For	PID_LASTPRINTED	11

	example, "1997/06/20 03:25:59".		
-r	Sets the <b>Create Time/Date Summary Property</b> . To specify the year, month, day, hour, minute, and second, use the format "yyyy/mm/dd hh:mm:ss." For example, "1997/06/20 03:25:59".	PID_CREATE_DTM	12
-q	Sets the <b>Last Saved Time/Date Summary Property</b> . To specify the year, month, day, hour, minute, and second, use the format "yyyy/mm/dd hh:mm:ss." For example, "1997/06/20 03:25:59".	PID_LASTSAVE_DTM	13
-g	Sets the <b>Page Count Summary Property</b> .	PID_PAGECOUNT	14
-w	Sets the <b>Word Count Summary Property</b> .	PID_WORDCOUNT	15
-h	Sets the <b>Character Count Summary Property</b> .	PID_CHARCOUNT	16
	Not applicable. Reserved.	PID_THUMBNAIL	17
-n	Sets the <b>Creating Application Summary Property</b> .	PID_APPNAME	18
-u	Sets the <b>Security Summary Property</b> .	PID_SECURITY	19

This tool is only available in the [Windows SDK Components for Windows Installer Developers](#).

## See Also

[Windows Installer Development Tools Released Versions, Tools, and Redistributables](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Msimerg.exe

Msimerg.exe is a command line utility that uses [MsiDatabaseMerge](#) to [merge](#) a reference database into a base database.

This tool is only available in the [Windows SDK Components for Windows Installer Developers](#).

## Syntax

**Msimerg** *{base database}{reference database}*

If there is a merge conflict between the two databases, the conflict information is placed in the `_MergeErrors` table.

## Command Line Options

None.

## See Also

[Windows Installer Development Tools Released Versions, Tools, and Redistributables](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Msimsp.exe

The recommended method for generating a patch package is to use patch creation tools such as Msimsp.exe and [Patchwiz.dll](#). The Msimsp.exe tool is only available in the [Windows SDK Components for Windows Installer Developers](#).

Msimsp.exe is an executable file that calls Patchwiz.dll. The tool can be used to create a patch package by passing in the path to a patch creation properties file (.pcp file) and the path to the patch package that is being created. Msimsp.exe can also be used to create a log file and to specify a temporary folder in which the transforms, cabinets, and files that are used to create the patch package are saved.

The command-line syntax for Msimsp.exe is:

**Msimsp.exe -s [path to .pcp file] -p [path to .msp file] {options}**

The command-line options are not case-sensitive, and slash delimiters can be used instead of a dash. If no options are specified, Msimsp.exe displays the current values of the summary Information properties.

**-s***[path to .pcp file]*

This is required and must be followed by the path to the patch creation properties file (.pcp extension). For more information, see [PatchWiz.dll](#).

**-p***path to .msp file*

This is required and followed by the path to patch package that is being created (.msp extension).

**-f***path to temporary folder*

Optional. Followed by path to temporary folder. The default location is %TMP%\~pcw\_tmp.tmp\.

**-k**

Optional. Fail if the temporary folder already exists.

**-l***path to log file*

Optional. Followed by the path to the log file that describes the patch creation process and errors. For more information, see [Return Values for UiCreatePatchPackage](#).

**-lp***path to log file with performance data*

Optional. Followed by the path to the log file that describes the patch creation process and errors. This option writes performance data to log file. This option requires version 4.0 of Patchwiz.dll.

**-d**

Optional. Displays a dialog if the patch creation completes successfully.

**-?**

Displays command-line help.

**Note** Msimsp.exe can fail when it calls Makecab.exe if there are values in the File column of the [File table](#) of the installation package that differ only by case. Windows Installer is case-sensitive and allows an installation package such as in the table below only when Comp1 and Comp2 are installed into different directories. However, in this scenario you cannot use Msimsp.exe or Patchwiz.dll to generate a patch for the package, because Msimsp.exe and Patchwiz.dll call Makecab.exe, which is case-insensitive.

Avoid authoring an installation package such as the following partial File table.

<b>File</b>	<b>Component_</b>	<b>FileName</b>
readme.txt	Comp1	readme.txt
ReadMe.txt	Comp2	readme.txt

## See Also

[Creating a Patch Package](#)

[A Small Update Patching Example](#)

[Windows Installer Development Tools](#)

[Released Versions, Tools, and Redistributables](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Msistuff.exe

Msistuff.exe is a command line utility that can be used to display or configure the resources in the Setup.exe bootstrap executable.

This tool is only available in the [Windows SDK Components for Windows Installer Developers](#).

## Syntax

**msistuff setup.exe** *option{value}*

If no data is specified following an option, that resource is removed.

## Command Line Options

Msistuff.exe uses the following case-insensitive command line options. A slash delimiter may also be used in place of a dash. If an option is listed multiple times, only the last occurrence is used.

Option	Resource ID	Description
no options specified		Display configurable resources in Setup.exe.
<b>-u</b>	ISETUPPROPNAME_BASEURL	Set BaseURL, the base URL for Setup.exe. If no value is provided, the location of Setup.exe default is used. Only UAC-enabled removable media. Only UAC-enabled installations are subject to a change in <b>WinVerifyTrust</b> . The trailing slash in the URL is optional. This option is omitted.
<b>-d</b>	ISETUPPROPNAME_DATABASE	Set Msi, the name of the .msi file, as a relative path to the .msi file, relative to the location of the Setup.exe. This option is required if the -u option is not specified. The -d and -u options are mutually exclusive. They

		specified.
<b>-m</b>	ISSETUPPROPNAME_PATCH	Set Msp, the name of the is a relative path to the .m relation to the location of program. This option is re option is not specified. Th options are mutually excl cannot both be specified.
<b>-n</b>	ISSETUPPROPNAME_PRODUCTNAME	Set Product Name, the na product. This provides the the banner text for the do interface. This option may omitted, the default is "th
<b>-o</b>	ISSETUPPROPNAME_OPERATION	Specify the type of operat The valid values are INST MINPATCH, MAJPATCH INSTALLUPD. For addit information on these optio <a href="#">Download Bootstrapping</a> .
<b>-v</b>	ISSETUPPROPNAME_MINIMUM_MSI	Set Minimum Msi Version version of Windows Insta the computer. If the minir the Windows Installer is r the machine, the appropri is installed to upgrade the Installer. The value of this the same format as the PID_PAGECOUNT value <a href="#">Count Summary</a> Propert must be at least 200, the v Windows Installer version option is required.
<b>-i</b>	ISSETUPPROPNAME_INSTLOCATION	Set InstMsi URL Location location of Windows Insta executables. If this value : location of the upgrade ex

		defaults to the location of option may be omitted.
<b>-a</b>	ISETUPPROPNAME_INSTMSIA	Set InstMsiA, the name of version of Windows Installer executable. This is a relative ANSI version of Instmsi.c the location specified by ISETUPPROPNAME_IN This option is required.
<b>-w</b>	ISETUPPROPNAME_INSTMSIW	Set InstMsiW, the name of version of Windows Installer executable. This is a relative Unicode version of Instm: the location specified by ISETUPPROPNAME_IN This option is required.
<b>-p</b>	ISETUPPROPNAME_PROPERTIES	Set the PROPERTY=VALUE These are the PROPERTY to include on the command option may be omitted. They cannot be listed multiple times must be listed last on the command line. All of the command line flags considered as a part of the

## See Also

[Windows Installer Development Tools](#)

[Internet Download Bootstrapping](#)

[A URL Based Windows Installer Installation Example](#)

[Released Versions, Tools, and Redistributables](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Msitool.mak

Msitool.mak is a makefile that you can use to build tools and custom actions. It may be used with Visual C++ version 4.0 or later. For more information, see the header file. It requires a set of values to be defined before including this file. Typically developers put those at the start of the .cpp, ifdef'd to be skipped by the C compiler. Likewise resources are added to the end of the .cpp file. See samples for details.

VCBIN can be defined to the directory where the Visual C++ tools are found or the makefile uses MSVCDIR and MSDEVDIR. If those are not defined, it does not specify a location, assuming the tools to be on the PATH. An issue with the environment variables in Visual C++ version 5.0 is that if both are not on your path, the linker cannot find Msdis100.dll. If you copy that from MSDEVDIR to MSVCDIR, it works. The other option is to copy Msdis100.dll and Rc.exe and Rcdll.dll to MSVCDIR, and set VCBIN to that.

This tool is only available in the [Windows SDK Components for Windows Installer Developers](#).

## See Also

[Windows Installer Development Tools Released Versions, Tools, and Redistributables](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Msitran.exe

Msitran.exe uses [MsiDatabaseGenerateTransform](#), [MsiCreateTransformSummaryInfo](#), and [MsiDatabaseApplyTransform](#) to generate or apply a transform file.

This tool is only available in the [Windows SDK Components for Windows Installer Developers](#).

## Syntax

Use the following syntax to generate a transform.

```
msitran -g {base db}{ref db}{transform file name}{{error conditions / validation conditions}}
```

Use the following syntax to apply a transform

```
msitran -a {transform}{database}{{error conditions}}
```

## Command Line Options

Msitran.exe uses the following case-insensitive command line options. A slash delimiter may also be used in place of a dash.

Option	Description
-g	Transform generation.
-a	Transform application.

The following errors may be suppressed when applying a transform. To suppress an error, include the appropriate character in the {error conditions} argument. Conditions specified with -g are placed in the summary information of the transform, but are not used when applying a transform with -a. For information, see [MsiDatabaseApplyTransform](#).

Option	Suppressed error
a	Add existing row.
b	Delete non-existing row.

c	Add existing table.
d	Delete non-existing table.
e	Modify existing row.
f	Change codepage.

The following validation conditions may be used to indicate when a transform may be applied to a package. These conditions may be specified with -g, but not -a.

Option	Validation condition
g	Check upgrade code.
l	Check language.
p	Check platform.
r	Check product.
s	Check major version only.
t	Check major and minor versions only.
u	Check major, minor, and upgrade versions.
v	Applied database version < Base database version.
w	Applied database version <= Base database version.
x	Applied database version = Base database version.
y	Applied database version >= Base database version.
z	Applied database version > Base database version.

## See Also

[Windows Installer Development Tools](#)

Database Transforms  
A Customization Transform Example  
Released Versions, Tools, and Redistributables

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Msival2.exe

Msival2.exe is a command line utility that can run a suite of [Internal Consistency Evaluators - ICEs](#).

This tool is only available in the [Windows SDK Components for Windows Installer Developers](#).

For more information about ICEs and the CUB file, see [Using Internal Consistency Evaluators](#).

## Syntax

```
Msival2 {database}{CUB file}[-f][-l {logfile}][-i {ICE Id}[:{ICE Id}...]]
```

## Command Line Options

Msival2.exe uses the following case-insensitive command line options. A slash delimiter may also be used in place of a dash.

Option	Description
-f	Filter out all informational messages from the displayed results. All other types of messages are displayed.
-i	Run only the ICEs listed on the command line in the order specified. Each ICE custom action should be listed as it appears in the <a href="#">CustomAction table</a> of the CUB file. If this option is omitted, the tool runs the default set of ICEs specified by the author of the CUB file.
-l	Write results to the specified file. The file must not already exist. If the file exists, it is not overwritten.

## See Also

[Windows Installer Development Tools](#)  
[Internal Consistency Evaluators - ICEs](#)  
[Released Versions, Tools, and Redistributables](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Msizap.exe

Msizap.exe is a command line utility that removes either all Windows Installer information for a product or all products installed on a computer. Products installed by the installer may fail to function after using Msizap.

On Windows 2000 and Windows XP, administrative privileges are required to use Msizap.exe.

This tool is only available in the [Windows SDK Components for Windows Installer Developers](#) and should not be redistributed. Use the recent version of Msizap.exe (version 3.1.4000.2726 or greater) that is available in the Windows SDK Components for Windows Installer Developers for Windows Vista or greater. Lesser versions of Msizap.exe can remove information about all updates that have been applied to other applications on the user's computer. If this information is removed, these other applications may need to be removed and reinstalled to receive additional updates.

## Syntax

**msizap T[WA!]{product code}**

**msizap T[WA!]<msi package>**

**msizap \*[WA!] ALLPRODUCTS**

**msizap PWSA?!**

## Command Line Options

Msizap.exe uses case-insensitive command line options shown in the following table.

Option	Description
*	Removes all Windows Installer folders and registry keys, adjusts shared DLL counts, and stops Windows Installer service. Also removes the In-Progress key and rollback information.
a	Only changes ACLs to Admin Full Control for any specified removal.

g	For all users, removes any cached Windows Installer data files that have been orphaned.
p	Removes the In-Progress key.
s	Removes Rollback Information.
t	Removes all information for the specified product code. When using this option, enclose the Product Code in curly braces. This option may be used with either the full path to the .msi file or with the product code.
w	Removes Windows Installer information for all users. When this option is not set, only the information for the current user is removed. Use of this option requires that the user's profile be loaded so that the user's per-user registry hive be available.
?	Verbose help.
!	Forces a 'yes' response to any prompt.

## See Also

[Released Versions, Tools, and Redistributables](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Orca.exe

Orca.exe is a database table editor for creating and editing Windows Installer packages and merge modules. The tool provides a graphical interface for validation, highlighting the particular entries where validation errors or warnings occur.

This tool is only available in the [Windows SDK Components for Windows Installer Developers](#). It is provided as an Orca.msi file. After installing the Windows SDK Components for Windows Installer Developers, double click Orca.msi to install the Orca.exe file.

## Syntax

`orca [<options>][<source file>]`

## Command Line Options

Orca.exe uses the following case-insensitive command line options. A slash delimiter may also be used in place of a dash.

Option	Description
-q	Quiet mode
-s	<i>&lt;database&gt;</i> Schema database ["orca.dat" - default]
-?	Help dialog

Orca.exe uses the following case-insensitive command line options with merge modules. A slash delimiter may also be used in place of a dash. When performing a merge the -f, -m and *<sourcefile>* are all required.

Option	Description
-c	Commit merge to database if no errors.
-!	Commit merge to a database even if there are errors.
-m	<i>&lt;module&gt;</i> Merge Module to merge into database.
-f	Feature[:Feature2] Feature(s) to connect to Merge Module.

-r	< <i>directory id</i> > Directory entry for the module root redirection.
-x	< <i>directory</i> > Extract files to an image under the directory.
-g	< <i>language</i> > Language used to open a module.
-l	< <i>log file</i> > File to use as a log, append if it already exists.
-i	< <i>directory</i> > Extract files to the source image under the directory.
-cab	< <i>filename</i> > Extract the MSM cabinet to file.
-lfn	Use Long File Names during the extraction.
-configure	< <i>filename</i> > Configure the module using data from a file.

## See Also

[Windows Installer Development Tools  
Released Versions, Tools, and Redistributables](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patchwiz.dll

To generate a patch package, it is recommended that you use a patch creation tool such as [Msimsp.exe](#) and Patchwiz.dll. Patchwiz.dll version 4.0 is compatible with packages and patches that were authored using earlier versions of the Patchwiz.dll. The Patchwiz.dll tool is only available in the [Windows SDK Components for Windows Installer Developers](#).

Patchwiz.dll version 4.0 has one new function, [UiCreatePatchPackageEx \(Patchwiz.dll\)](#), that extends the functionality of [UiCreatePatchPackage \(Patchwiz.dll\)](#). These functions take a patch creation properties file (.pcp file) and generate an installer [Patch Package](#).

The .pcp file is a binary database file with the same format as a Windows Installer database (.msi file), but with a different database schema. Therefore a .pcp file can be authored by using the same tools used for an installer database.

You can create a .pcp file by using a table editor such as [Orca.exe](#) to enter information into the blank .pcp database provided with the Windows Installer SDK, Template.pcp. For more information, see [A Small Update Patching Example](#).

The following database tables are required in every .pcp file:

- [Properties Table \(Patchwiz.dll\)](#)
- [ImageFamilies Table \(Patchwiz.dll\)](#)
- [UpgradedImages Table \(Patchwiz.dll\)](#)
- [TargetImages Table \(Patchwiz.dll\)](#)

The following database tables are optional:

- [UpgradedFiles\\_OptionalData Table \(Patchwiz.dll\)](#)
- [FamilyFileRanges Table \(Patchwiz.dll\)](#)
- [TargetFiles\\_OptionalData Table \(Patchwiz.dll\)](#)
- [ExternalFiles Table \(Patchwiz.dll\)](#)
- [UpgradedFilesToIgnore Table \(Patchwiz.dll\)](#)

The following table is required in .pcp files that have a MinimumRequiredMsiVersion equal to 300 in the Properties table.

- [PatchMetadata Table \(Patchwiz.dll\)](#)

**Note** The table is optional if MinimumRequiredMsiVersion is not equal to 300.

The version of Patchwiz.dll released with Windows Installer 3.0 can automatically generate patch sequencing information and add it to the [MsiPatchSequence Table](#) of a new patch. The [PatchSequence Table](#) can be used to manually add patch sequencing information the MsiPatchSequence Table. For more information, see [Generating Patch Sequence Information](#).

Beginning with Patchwiz.dll version 2.0, you can increase the speed of subsequent patch creation by using [Patch Information Caching \(Patchwiz.dll\)](#).

Using public symbols for your target and upgrade image binaries can reduce binary patch sizes by approximately one half. For more information, see [Using Symbols to Reduce Binary Patch Size](#).

You can specify that certain regions of the target file be preserved from being overwritten during patching and that the information in those regions be retained. For more information, see [Patching Selected Regions of a File](#).

## See Also

[Released Versions, Tools, and Redistributables](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patch Information Caching (Patchwiz.dll)

Generating a new patch may require significant time. After you have generated a patch using [Patchwiz.dll](#), you may need to change the update image again and generate another patch. Patch information caching can reduce the time required to generate subsequent patches by reusing existing patches. For example, the time required to create service packs can be reduced by using the binary patches generated from previous patches. Patchwiz.dll can use PATCH\_CACHE\_DIR to find an existing binary patch and add it to the service pack's cabinet without having to re-create the binary patch.

Patch information caching requires the use of Patchwiz.dll. To activate patch caching, set the PATCH\_CACHE\_ENABLED and PATCH\_CACHE\_DIR properties in the [Properties Table \(Patchwiz.dll\)](#) of the patch creation properties file (.pcp file). Patchwiz stores all patch creation information in the folder identified by the PATCH\_CACHE\_DIR property and creates this folder if necessary. The next time you attempt to create a patch, Patchwiz checks this folder to see whether the files to be compared match the files in the cache. If the files match, Patchwiz uses the cached information rather than regenerating the binary patch for the file. If the files do not match, or if the information is missing from the cache, Patchwiz generates the patch for the file.

To use patch information caching, the folder specified by PATCH\_CACHE\_DIR must be preserved after creating a .msp file. If the folder is deleted, PatchWiz has to re-generate binary patches for subsequent .msp files. For more information about preserving information in selected regions of a target file see [Patching Selected Regions of a File](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Using Symbols to Reduce Binary Patch Size

Using public symbols for your target and upgrade image binaries can reduce binary patch sizes by approximately one half. The actual reduction depends upon the symbols used. Note that using symbols can result in slower patch creation times because it takes longer to process the symbol files.

To reduce the size of a binary patch using symbols, you must provide symbols for both the target and upgrade image binaries. Specify the symbols in the SymbolPaths column of the [TargetImages](#) table and the SymbolPaths column of the [UpgradedImages](#) table. You must use Visual C++ to generate symbols in the program database (PDB) file format. Newer versions of Visual C++ provide all of the necessary information in the PDB file. Older versions of Visual C++ also generate the debug (DBG) file format. In this case, the SymbolsPaths value should specify the location of both the PDB and DBG files.

For example, the TargetImage for a patch might be the installation package that shipped with MicrosoftWindows 2000 and that installs the 1.1.1029.0 version of MSI.DLL. The UpgradedImage might be the updated installation package that shipped with Windows 2000 SP1 and that installs the 1.11.1314.0 version of MSI.DLL. Two Patch Creation Properties (PCP) files would then have to be created, one with the SymbolPaths column of both the TargetImages and UpgradedImages tables left NULL (blank) and the other with the SymbolPaths column of both the TargetImages and UpgradedImages tables populated with the location of the symbols for the binaries. In this case, the size of the patch generated without using symbols can be approximately three times the size of the patch generated using symbols.

The Mpatch.exe utility can be used to test the generation of binary patches for a single file and to check whether or not the symbols are valid. The Mpatch.exe utility is included in the [Windows SDK Components for Windows Installer Developers](#). The output of Mpatch.exe will indicate if the symbols do not match.

For example, enter the following command line to check whether or not

the symbols are valid.

**`mpatch.exe -NEWSYMPATH:d:\update -OLDSYMPATH:d:\target  
d:\target\example.dll d:\update\example.dll example.pat`**

If the symbols are not in the correct location, the output of Mpatch.exe may include the following warning.

```
WARNING: no debug symbols for d:\update\example.dll
```

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Patching Selected Regions of a File

When patching files having variable content, it may be necessary to retain selected regions of the target file to prevent the loss of critical information. For example, some applications stamp user information into the executable file. Because the contents of the target file might depend upon the computer on which the application is installed, it becomes difficult to determine whether a particular file is a valid target for the patch. User information written in the target file can also be overwritten by a patch.

When you generate a patch creation properties (PCP) file with [Msimsp.exe](#) and [PATCHWIZ.DLL](#), you can specify that the information in certain regions of the target file be ignored during patching. You can also specify that information in other regions of the target file be retained and copied to an offset location in the updated file. You specify which regions of the target file to ignore and which regions to retain when authoring the [TargetFiles OptionalData](#), [ExternalFiles](#), and [FamilyFileRanges](#) tables.

Use the RetainOffsets column of the TargetFiles OptionalData table and the RetainOffsets and RetainLengths columns of the FamilyFileRanges table to copy a range of information from the target file to an offset range in the updated file. The information in this range is retained. Specify the length of the range using the RetainLengths columns of the FamilyFileRanges table. The length of the retained range is the same in the target and updated files. Specify the offset of the range in the target file using the RetainOffsets column of the TargetFiles OptionalData table. Specify the offset of the range in the updated file using the RetainOffsets column of the FamilyFileRanges table. The range of the target file retained is therefore the value of RetainOffsets in the TargetFiles OptionalData table plus RetainLengths. This information gets copied to the update file in the range given by the value of RetainOffsets in the FamilyFileRanges tables plus RetainLengths. You can specify multiple ranges but the order of the lengths must match the order of the offsets.

Use the TargetFiles OptionalData table to ignore a range of the target file. Information in this range of the target file can still be overwritten by the patch. Specify the offset of the range in the IgnoreOffsets column and its length in the IgnoreLengths column. You can specify multiple ranges but the order of the lengths must match the order of the offsets.

The values in the lengths and offsets columns can be decimal or hexadecimal. PATCHWIZ.DLL treats the value as hexadecimal if it is prefixed by "0x". The columns are string columns, so PATCHWIZ.DLL converts the values to ULONGs. The length column specifies the length in bytes.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Generating Patch Sequence Information (PATCHWIZ.DLL)

The version of [PATCHWIZ.DLL](#) released with Windows Installer 3.0 can automatically generate patch sequencing information and add to the [MsiPatchSequence Table](#) a new patch.

Set the `SEQUENCE_DATA_GENERATION_DISABLED` property to 1 (one) in the [Properties Table](#) of the .pcp file to prevent the automatic generation of patch sequencing information. If this property is absent, the information is automatically generated and added.

When the [PATCHWIZ.DLL](#) released with Windows Installer 3.0 is used to automatically generate the patch sequencing information, the following occurs:

- A new row is added to the MsiPatchSequence Table for each product code of a target image that is listed in the [TargetImages Table](#).
- The values added to the PatchFamily column in the new rows correspond to the target product codes of the target images that are listed in the TargetImages Table.
- The values added to the Sequence columns in the new rows are generated using the highest product version targeted by the patch and the UTC time when the patch is generated. The sequence number is <Product Minor Version>.<Build Major Number>.<Time Stamp 1>.<Time Stamp 2>.
  - The first field is the product version of the highest version of the product that is targeted by the patch.
  - The second field is the build major number of the highest version of the product that is targeted by the patch.

The two time stamp fields account for the 32 bit time stamp that is needed to count the seconds in Coordinated Universal Time (UTC).

**Note** Product versions have the following format: <Product Major

Version>.<Product Minor Version>.<Build Major Number>.<Build Minor Number> and a product with a version number 2.1.0.0 is a higher version than a product with version number 1.2.0.0

- The msidbPatchSequenceSupersedeEarlier attribute is entered into the Attribute column of new rows that are generated for service packs (SP) or minor upgrade patches. The msidbPatchSequenceSupersedeEarlier attribute is not added to a QFE or small update patch.

**Note** A service pack (SP) should contain the fixes of all the QFEs that were released prior to it. However, if a patch author sets the SEQUENCE\_DATA\_SUPERSEDEENCE property to 0 (zero) or 1 (one) in the .pcp file, the Attributes column of all rows in the MsiPatchSequence table is set to the value that is specified for SEQUENCE\_DATA\_SUPERSEDEENCE. Patch authors who require more control must author the Attributes column manually.

If you include a [PatchSequence Table](#) in the .pcp file, the SEQUENCE\_DATA\_GENERATION\_DISABLED property is ignored and the information provided in the PatchSequence Table can be added to the MsiPatchSequence Table of the patch.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UiCreatePatchPackageEx (Patchwiz.dll)

The UiCreatePatchPackageEx function takes a package creation file (.pcp file) and generates a Windows Installer patch package (.msp package). Calling [Msimsp.exe](#) is the recommended method for using [Patchwiz.dll](#).

The UiCreatePatchPackageEx function is available beginning with Patchwiz.dll version 4.0 and extends the functionality of the [UiCreatePatchPackage](#) function.

```
UINT UiCreatePatchPackageEx(  
    LPCTSTR szPcpPath,  
    LPCTSTR szPatchPath,  
    LPCTSTR szLogPath,  
    HWND hwndStatus,  
    LPCTSTR szTempFolder,  
    BOOL fRemoveTempFolderContents,  
    DWORD dwFlags,  
    DWORD dwReserved  
);
```

## Parameters

### *szPcpPath*

Full path to the patch creation properties file (.pcp file) for this patch.

### *szPatchPath*

Full path to the Windows Installer patch package (.msp file) that is to be created. This parameter may be null or an empty string but may not be omitted. If it is null or an empty string, the function uses the value of PatchOutputPath in the [Properties Table \(Patchwiz.dll\)](#).

### *szLogPath*

Full path to a text log file that will be appended. This parameter may be null or an empty string but may not be omitted.

### *hwndStatus*

Handle to a window that displays the status text. This parameter may be null or an empty string but may not be omitted.

#### *szTempFolder*

Location for temporary files. This parameter may be null or an empty string but may not be omitted. The user must have sufficient privileges to read and write to this folder. The default location is %TMP%\~pcw\_tmp.tmp\.

#### *fRemoveTempFolderContents*

If TRUE, remove the temporary folder and all of its contents if present. If FALSE, and folder is present, the function fails.

#### *dwFlags*

This parameter can be set to one or a combination of the following values to specify logging or user interface options.

<b>Flag</b>	<b>Value</b>	<b>Meaning</b>
LOGNONE	0x00000000	Write no messages to the log.
LOGINFO	0x00000001	Write informational messages to the log.
LOGWARN	0x00000002	Write warnings to the log.
LOGERR	0x00000004	Write error messages to the log.
LOGPERFMESSAGES	0x00000008	Write performance messages to the log.
UINONE	0x0000000f	Do not display the user interface.
UIALL	0x00000010	Display the user interface.

#### *dwReserved*

Reserved. This parameter must be set to zero.

## **Return Values**

See the table in [Return Values for UiCreatePatchPackage](#).

## Remarks

For an example of authoring a .pcp file and using UiCreatePatchPackage to generate a Windows Installer patch package, see the section [A Small Update Patching Example](#).

Creating a patch requires an uncompressed setup image, such as an administrative image or an uncompressed setup image from a CD-ROM. UiCreatePatchPackage does not generate binary patches for files in cabinets.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UiCreatePatchPackage (Patchwiz.dll)

The `UiCreatePatchPackage` function takes a package creation file (.pcp file) and generates a Windows Installer patch package (.msp package). Calling [Msimsp.exe](#) is the recommended method for using [Patchwiz.dll](#). The [UiCreatePatchPackageEx](#) function is available in version 4.0 of `Patchwiz.dll` and extends the functionality of the `UiCreatePatchPackage` function.

```
UINT UiCreatePatchPackage(  
    LPCTSTR szPcpPath,  
    LPCTSTR szPatchPath,  
    LPCTSTR szLogPath,  
    HWND hwndStatus,  
    LPCTSTR szTempFolder,  
    Bool fRemoveTempFolderContents  
);
```

## Parameters

### *szPcpPath*

Full path to the patch creation properties file (.pcp file) for this patch.

### *szPatchPath*

Full path to the Windows Installer patch package (.msp file) that is to be created. This parameter may be null or an empty string but may not be omitted. If it is null or an empty string, the function uses the value of `PatchOutputPath` in the [Properties Table \(Patchwiz.dll\)](#).

### *szLogPath*

Full path to a text log file that will be appended. This parameter may be null or an empty string but may not be omitted.

### *hwndStatus*

Handle to a window that displays the status text. This parameter may be null or an empty string but may not be omitted.

### *szTempFolder*

Location for temporary files. This parameter may be null or an empty string but may not be omitted. The default location is

%TMP%\~pcw\_tmp.tmp\.

### *fRemoveTempFolderContents*

If TRUE, remove the temporary folder and all of its contents if present. If FALSE, and folder is present, the function fails.

## **Return Values**

See the table in [Return Values for UiCreatePatchPackage](#).

## **Remarks**

For an example of authoring a .pcp file and using UiCreatePatchPackage to generate a Windows Installer patch package, see the section [A Small Update Patching Example](#).

Creating a patch requires an uncompressed setup image, such as an administrative image or an uncompressed setup image from a CD-ROM. UiCreatePatchPackage does not generate binary patches for files in cabinets.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Return Values for UiCreatePatchPackage

UiCreatePatchPackageEx (PATCHWIZ.DLL) may return the following values. The return value is ERROR\_SUCCESS plus the values in this table.

Error Code	Description
ERROR_PCW_BASE	Base
ERROR_PCW_PCP_DOESNT_EXIST	The patch package does not exist
ERROR_PCW_PCP_BAD_FORMAT	The patch package is invalid
ERROR_PCW_CANT_CREATE_TEMP_FOLDER	Cannot create temporary folder
ERROR_PCW_MISSING_PATCH_PATH	The patch path is missing. <a href="#">Propagate Patch</a>
ERROR_PCW_CANT_OVERWRITE_PATCH	Unable to overwrite patch
ERROR_PCW_CANT_CREATE_PATCH_FILE	Unable to create patch file
ERROR_PCW_MISSING_PATCH_GUID	The patch GUID is missing. <a href="#">Table</a>
ERROR_PCW_BAD_PATCH_GUID	The patch GUID is bad. <a href="#">Table</a>
ERROR_PCW_BAD_GUIDS_TO_REPLACE	The patch GUIDs to replace are bad. <a href="#">Table</a>
ERROR_PCW_BAD_TARGET_PRODUCT_CODE_LIST	The patch target product code list is bad. <a href="#">Table</a>
ERROR_PCW_NO_UPGRADED_IMAGES_TO_PATCH	There are no upgraded images to patch

	Not i
ERROR_PCW_BAD_API_PATCHING_SYMBOL_FLAGS	The . the P
ERROR_PCW_OODS_COPYING_MSI	Out o
ERROR_PCW_UPGRADED_IMAGE_NAME_TOO_LONG	The j Upgr
ERROR_PCW_BAD_UPGRADED_IMAGE_NAME	The j Upgr
ERROR_PCW_DUP_UPGRADED_IMAGE_NAME	The j Upgr
ERROR_PCW_UPGRADED_IMAGE_PATH_TOO_LONG	The j Upgr
ERROR_PCW_UPGRADED_IMAGE_PATH_EMPTY	The j Upgr
ERROR_PCW_UPGRADED_IMAGE_PATH_NOT_EXIST	The j Upgr
ERROR_PCW_UPGRADED_IMAGE_PATH_NOT_MSI	The j Upgr file.
ERROR_PCW_UPGRADED_IMAGE_COMPRESSED	A co refer Upgr imag
ERROR_PCW_TARGET_IMAGE_NAME_TOO_LONG	The j Targ
ERROR_PCW_BAD_TARGET_IMAGE_NAME	The j Targ
ERROR_PCW_DUP_TARGET_IMAGE_NAME	The j Targ

ERROR_PCW_TARGET_IMAGE_PATH_TOO_LONG	The j Targ
ERROR_PCW_TARGET_IMAGE_PATH_EMPTY	The j Targ
ERROR_PCW_TARGET_IMAGE_PATH_NOT_EXIST	The j Targ
ERROR_PCW_TARGET_IMAGE_PATH_NOT_MSI	The j Targ
ERROR_PCW_TARGET_IMAGE_COMPRESSED	A co by th Tabl unco
ERROR_PCW_TARGET_BAD_PROD_VALIDATE	The field inval
ERROR_PCW_TARGET_BAD_PROD_CODE_VAL	The j the T Tabl
ERROR_PCW_UPGRADED_MISSING_SRC_FILES	Ther upgr Upgr Tabl
ERROR_PCW_TARGET_MISSING_SRC_FILES	Ther targe field
ERROR_PCW_IMAGE_FAMILY_NAME_TOO_LONG	The j Imag (PAI
ERROR_PCW_BAD_IMAGE_FAMILY_NAME	The j Imag (PAI
ERROR_PCW_DUP_IMAGE_FAMILY_NAME	The j Imag

	(PAT
ERROR_PCW_BAD_IMAGE_FAMILY_SRC_PROP	The j Imag (PAT
ERROR_PCW_UFILEDATA_LONG_FILE_TABLE_KEY	The j Upgr too l
ERROR_PCW_UFILEDATA_BLANK_FILE_TABLE_KEY	The j Upgr blank
ERROR_PCW_UFILEDATA_MISSING_FILE_TABLE_KEY	The j Upgr missi
ERROR_PCW_EXTFILE_LONG_FILE_TABLE_KEY	The j Exte
ERROR_PCW_EXTFILE_BLANK_FILE_TABLE_KEY	The j Exte
ERROR_PCW_EXTFILE_BAD_FAMILY_FIELD	The j the E
ERROR_PCW_EXTFILE_LONG_PATH_TO_FILE	The j Exte
ERROR_PCW_EXTFILE_BLANK_PATH_TO_FILE	The j Exte
ERROR_PCW_EXTFILE_MISSING_FILE	The j Exte
ERROR_PCW_FILERANGE_LONG_FILE_TABLE_KEY	The j Fami
ERROR_PCW_FILERANGE_BLANK_FILE_TABLE_KEY	The j Fami
ERROR_PCW_FILERANGE_MISSING_FILE_TABLE_KEY	The j Fami
ERROR_PCW_FILERANGE_LONG_PATH_TO_FILE	The j

	Fami
ERROR_PCW_FILERANGE_MISSING_FILE	The : Fami
ERROR_PCW_FILERANGE_INVALID_OFFSET	The : field inval
ERROR_PCW_FILERANGE_INVALID_SIZE	The : field inval
ERROR_PCW_FILERANGE_INVALID_RETAIN	The : Fami
ERROR_PCW_BAD_MEDIA_SRC_PROP_NAME	The : Imag (PAT
ERROR_PCW_BAD_MEDIA_DISK_ID	The : Imag (PAT
ERROR_PCW_BAD_FILE_SEQUENCE_START	The : Imag (PAT
ERROR_PCW_CANT_COPY_FILE_TO_TEMP_FOLDER	Unat folde
ERROR_PCW_CANT_CREATE_ONE_PATCH_FILE	Unat
ERROR_PCW_BAD_IMAGE_FAMILY_DISKID	The : Imag (PAT
ERROR_PCW_BAD_IMAGE_FAMILY_FILESEQSTART	The : Imag (PAT
ERROR_PCW_BAD_UPGRADED_IMAGE_FAMILY	The : Tabl

ERROR_PCW_BAD_TARGET_IMAGE_UPGRADED	The <a href="#">Table</a>
ERROR_PCW_DUP_TARGET_IMAGE_PACKCODE	The dupli
ERROR_PCW_UFILEDATA_BAD_UPGRADED_FIELD	The <a href="#">Upgr</a> ( <a href="#">Patc</a>
ERROR_PCW_MISMATCHED_PRODUCT_CODES	The j
ERROR_PCW_MISMATCHED_PRODUCT_VERSIONS	The j
ERROR_PCW_CANNOT_WRITE_DDF	Canr
ERROR_PCW_CANNOT_RUN_MAKECAB	Canr
ERROR_PCW_CANNOT_CREATE_STORAGE	Canr
ERROR_PCW_CANNOT_CREATE_STREAM	Canr
ERROR_PCW_CANNOT_WRITE_STREAM	Canr
ERROR_PCW_CANNOT_READ_CABINET	Canr
ERROR_PCW_WRITE_SUMMARY_PROPERTIES	Error
ERROR_PCW_TFILEDATA_LONG_FILE_TABLE_KEY	The : <a href="#">Targ</a> long.
ERROR_PCW_TFILEDATA_BLANK_FILE_TABLE_KEY	The : <a href="#">Targ</a>
ERROR_PCW_TFILEDATA_MISSING_FILE_TABLE_KEY	The : <a href="#">Targ</a>

	missi
ERROR_PCW_TFILEDATA_BAD_TARGET_FIELD	The t Targ inval
ERROR_PCW_UPGRADED_IMAGE_PATCH_PATH_TOO_LONG	The j the U
ERROR_PCW_UPGRADED_IMAGE_PATCH_PATH_NOT_EXIST	The j the U none
ERROR_PCW_UPGRADED_IMAGE_PATCH_PATH_NOT_MSI	The j the U with
ERROR_PCW_DUP_UPGRADED_IMAGE_PACKCODE	The i dupli
ERROR_PCW_UFILEIGNORE_BAD_UPGRADED_FIELD	The i Upgr (Pac
ERROR_PCW_UFILEIGNORE_LONG_FILE_TABLE_KEY	The i Upgr (Pac
ERROR_PCW_UFILEIGNORE_BLANK_FILE_TABLE_KEY	The i Upgr (Pac
ERROR_PCW_UFILEIGNORE_BAD_FILE_TABLE_KEY	The i Upgr (Pac
ERROR_PCW_FAMILY_RANGE_NAME_TOO_LONG	The i Fami is too
ERROR_PCW_BAD_FAMILY_RANGE_NAME	The i Fami is inv
ERROR_PCW_FAMILY_RANGE_LONG_FILE_TABLE_KEY	The i

	Fami is to
ERROR_PCW_FAMILY_RANGE_BLANK_FILE_TABLE_KEY	The : Fami is bla
ERROR_PCW_FAMILY_RANGE_LONG_RETAIN_OFFSETS	The ) of th (Patc
ERROR_PCW_FAMILY_RANGE_BLANK_RETAIN_OFFSETS	The ) of th (Patc
ERROR_PCW_FAMILY_RANGE_BAD_RETAIN_OFFSETS	The ) of th (Patc
ERROR_PCW_FAMILY_RANGE_LONG_RETAIN_LENGTHS	The ) field (Patc
ERROR_PCW_FAMILY_RANGE_BLANK_RETAIN_LENGTHS	The ) field (Patc
ERROR_PCW_FAMILY_RANGE_BAD_RETAIN_LENGTHS	The ) field (Patc
ERROR_PCW_FAMILY_RANGE_COUNT_MISMATCH	The ) Ret Fami are n
ERROR_PCW_EXTFILE_LONG_IGNORE_OFFSETS	The ) field (Patc
ERROR_PCW_EXTFILE_BAD_IGNORE_OFFSETS	The ) field (Patc

ERROR_PCW_EXTFILE_LONG_IGNORE_LENGTHS	The field (Patc
ERROR_PCW_EXTFILE_BAD_IGNORE_LENGTHS	The field (Patc
ERROR_PCW_EXTFILE_IGNORE_COUNT_MISMATCH	The Igro Tabl
ERROR_PCW_EXTFILE_LONG_RETAIN_OFFSETS	The field (Patc
ERROR_PCW_EXTFILE_BAD_RETAIN_OFFSETS	The field (Patc
ERROR_PCW_EXTFILE_RETAIN_COUNT_MISMATCH	The Reta Tabl
ERROR_PCW_TFILEDATA_LONG_IGNORE_OFFSETS	The field Tabl
ERROR_PCW_TFILEDATA_BAD_IGNORE_OFFSETS	The field Tabl
ERROR_PCW_TFILEDATA_LONG_IGNORE_LENGTHS	The field Tabl
ERROR_PCW_TFILEDATA_BAD_IGNORE_LENGTHS	The field Tabl
ERROR_PCW_TFILEDATA_IGNORE_COUNT_MISMATCH	The lengt Tabl

ERROR_PCW_TFILEDATA_LONG_RETAIN_OFFSETS	The field (Patc
ERROR_PCW_TFILEDATA_BAD_RETAIN_OFFSETS	The field Tabl
ERROR_PCW_TFILEDATA_RETAIN_COUNT_MISMATCH	The the T (Patc
ERROR_PCW_CANT_GENERATE_TRANSFORM	Unat
ERROR_PCW_CANT_CREATE_SUMMARY_INFO	Unat infor
ERROR_PCW_CANT_GENERATE_TRANSFORM_POUND	Unat
ERROR_PCW_CANT_CREATE_SUMMARY_INFO_POUND	Unat infor
ERROR_PCW_BAD_UPGRADED_IMAGE_PRODUCT_CODE	The imag
ERROR_PCW_BAD_UPGRADED_IMAGE_PRODUCT_VERSION	The imag
ERROR_PCW_BAD_UPGRADED_IMAGE_UPGRADE_CODE	The imag
ERROR_PCW_BAD_TARGET_IMAGE_PRODUCT_CODE	The inval
ERROR_PCW_BAD_TARGET_IMAGE_PRODUCT_VERSION	The inval
ERROR_PCW_BAD_TARGET_IMAGE_UPGRADE_CODE	The inval
ERROR_PCW_MATCHED_PRODUCT_VERSIONS	The and u prod

	least Prod upgr.
ERROR_PCW_MATCHED_PRODUCT_VERSIONS	The j and t prod least Prod upgr.
ERROR_PCW_INVALID_PARAMETER	An in speci Avai versi
ERROR_PCW_CREATEFILE_LOG_FAILED	UiCr the l Patcl
ERROR_PCW_INVALID_LOG_LEVEL	An in by U begin
ERROR_PCW_INVALID_UI_LEVEL	An in speci Avai versi
ERROR_PCW_ERROR_WRITING_TO_LOG	UiCr write with
ERROR_PCW_OUT_OF_MEMORY	UiCr insuf oper: Patcl
ERROR_PCW_UNKNOWN_ERROR	UiCr beca Avai versi

ERROR_PCW_UNKNOWN_INFO	UiCr unre Avai versi
ERROR_PCW_UNKNOWN_WARN	UiCr unre begin
ERROR_PCW_OPEN_VIEW	UiCr open Patcl
ERROR_PCW_EXECUTE_VIEW	UiCr run a Patcl
ERROR_PCW_VIEW_FETCH	UiCr fetch Patcl
ERROR_PCW_FAILED_EXPAND_PATH	UiCr resol begin
ERROR_PCW_INTERNAL_ERROR	UiCr an in with
ERROR_PCW_INVALID_PCP_PROPERTY	The ) file) Avai versi
ERROR_PCW_INVALID_PCP_TARGETIMAGES	The ' inval Patcl
ERROR_PCW_LAX_VALIDATION_FLAGS	The ' Prod Targ com begin

ERROR_PCW_FAILED_CREATE_TRANSFORM	UiCr creat begin
ERROR_PCW_CANT_DELETE_TEMP_FOLDER	UiCr remc begin
ERROR_PCW_MISSING_DIRECTORY_TABLE	The begin
ERROR_PCW_INVALID_SUPERSEDENCE_VALUE	The SEQ prop (Patc begin
ERROR_PCW_INVALID_PATCH_TYPE_SEQUENCING	The MsiE infor Avai versi
ERROR_PCW_CANT_READ_FILE	UiCr com could with
ERROR_PCW_TARGET_WRONG_PRODUCT_VERSION_COMP	The Allo value Avai versi
ERROR_PCW_INVALID_PCP_UPGRADEDFILESTOIGNORE	The (Patc inval Patcl
ERROR_PCW_INVALID_PCP_UPGRADEDIMAGES	The table begin

ERROR_PCW_INVALID_PCP_EXTERNALFILES	The   the s begin
ERROR_PCW_INVALID_PCP_IMAGEFAMILIES	The   or th begin
ERROR_PCW_INVALID_PCP_PATCHSEQUENCE	The   or th begin
ERROR_PCW_INVALID_PCP_TARGETFILES_OPTIONALDATA	The ' (Patc inval Patch
ERROR_PCW_INVALID_PCP_UPGRADEDFILES_OPTIONALDATA	The   (Patc inval Patch
ERROR_PCW_MISSING_PATCHMETADATA	The   or th begin
ERROR_PCW_IMAGE_PATH_NOT_EXIST	Unat beca does Patch
ERROR_PCW_INVALID_RANGE_ELEMENT	Unat beca inval Patch
ERROR_PCW_INVALID_MAJOR_VERSION	Unat beca chan prod with

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Properties Table (Patchwiz.dll)

The Properties Table contains global settings for the patch package. The Properties Table is required in the patch creation database (.pcp file), and is used by the [UiCreatePatchPackageEx](#) function.

The Properties Table has the following columns.

Column	Type	Key	Nullable
Name	text	Y	N
Value	text		Y

## Columns

### Name

The name of a patch creation property.

### Value

The value of the patch creation property.

## Remarks

The following table identifies the patch creation property strings that can be entered into the Properties Table of the .pcp file.

Property	Description
AllowLaxValidationFlags	If this property is set to 1, an entry is written in the log and no error is returned if the ProductValidateFlags field in the <a href="#">TargetImages table</a> can be validated. This property should only be set when the patch author has changed the value in the

	<p>ProductValidateFlags field. The default value for this property is 0. Available beginning with Patchwiz.dll version 4.0.</p>
AllowProductCodeMismatches	<p>Set to '1' for True if the <b>ProductCode</b> Property may differ between the upgrade images listed in the <b>UpgradedImages Table</b> and the target images listed in the <b>TargetImages Table</b>. Set to '0' or blank to mean False if the product codes must be the same.</p>
AllowProductVersionMajorMismatches	<p>Set to '1' to mean True if the first field (the major version field) of the <b>ProductVersion</b> Property may differ between upgraded images and target images. Set to '0' or blank to mean False if the major versions must match.</p>
ApiPatchingSymbolFlags	<p>An 8-digit hex integer representing the combination of patch symbol usage flags used when creating a binary patch. Default is 0x000000. See Patchapi.h for the complete list of possible <b>PATCH_SYMBOL_*</b> flags. See the glossary for more information on symbol files.</p>
DontRemoveTempFolderWhenFinished	<p>Set to '1' to mean True if the temporary folder containing the transforms, the byte-level patches, and the entire new</p>

	<p>files are not being removed after creating the patch package. This is essentially the contents of the .msp file before being embedded into the patch package. This may be useful for debugging patches. Set to '0' or blank to mean False if the temporary folder is to be removed.</p>
IncludeWholeFilesOnly	<p>Set to '1' to mean True if files being changed are to be included in their entirety when creating the patch package instead of creating a binary file patch. The patch files will be bigger in size but the AI runs faster. Set to '0' or blank to mean False if creating a binary file patch.</p>
ListOfPatchGUIDsToReplace	<p>A list of PatchGUID identifiers with no delimiters. Any of these patches are found to be installed on the user's computer and registered with Windows Installer, they are unregistered from the appropriate product and the patch transforms are removed from the list of transforms associated with the product. Note that the removal of a patch does not affect any of the files, only the registration of the patch. Optional.</p>
ListOfTargetProductCodes	<p>A list of <a href="#">Product Codes</a> for products that may receive t</p>

	<p>patch. This is a semicolon delimited list of <a href="#">ProductCode</a> Property values. If the list begins with an asterisk, the list of product codes is generated from the .msi file of the targets listed in the <a href="#">TargetImages Table</a>. If any product codes follow a leading asterisk, they are appended to the list that replaces the asterisk. If the property is not set, a list of product codes is generated from the .msi files of the targets listed in the <a href="#">TargetImages Table</a>.</p>
MsiFileToUseToCreatePatchTables	<p>The full path to a template .msi file from which to export the <a href="#">Patch Table</a> and <a href="#">PatchPackage Table</a>. The Properties Table accepts environment variables for paths beginning with version 4.0 of Patchwiz.dll. Use the Windows format for the environment variable, such as <code>%ENV_VAR%</code>. Do not use <a href="#">Formatted</a> column data type.</p>
OptimizePatchSizeForLargeFiles	<p>This property is set when the value exists and is not "0". When this property is set, patches for files greater than approximately 4 MB in size may be made smaller.</p>
PatchGUID	<p>A <a href="#">GUID</a> identifier for this patch package (.msp file).</p>

	<p>Every patch package must have a unique PatchGUID value. Required.</p>
PatchOutputPath	<p>The full path, including file name, of the patch package file that is to be generated. <i>szPatchPath</i> is passed by the <code>UiCreatePatchPackageEx</code> function, the passed value is used. This property is required if <i>szPatchPath</i> is null or an empty string.</p> <p>The Properties Table accepts environment variables for paths beginning with version 4.0 of Patchwiz.dll. Use the Windows format for the environment variable, such as <code>%ENV_VAR%</code>. Do not use Formatted column data type</p>
PatchSourceList	<p>A source used to locate the .msp file for the patch in the event that the locally cached copy is unavailable. This value is added to the source list of the patch when it is applied to a product. Optional</p>
MinimumRequiredMsiVersion	<p>Set this property to force Patchwiz.dll to generate a patch that requires a particular version of Windows Installer. This property value helps determine what value to use for the <b>Word Count Summary</b> Property of the patch package. The value for this property is of the same</p>

form as the **Page Count Summary** Property of the installation package.

If a .pcp file has a MinimumRequiredMsiVersion equal to 200, Patchwiz.dll sets the **Word Count Summary** property of the patch package to 3. This prevents the patch from being applied by Windows Installer version earlier than version 2.0.

If a .pcp file has a MinimumRequiredMsiVersion equal to 300, Patchwiz.dll sets the **Word Count Summary** Property of the patch package to 4. This prevents the patch from being applied by Windows Installer versions earlier than version 3.0.

If a .pcp file has a MinimumRequiredMsiVersion equal to 310, Patchwiz.dll sets the **Word Count Summary** Property of the patch package to 5. This prevents the patch from being applied by Windows Installer earlier than version 3.1.

If a .pcp file has a MinimumRequiredMsiVersion equal to 400, Patchwiz.dll sets the **Word Count Summary** property of the patch package to 6. This prevents the patch from being applied by

	Windows Installer earlier than version 4.0.
PATCH_CACHE_ENABLED	<p>Set this property to 1 to cache the patch creation information in the folder specified by the PATCH_CACHE_DIR property.</p> <p>Patch caching increases the speed of patch creation when recreating a new patch after updating the update image. This property requires Patchwiz.dll in Windows Installer 2.0 or later. For information, see <a href="#">Patch Information Caching (Patchwiz.dll)</a>.</p>
PATCH_CACHE_DIR	<p>Set this property to the name of the folder that stores the cached patch information. Patchwiz.dll creates this folder if necessary. The folder should be on a drive with sufficient disk space. This property is only used if the PATCH_CACHE_ENABLED property is set to 1.</p> <p>The Properties Table accepts environment variables for paths beginning with version 4.0 of Patchwiz.dll. Use the Windows format for the environment variable, such as %ENV_VAR%. Do not use Formatted column data type</p>
SEQUENCE_DATA_GENERATION_DISABLED	Set this property to 1 (one)

	prevent the automatic generation of patch sequencing information. If this property is absent, sequencing information is automatically generated and added.
SEQUENCE_DATA_SUPERSEDENCE	Set this property to 0 (zero) or 1 (one) to write that value in the Attributes field of all records in the <a href="#">MsiPatchSequence table</a> .
TrustMsi	Set this property to 1 in the Properties Table to use the version information, size, and hash values provided in the .msi file. If this property is not set but the information in the .msi file is incorrect, the patch created may not function correctly. You should update the target and upgraded .msi files using <a href="#">MsiFiler.exe</a> . The default value for this property is 0. Available beginning with Patchwiz.dll version 4.0.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ImageFamilies Table (Patchwiz.dll)

An image family is a group of one or more upgraded images of a product that have been updated to the most recent version. Each upgraded image can belong to only one family. Upgraded images belonging to an image family share one or more files. Each image family has its own cabinet file in the .msp file containing the binary patches and new files needed to update the differences between target and upgraded files. The cabinet file does not replicate the binary patches and new files used by the shared files.

An ImageFamilies table containing at least one record is required in every patch creation database (.pcp file). This table is used by the [UiCreatePatchPackageEx](#) function.

The ImageFamilies table contains the patching information that is to be added to the [Media table](#). A patch adds one entry to the Media table. Each record in the ImageFamilies tables refers to a group of related product images that have been updated to the most recent version of the product.

The ImageFamilies table has the following columns. A null value can be used in the MediaSrcPropName, MediaDiskId, and FileSequenceStart columns if the patch is applied with Windows Installer and Patchwiz.dll version 2.0.

Column	Type	Key	Nullable
Family	text	Y	N
MediaSrcPropName	text		Y
MediaDiskId	integer		Y
FileSequenceStart	integer		Y
DiskPrompt	text		Y
VolumeLabel	text		Y

## Columns

## Family

The value entered in this field is an identifier for a group of related product images that have been updated to the most recent version of the product. Limited to a total of 8 alphanumeric characters or underscores. The installer embeds a cabinet stream in the Windows Installer patch file (.msp file) for each family in the table. The cabinet contains the binary patches and new files required to update a target image into an upgraded image of the product. The installer prefixes the family name with PCW\_CAB\_ to generate the cabinet's stream name it enters into the Cabinet field of the new [Media table](#) entry.

## MediaSrcPropName

The value entered into the Source field of the new Media table entry of the upgraded image. This field can be null only if you are using version 2.0 of Patchwiz.dll and if the MinimumRequiredMsiVersion in the [Properties table \(Patchwiz.dll\)](#) is set to 200.

## MediaDiskId

The installer enters this value into the DiskId field of the new Media table record. The DiskID value must be greater than any current DiskID in the target package. The limit for MediaDiskId is 32767. This field can be null only if you are using version 2.0 of Patchwiz.dll and if the MinimumRequiredMsiVersion in the [Properties table \(Patchwiz.dll\)](#) is set to 200.

## FileSequenceStart

This field is the sequence number for the starting file. This same file sequence number must not exist in two patches for the same product. To ensure this, the value in this field must be greater than all sequence numbers used in previous patches or in the original installation package. The greatest sequence number in a patch can be determined by adding the total number of entries in the patch cabinet file to the FileSequenceStart number for that patch. One way to determine this is to look at the .ddf file generated by [Patchwiz.dll](#) during the creation of the patch. The limit for FileSequenceStart is 32767. This field can be null only if you are using version 2.0 of Patchwiz.dll and if the MinimumRequiredMsiVersion in the [Properties table \(Patchwiz.dll\)](#) is set to 200.

## DiskPrompt

The installer enters this value into the DiskPrompt field of the new

Media table record.

#### VolumeLabel

The installer enters this value into the VolumeLabel field of the new Media record.

## Remarks

The patch adds the name of the cabinet in the .msp file to the Cabinet field of the new record added to the Media table. Because it is an embedded cabinet, the name is prefixed with a '#' character. The patch adds a property to the Source field of the new record in the Media table. No two patches may have the same source property.

The files that are shared within the image family must have the same file table key in each upgraded image of the family. Any file table keys shared among the upgraded images must represent the same file and must be identical in all the upgraded images. The file table key is the value entered in the File column of the [File table](#).

The limit for MediaDiskId and FileSequenceStart is 32767. To increase this limit export the ImageFamilies table to an .idt file with [Msidb.exe](#) and change the column type from i2 to i4, or from I2 to I4, and then import the .idt file back into the .pcp database. Transforms and patches cannot be created between two packages having different column types.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UpgradedImages Table (Patchwiz.dll)

The [UpgradedImages table](#) contains information about the upgraded images of the product. The upgraded image should be a fully uncompressed setup image of the latest version of the product, for example, an administrative image or an uncompressed setup image from a CD-ROM. A Windows Installer patch package updates a target image into an upgraded image. The UpgradedImages table is required in the patch creation database (.pcp file) and is used by [UiCreatePatchPackageEx](#).

An UpgradedImages table containing at least one record is required in every patch creation database (.pcp file). This table is used by [UiCreatePatchPackageEx](#).

The UpgradedImages table has the following columns.

Column	Type	Key	Nullable
Upgraded	text	Y	N
MsiPath	text		N
PatchMsiPath	text		Y
SymbolPaths	text		Y
Family	text		N

## Columns

### Upgraded

The Upgraded field is an arbitrary identifier to connect the target images with an upgraded image of the product.

### MsiPath

This field specifies the full path, including the file name, to the location of the .msi file for the upgraded image. This is the location of the source files for the upgraded image.

## PatchMsiPath

The optional patchMsiPath points to a modified copy of the upgraded installation database that contains additional authoring specific to the patch installation process. For example, additional dialogs or custom actions conditioned on the **PATCH** property.

## SymbolPaths

A semicolon delimited list of folders that are to be searched for symbol files that may be used to optimize the generation of the binary patch. Note that the subdirectories of folders specified in this field are not searched. An optimized binary patch may be smaller. Visual C++ must be installed on the computer generating the patch and used to create the symbol files. This field is optional, and the installer creates a binary patch even if no symbol files are specified or if the symbol files become unavailable to Patchwiz.dll.

## Family

Foreign key into the [ImageFamilies table](#). Each upgraded image must belong to only one family.

## Remarks

Although each upgraded image can be grouped into a separate image family, grouping upgraded images that share files together may make the .msp smaller.

This table accepts environment variables as paths beginning with version 4.0 of Patchwiz.dll.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# TargetImages Table (Patchwiz.dll)

The TargetImages table contains information about the target images of the product. A Windows Installer patch package updates a target image into an upgraded image.

A TargetImages table containing at least one record is required in every patch creation database (.pcp file). This table is used by the [UiCreatePatchPackage](#) function.

The TargetImages table has the following columns.

Column	Type	Key	Nullable
Target	text	Y	N
MsiPath	text		N
SymbolPaths	text		Y
Upgraded	text		N
Order	integer		N
ProductValidateFlags	text		Y
IgnoreMissingSrcFiles	integer		N

## Columns

### Target

Identifier for a target image. The patch package updates the target image specified in this column to the upgraded image specified in the Upgraded column. There are one or more target images for each upgraded image. The target image must be a fully uncompressed setup image of the product, such as an administrative image or an uncompressed setup image on a CD-ROM. Note that the [UiCreatePatchPackageEx](#) function does not generate binary patches for files in cabinets. The value in this field is used with the value in the Upgraded field to generate the names of the transforms that the installer adds to the patch package.

## MsiPath

This field specifies the full path, including the file name, to the location of the .msi file for the target image. This is the location of the source files for the target image.

## SymbolPaths

A semicolon delimited list of folders that are to be searched for symbol files that may be used to optimize the generation of the binary patch. Note that the subdirectories of folders specified in this field are not searched. An optimized binary patch may be smaller. Microsoft Visual C++ must be installed on the computer generating the patch and used to create the symbol files. This field is optional, and the installer creates a binary patch even if no symbol files are specified or if the symbol files become unavailable to Patchwiz.dll.

## Upgraded

Foreign key to the Upgraded column of the [UpgradedImages table](#). The UiCreatePatchPackageEx function ignores any upgraded image that is not referenced by at least one record of the TargetImages table.

## Order

Relative order of the target image. Because multiple targets can be patched to an upgraded image, the Order field provides a means to sequence the transforms in the patch transforms list. Commonly, the order is from oldest to newest image.

## ProductValidateFlags

The ProductValidateFlags field is used to specify product checking to avoid applying irrelevant transforms. The value entered in this field must be an 8-digit hex integer and one of the valid values for the *iValidation* parameter of the [MsiCreateTransformSummaryInfo](#) function. The default value is 0x00000922 which equals  
MSITRANSFORM\_VALIDATE\_UPDATEVERSION +  
MSITRANSFORM\_VALIDATE\_NEWEQUALBASEVERSION +  
MSITRANSFORM\_VALIDATE\_UPGRADECODE +  
MSITRANSFORM\_VALIDATE\_PRODUCT.

## IgnoreMissingSrcFiles

If this field is set to a nonzero value, files missing from the target image are ignored by the installer and left unchanged during

patching. This enables patches to be made without requiring the entire image; only the changed files of the product and the .msi file are required. This may reduce the time required to generate the patch.

**Note** Do not use the IgnoreMissingSrcFiles value with TrustMsi set to 1 in the [Properties Table](#).

## Remarks

This table accepts environment variables as paths beginning with version 4.0 of Patchwiz.dll.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UpgradedFiles\_OptionalData Table (Patchwiz.dll)

The UpgradedFile\_OptionalData table contains information about specific files in an upgraded image. This table is optional in the patch creation database (.pcp file) and is used by the [UiCreatePatchPackageEx](#) function.

The UpgradedFile\_OptionalData table has the following columns.

Column	Type	Key	Nullable
Upgraded	text	Y	N
FTK	text	Y	N
SymbolPaths	text		Y
AllowIgnoreOnPatchError	integer		Y
IncludeWholeFile	integer		Y

## Columns

### Upgraded

Foreign key to the Upgraded column of the [UpgradedImages Table \(Patchwiz.dll\)](#).

### FTK

File table key. Foreign key into [File table](#) of the .msi file of the upgraded image. If two or more upgraded images within a family have the same FTK value, the value must refer to the same file. Files shared by multiple upgrade images should have the same FTK to minimize cabinet file size.

### SymbolPaths

The value in this field is added to the semicolon-delimited list of folders in the SymbolPaths column of the [UpgradedImages Table \(Patchwiz.dll\)](#) when the patch is generated, and can be used to add symbol files for a specific file.

### AllowIgnoreOnPatchError

Set to 1 to indicate that the patch is non-vital. Set to 0 to indicate that the patch is vital. If Windows Installer encounters a problem when applying this patch to the file specified in the FTK column, the value in this field determines whether the error message box includes an **Ignore** button to enable the user to continue the patching process.

### IncludeWholeFile

Set to a nonzero value if the entire file specified in the FTK column should be installed rather than creating a binary patch.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# FamilyFileRanges Table (Patchwiz.dll)

The FamilyFileRanges table contains information about particular files of an upgraded image with ranges that should never be overwritten. This table is optional in the patch creation database (.pcp file) and is used by the [UiCreatePatchPackageEx](#) function.

The FamilyFileRanges table has the following columns.

Column	Type	Key	Nullable
Family	text	Y	N
FTK	text	Y	N
RetainOffsets	text		N
RetainLengths	text		N

## Columns

### Family

Foreign key to the Family column of the [ImageFamilies Table \(Patchwiz.dll\)](#).

### FTK

Foreign key into the [File tables](#) of all the upgraded images in the image family.

### RetainOffsets

The offset of the ranges that cannot be overwritten. The value in this field is a list of the range offset numbers for ranges that are not to be overwritten in the target files. The order and number of the ranges in the list must match the items in the RetainLengths column.

The values can be decimal or hexadecimal. [Patchwiz.dll](#) treats the value as hexadecimal if it is prefixed by "0x". The columns are string columns and Patchwiz.dll will convert the values to ULONGs.

### RetainLengths

The length in bytes of the ranges that cannot be overwritten. The value in this field is a list of range length numbers for ranges to retain in target files. The order and number of the ranges in the list must match the items in the RetainOffsets column.

The values can be decimal or hexadecimal. Patchwiz.dll treats the value as hexadecimal if it is prefixed by "0x". The columns are string columns and Patchwiz.dll will convert the values to ULONGs.

## Remarks

The offsets and lengths entered in RetainOffsets and RetainLengths must not specify overlapping ranges.

## See Also

[Patching Selected Regions of a File](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# TargetFiles\_OptionalData Table (Patchwiz.dll)

The TargetFiles\_OptionalData table contains information about specific files in a target image. This table is optional in the patch creation database (.pcp file) and is used by the [UiCreatePatchPackageEx](#) function.

The TargetFiles\_OptionalData table has the following columns.

Column	Type	Key	Nullable
Target	text	Y	N
FTK	text	Y	N
SymbolPaths	text		Y
IgnoreOffsets	text		Y
IgnoreLengths	text		Y
RetainOffsets	text		Y

## Columns

### Target

Foreign key to the Target column of the [TargetImages Table \(Patchwiz.dll\)](#).

### FTK

Foreign key into the [File table](#) of target image.

### SymbolPaths

The value in this field is added to the semicolon delimited list of folders in the SymbolPaths column of the [TargetImages Table \(Patchwiz.dll\)](#) when the patch is generated, and can be used to add symbol files for a specific file.

### IgnoreOffsets

The value in this field is a comma delimited list of range offset

numbers for the ranges to be ignored in the Target file. The order and number of the ranges in the list must match the items in the IgnoreLengths column. This column is optional.

The values can be decimal or hexadecimal. [Patchwiz.dll](#) treats the value as hexadecimal if it is prefixed by "0x". The columns are string columns and Patchwiz.dll will convert the values to ULONGs.

#### IgnoreLengths

The value in this field is a comma delimited list of range lengths in bytes for the ranges to be ignored in the Target file. The order and number of the ranges in the list must match the items in the IgnoreOffsets column. This column is optional.

The values can be decimal or hexadecimal. Patchwiz.dll treats the value as hexadecimal if it is prefixed by "0x". The columns are string columns and Patchwiz.dll will convert the values to ULONGs.

#### RetainOffsets

The value in this field is a comma delimited list of range offset numbers for the ranges to be retained in the Target file. The order and number of the ranges in the list must match the items in the RetainOffsets column of the corresponding record in the [FamilyFileRanges Table \(Patchwiz.dll\)](#)

The values can be decimal or hexadecimal. Patchwiz.dll treats the value as hexadecimal if it is prefixed by "0x". The columns are string columns and Patchwiz.dll will convert the values to ULONGs.

## See Also

[Patching Selected Regions of a File](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# ExternalFiles Table (Patchwiz.dll)

The ExternalFiles table contains information about specific files that are not part of a regular target image. These files may exist in products that have been updated by another product, upgrade, or patch. This table is optional in the patch creation database (.pcp file) and is used by the [UiCreatePatchPackageEx](#) function.

The ExternalFiles table has the following columns.

Column	Type	Key	Nullable
Family	text	Y	N
FTK	text	Y	N
FilePath	text	Y	N
SymbolPaths	text		Y
IgnoreOffsets	text		Y
IgnoreLengths	text		Y
RetainOffsets	text		N
Order	integer		Y

## Columns

### Family

Foreign key to the Family column of the [ImageFamilies Table \(Patchwiz.dll\)](#).

### FTK

Foreign key into [File table](#) of the .msi file of the upgraded image.

### FilePath

Full path of the external file including the file name. FilePath field is used to locate the file specified in the FTK column.

### SymbolPaths

Full path searched for symbol files of the file specified in the FTK

column.

### IgnoreOffsets

The value in this field is a comma-delimited list of range offset numbers for the ranges to be ignored in the external file. The order and number of the ranges in the list must match the items in the IgnoreLengths column. This column is optional.

The values can be decimal or hexadecimal. [Patchwiz.dll](#) treats the value as hexadecimal if it is prefixed by "0x". The columns are string columns and Patchwiz.dll will convert the values to ULONGs.

### IgnoreLengths

The value in this field is a comma-delimited list of range lengths in bytes for the ranges to be ignored in the external file. The order and number of the ranges in the list must match the items in the IgnoreOffsets column. This column is optional.

The values can be decimal or hexadecimal. Patchwiz.dll treats the value as hexadecimal if it is prefixed by "0x". The columns are string columns and Patchwiz.dll will convert the values to ULONGs.

### RetainOffsets

The value in this field is a comma-delimited list of range offset numbers for the ranges to be retained in the External file. The order and number of the ranges in the list must match the items in the RetainOffsets column of the corresponding record in the [FamilyFileRanges Table \(Patchwiz.dll\)](#).

The values can be decimal or hexadecimal. Patchwiz.dll treats the value as hexadecimal if it is prefixed by "0x". The columns are string columns and Patchwiz.dll will convert the values to ULONGs.

### Order

If two or more versions are specified for the same external file, the table may contain multiple records with matching values in the FTK and Family fields. In this case, the Order field may specify the order of external files to use when creating the patch. The order is from the oldest to the most recent version.

## Remarks

This table accepts environment variables as paths beginning with version 4.0 of Patchwiz.dll.

## **See Also**

[Patching Selected Regions of a File](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# UpgradedFilesToIgnore Table (Patchwiz.dll)

The UpgradedFilesToIgnore table prevents the updating of specific files that are in fact changed in the upgraded image relative to the target images. It may be useful to do this in certain cases. For example, a patch package intended only for use with non-administrative installations does not need to include patching files that are only part of administrative images. Excluding such files used only in administrative images can reduce the size of the patch package; however, administrators should be informed on how to update these files separately. This table is optional in the patch creation database (.pcp file) and is used by the [UiCreatePatchPackageEx](#) function.

The UpgradedFilesToIgnore table has the following columns.

Column	Type	Key	Nullable
Upgraded	text	Y	N
FTK	text	Y	N

## Columns

### Upgraded

Foreign key to the Upgraded column of the [UpgradedImages Table \(Patchwiz.dll\)](#). The patch creation tool excludes updating the file specified in the FTK column of the UpgradedFilesToIgnore table when upgrading a target to the image specified in the Upgraded field. Enter a value of "\*" in the Upgraded field to exclude updating the file for all upgraded images.

### FTK

Foreign key into the [File table](#) of the upgraded image. A value of the form "<prefix>\*" matches all file table keys in the File table that begin with that prefix. No text can follow the asterisk.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PatchMetadata Table (PATCHWIZ.DLL)

The PatchMetadata Table contains information about a Windows Installer patch that is required to remove a patch and that is used by Add/Remove Programs. All the properties in the PatchMetadata Table are added to the [MsiPatchMetadata Table](#) of the .msp file for a patch.

The PatchMetadata Table is required in patch creation properties files (.pcp files) that have a MinimumRequiredMsiVersion equal to 300 in the [Properties Table](#). The table is optional if MinimumRequiredMsiVersion is not equal to 300.

The PatchMetadata Table has the following columns.

Column	Type	Key	Nullable
Company	text	Y	Y
Property	text	Y	N
Value	text		Y

## Columns

### Company

The name of the company. An empty field (a Null value) indicates that this row contains one of the standard metadata properties. A company can extend the property set by adding a row to the table, and entering a company name in this field.

### Property

The name of a metadata property. The AllowRemoval, ManufacturerName, TargetProductName, MoreInfoURL, DisplayName, Description, and Classification properties are required in the PatchMetadata Table . This field must contain one of the following standard metadata properties if the Company field is empty (a Null value).

Property	Description
AllowRemoval	<p>An integer value that indicates whether or not the patch is an <b>Uninstallable Patch</b>. If the Value field contains a 0 (zero), the patch cannot be removed. If the Value field contains 1 (one), the patch is an Uninstallable Patch. This property is required.</p> <p>This property is registered and its value can be obtain by using the <b>MsiGetPatchInfoEx</b> function.</p>
ManufacturerName	<p>A string value that contains the name of the manufacturer of the application. This property is required.</p>
MinorUpdateTargetRTM	<p>Indicates that the patch targets the RTM version of the product or the most recent major upgrade patch.</p> <p>Author this optional property in minor upgrade patches that contain sequencing information to indicate that the patch removes of all patches up to the RTM version of the product, or up to the most recent major upgrade patch. This property is available beginning with Windows Installer 3.1.</p> <p><b>Note</b> To require that Windows Installer 3.1 be installed to apply the patch, set the <b>MinimumRequiredMsiVersion</b> property to 310 in the Properties Table of the .pcp file.</p>

**TargetProductName** A string value that contains the name of the application or target application suite. This property is required.

**MoreInfoURL** A string value that contains a URL pointing to information for this patch.

This required property is registered and its value can be obtained by using the **MsiGetPatchInfoEx** function. Beginning with Windows XP with Service Pack 2 (SP2), this value can be the support link for the patch displayed in Add/Remove Programs.

**CreationTimeUTC** A string value that contains the creation time of the .msp file in the form mm-dd-yy HH:MM (month-day-year hour:minute). This property is optional. **DisplayName** A string value that contains the title for the patch that is suitable for public display. This property is required.

This property is registered and its value can be obtained by using the **MsiGetPatchInfoEx** function. Beginning with Windows XP with SP2, this value is the name of the patch displayed in Add/Remove Programs beginning with Windows XP with SP2.

**Description** A string value that contains a brief description of the patch. This property is required. **Classification** A string value that contains the arbitrary category of updates as defined by the author of the patch. For example, patch authors can specify that each patch be classified as a Hotfix, Security Rollup, Critical Update, Update, Service Pack, or Update Rollup. This property is required.

**OptimizedInstallMode** If this property is set to 1 (one) in all the patches to be applied in a transaction, the application of the patch is optimized if possible. For information, see [Patch Optimization](#). Available beginning with Windows Installer 3.1.

## Value

Value of the metadata property. This can never be Null or an empty string. This value can be localized.

## Remarks

Available beginning in Windows Installer 3.0.

All properties authored into the PatchMetadata Table are added to the MsiPatchMetadata table of the msp file. AllowRemoval, MoreInfoURL and DisplayName properties are registered and are accessible through the **MsiGetPatchInfoEx**.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# PatchSequence Table (PATCHWIZ.DLL)

The PatchSequence Table is used to generate the [MsiPatchSequence Table](#) in a patch. The table requires the version of PATCHWIZ.DLL that is available with Windows Installer 3.0.

The following table identifies the columns of the PatchSequence Table.

Column	Type	Key	Nullable
PatchFamily	Identifier	Y	N
Target	Text	Y	Y
Sequence	Version		Y
Supersede	Integer		Y

## Columns

### PatchFamily

The identifier that indicates the sequence families to which this patch belongs.

The values in the Target and PatchFamily columns together define the primary key for the table. A patch that belongs to multiple sequence families, or has different sequences depending on the product code of the target, can have one row for each pairing. This value is used to populate the PatchFamily column of the MsiPatchSequence Table that belongs to the patch.

### Target

The Target column is used to filter the PatchFamily by product code.

A NULL value in this column indicates that this PatchFamily applies to all targets of the patch. If this column contains a foreign key to the [TargetImages Table](#), the product code of the specified image is retrieved and used to populate the product code value in the new patch's row of the MsiPatchSequence Table. If this column contains

a GUID, the GUID is used to populate the product code value of the row in the MsiPatchSequence Table.

#### Sequence

The value in the Sequence column is used to populate the Sequence column of the MsiPatchSequence Table of the new patch file.

If the value is NULL, a sequence number is generated automatically.

#### Supersede

A value of msidbPatchSequenceSupersedeEarlier or 1 in this field indicates that this patch supersedes earlier [small updates](#) in the sequence families to which this patch belongs.

The value in this column is used to set the Attributes column of the new patch's row in the MsiPatchSequence Table .

#### Remarks

Available beginning in Windows Installer 3.0.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Wilogutl.exe

Wilogutl.exe assists the analysis of log files from a Windows Installer installation, and it displays suggested solutions to errors that are found in a log file.

Non-critical errors are not displayed. Wilogutl.exe can be run in quiet mode or with a user interface (UI). The tool generates reports as text files in both the UI and quiet modes. It works best with verbose Windows Installer log files, but also works with non-verbose logs. For more information, see [Logging](#).

This tool is only available in the [Windows SDK Components for Windows Installer Developers](#).

## Syntax

**wilogutl.exe** [*<options>*][*<source file>*][*<options>*][*<report file directory>*]

You can use the following command lines to run in quiet mode.

```
wilogutl /q /l c:\mysilog.log /o c:\outputdir\
```

```
wilogutl /q /l c:\mysilog.log
```

## Command-Line Options

Wilogutl.exe uses the following case insensitive command-line options. A dash delimiter can be used in place of a slash.

Option	Description
none	Runs in UI mode—without command-line options.
/q	Specifies the quiet mode. Wilogutl.exe generates report files and does not display a user interface.
/l	Specifies the name of the log file to be analyzed. This option is required when using the quiet mode.
/o	Specifies the output directory for report files. This output path is used only when running in quiet mode. If the option is not present, the reports are put in the C:\WiLogResults directory.

When run in UI mode, Wilogutl.exe displays the following dialog boxes.

Name	Description
Windows Installer Verbose Log Analyzer	<p>The Windows Installer Verbose Log Analyzer dialog box enables users to select a log file for analysis:</p> <ul style="list-style-type: none"><li>• The <b>Open</b> button opens the file in Notepad. The preview area can be used to verify that the correct log file has been selected.</li><li>• The <b>Analyze</b> button begins log file analysis and displays the Detailed Log File View dialog box.</li></ul>
Detailed Log File View	<p>The Detailed Log File View dialog box displays logged error information. Use the <b>Back</b> and <b>Next</b> buttons to navigate through multiple errors.</p> <p>To display non-critical errors select the <b>Show Ignored Debug Errors</b> check box. The installer version on the computer used to run the logged installation is displayed. If the logged installation was run with elevated permissions, the <b>Elevated install</b> check box is selected and information is provided in the <b>Client Side Privilege Details</b> and <b>Server Side Privilege Details</b> text boxes. The Detailed Log File View dialog box contains the following buttons:</p> <ul style="list-style-type: none"><li>• <b>States</b> - Show the Feature and Component States dialog box.</li><li>• <b>Properties</b> - Show the Properties dialog box.</li><li>• <b>Policies</b> - Show the Policies dialog box.</li><li>• <b>HTML Annotated Log</b> - Show log as annotated HTML file.</li><li>• <b>Save Results</b> - Save report files to specified directory.</li><li>• <b>Error Message Help</b> - Show installer error message help.</li><li>• <b>Help</b> - Show help for Windows Installer Setup Log</li></ul>

	<p>Analyzer.</p> <ul style="list-style-type: none"> <li>• <b>How to Read a Log File</b> - Show the log file help document.</li> </ul>
<p>Feature and Component States</p>	<p>The <b>Feature and Component States</b> dialog box displays the states of features and components:</p> <ul style="list-style-type: none"> <li>• The <b>Feature</b> column shows the name for the feature in the installation package.</li> <li>• The <b>Component</b> column shows the name of the component in the installation package.</li> <li>• The <b>Installed</b> column shows the feature or component's state at the end of the installation.</li> <li>• The <b>Request</b> column shows the user's selection during the installation for the feature or component's state.</li> <li>• The <b>Action</b> column shows the action taken by the installer for the feature or component.</li> </ul> <p>For more information, see <a href="#">MsiGetComponentState</a> and <a href="#">MsiGetFeatureState</a>.</p>
<p>Properties</p>	<p>The Properties dialog box shows Windows Installer <a href="#">Properties</a> and their values at the end of the installation. You can sort the properties by name or by value:</p> <ul style="list-style-type: none"> <li>• The <b>Client</b> tab shows properties and values during the client side portion of the installation.</li> <li>• The <b>Server</b> tab shows properties and values during the server portion of the installation.</li> <li>• The <b>Nested</b> tab shows the properties and values of any <a href="#">Concurrent Installations</a>.</li> </ul>
<p>Policies</p>	<p>The Policies dialog box displays the <a href="#">System Policy</a> set after the installation:</p>

- A value of 0 (zero) set for the policy means the policy is not enabled.
- A value of 1 (one) means the policy is enabled.
- A value of ? (question mark) means the policy value is not recorded in the log.

If you need a policy value that is not in the log, try using Regedit.exe to check the registry keys on the computer failing the installation.

## Report Files

When performing a quiet mode analysis or clicking the **Save Results** button on the **Detailed Log File View** dialog, the Windows Installer Setup Analyzer tool generates three text files and an HTML annotated log file.

The following table identifies the names and contents in the report files.

Name	Description
logfile_name_summary.txt	Summarizes the log file. Lists the information displayed by the Detailed Log File View dialog box and the first error.
logfile_name_errors.txt	Identifies the number of errors, the errors, and recommended solutions. This file lists both critical and non-critical errors.
logfile_name_policies.txt	Identifies the policy names and values set at the end of the installation as in the Policies dialog box.
details_logfilename.htm	An HTML annotated log with a legend for the color coding.

## Return Values

If invalid command-line arguments are passed for quiet mode operations, Wilogutl.exe does nothing, and the process returns one of the values in the following table.

Value	Meaning
1	Bad output directory is specified.
2	Bad log file name is specified.
3	Passed /q, but is missing the required switch /l for the log file name.
4	Passed /l, but is missing the required switch /q for quiet mode.

## See Also

[Released Versions, Tools, and Redistributables  
Windows Installer Development Tools](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Errors Reference

For a complete list of the error codes returned by Windows Installer functions MsiExec.exe and InstMsi.exe, see [Error Codes](#). For a complete list of the error messages and error codes returned by Windows Installer, see [Windows Installer Error Messages](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Windows Installer Error Messages

Windows Installer errors have an error code of 1000 or greater. The error codes numbered 1000 to 1999 are ship errors and must be authored into the [Error table](#). The error codes numbered greater than 2000 are internal errors and do not have authored strings, but these can occur if the installation package has been incorrectly authored. For a list of reserved error codes, see [Error table](#).

**Note** If you are a user experiencing difficulty with your computer either during or after installing or uninstalling an application, you should contact customer support for the software you are trying to install or remove. If you feel you are in need of support for a Microsoft product, please go to our technical support site at [support.microsoft.com](http://support.microsoft.com).

**Note** You can search for solutions to many of the messages in the following table in the Microsoft Support Knowledge Base (KB). Go to the [Search the Support Knowledge Base](#) page and use the tool provided there to perform a search for a specific Windows Installer error message. Select "Search Product: All Products" to perform a comprehensive search for the message. In the "For:" pane, enter a character string like the following, with quotes enclosing the words Windows Installer, the appropriate *Message Code* value from the following table, and the keyword "kberrmsg".

"Windows Installer" *Message Code* kberrmsg

**Note** If you are viewing this documentation using the online MSDN library, you can also check the [Community Content](#) area at the bottom of this page to see whether any solutions for specific error messages have been contributed.

Developers of installation packages can also test the internal consistency of their packages by using Internal Consistency Evaluators. For more information, see [Internal Consistency Evaluators - ICEs](#).

See also the [Error Codes](#) returned by the Windows Installer functions MsiExec.exe and InstMsi.exe.

Message Code	Message	Remarks
--------------	---------	---------

1101	Could not open file stream: [2]. System error: [3]	
1301	Cannot create the file '[2]'. A directory with this name already exists.	
1302	Please insert the disk: [2]	
1303	The Installer has insufficient privileges to access this directory: [2].	
1304	Error writing to File: [2]	
1305	Error reading from File: [2]; System error code: [3]	
1306	The file '[2]' is in use. If you can, please close the application that is using the file, then click <b>Retry</b> .	A system restart may be required because a file being updated is also currently in use. For more information, see <a href="#">System Reboots</a> .
1307	There is not enough disk space remaining to install this file: [2]. If you can, free up some disk space, and click <b>Retry</b> , or click <b>Cancel</b> to exit.	
1308	Source file not found: [2]	
1309	Error attempting to open the source file: [3]. System error code: [2]	
1310	Error attempting to create the destination file: [3]. System error code: [2]	

1311	Could not locate source file cabinet: [2].	
1312	Cannot create the directory '[2]'. A file with this name already exists. Please rename or remove the file and click <b>Retry</b> , or click <b>Cancel</b> to exit.	
1313	The volume [2] is currently unavailable. Please select another.	
1314	The specified path '[2]' is unavailable.	
1315	Unable to write to the specified folder: [2].	
1316	A network error occurred while attempting to read from the file: [2]	
1317	An error occurred while attempting to create the directory: [2]	
1318	A network error occurred while attempting to create the directory: [2]	
1319	A network error occurred while attempting to open the source file cabinet: [2].	
1320	The specified path is too long: '[2]'	

1321	The Installer has insufficient privileges to modify this file: [2].	
1322	A portion of the folder path '[2]' is invalid. It is either empty or exceeds the length allowed by the system.	
1323	The folder path '[2]' contains words that are not valid in folder paths.	
1324	The folder path '[2]' contains an invalid character.	
1325	'[2]' is not a valid short file name.	
1326	Error getting file security: [3] GetLastError: [2]	
1327	Invalid Drive: [2]	
1328	Error applying patch to file [2]. It has probably been updated by other means, and can no longer be modified by this patch. For more information, contact your patch vendor. System Error: [3]	
1329	A file that is required cannot be installed because the cabinet file [2] is not digitally signed. This may indicate that the cabinet file is corrupt.	

1330	<p>A file that is required cannot be installed because the cabinet file [2] has an invalid digital signature. This may indicate that the cabinet file is corrupt. { Error [3] was returned by <b>WinVerifyTrust.</b> }</p>	
1331	<p>Failed to correctly copy [2] file: CRC error.</p>	
1332	<p>Failed to correctly move [2] file: CRC error.</p>	
1333	<p>Failed to correctly patch [2] file: CRC error.</p>	
1334	<p>The file '[2]' cannot be installed because the file cannot be found in cabinet file '[3]'. This could indicate a network error, an error reading from the CD-ROM, or a problem with this package.</p>	
1335	<p>The cabinet file '[2]' required for this installation is corrupt and cannot be used. This could indicate a network error, an error reading from the CD-ROM, or a problem with this package.</p>	
1336	<p>There was an error creating a temporary file that is needed to complete this</p>	

	installation. Folder: [3]. System error code: [2]	
1401	Could not create key: [2]. System error [3].	
1402	Could not open key: [2]. System error [3].	
1403	Could not delete value [2] from key [3]. System error [4].	
1404	Could not delete key [2]. System error [3].	
1405	Could not read value [2] from key [3]. System error [4].	
1406	Could not write value [2] to key [3]. System error [4].	
1407	Could not get value names for key [2]. System error [3].	
1408	Could not get sub key names for key [2]. System error [3].	
1409	Could not read security information for key [2]. System error [3].	
1410	Could not increase the available registry space. [2] KB of free registry space is required for the installation of this application.	
1500	Another installation is in	Test packages in high-traffic

	progress. You must complete that installation before continuing this one.	environments where users request the installation of many applications. For more information, see <a href="#">_MSIExecute Mutex</a> .
1501	Error accessing secured data. Please make sure the Windows Installer is configured properly and try the install again.	
1502	User '[2]' has previously initiated an install for product '[3]'. That user will need to run that install again before they can use that product. Your current install will now continue.	Test packages in high-traffic environments where users request the installation of many applications. For more information, see <a href="#">_MSIExecute Mutex</a> .
1503	User '[2]' has previously initiated an install for product '[3]'. That user will need to run that install again before they can use that product.	Test packages in high-traffic environments where users request the installation of many applications. For more information, see <a href="#">_MSIExecute Mutex</a> .
1601	Out of disk space -- Volume: '[2]'; required space: [3] KB; available space: [4] KB	Ensure that the custom action costs do not exceed available space.
1602	Are you sure you want to cancel?	
1603	The file [2][3] is being held in use by the following process: Name: [4], Id: [5], Window Title: '[6]'.	A system restart may be required because the file being updated is also currently in use. Users may be given the opportunity to avoid some system restarts by using the <a href="#">FilesInUse Dialog</a> or the

		<p><a href="#">MsiRMFilesInUse Dialog</a>. For more information, see <a href="#">System Reboots</a> and <a href="#">Logging of Reboot Requests</a>.</p>
1604	<p>The product '[2]' is already installed, and has prevented the installation of this product.</p>	
1605	<p>Out of disk space -- Volume: '[2]'; required space: [3] KB; available space: [4] KB. If rollback is disabled, enough space is available. Click <b>Cancel</b> to quit, <b>Retry</b> to check available disk space again, or <b>Ignore</b> to continue without rollback.</p>	<p>Ensure that the custom action costs do not exceed the available space.</p>
1606	<p>Could not access location [2].</p>	<p>Do not list directories in the <a href="#">Directory</a> table which are not used by the installation.</p> <p>Rarely, this message is due to the issue discussed by <a href="#">KB886549</a>.</p>
1607	<p>The following applications should be closed before continuing the install:</p>	<p>A system restart may be required because a file that is being updated is also currently in use. Users may be given the opportunity to avoid some system restarts by selecting to close some applications. For more information, see <a href="#">System Reboots</a>.</p>
1608	<p>Could not find any previously installed compliant products on the machine for installing this</p>	<p>No file listed in the <a href="#">CCPSearch</a> table can be found on the user's computer.</p>

	product	
1609	An error occurred while applying security settings. [2] is not a valid user or group. This could be a problem with the package, or a problem connecting to a domain controller on the network. Check your network connection and click <b>Retry</b> , or <b>Cancel</b> to end the install. Unable to locate the user's SID, system error [3]	
1610	The setup must update files or services that cannot be updated while the system is running. If you choose to continue, a reboot will be required to complete the setup.	Available in Windows Installer version 4.0.
1611	The setup was unable to automatically close all requested applications. Please ensure that the applications holding files in use are closed before continuing with the installation.	Available in Windows Installer version 4.0.
1651	Admin user failed to apply patch for a per-user managed or a per-machine application which is in advertise state.	Available in Windows Installer version 3.0.
1701	[2] is not a valid entry for a	

	product ID.	
1702	Configuring [2] cannot be completed until you restart your system. To restart now and resume configuration click <b>Yes</b> , or click <b>No</b> to stop this configuration.	A scheduled system restart message. For more information, see <a href="#">System Reboots</a> and <a href="#">ScheduleReboot Action</a> . This message may be customized using the Error table.
1703	For the configuration changes made to [2] to take effect you must restart your system. To restart now click <b>Yes</b> , or click <b>No</b> if you plan to manually restart at a later time.	The scheduled system restart message when no other users are logged on the computer. For more information, see <a href="#">System Reboots</a> and <a href="#">ScheduleReboot Action</a> . This message may be customized using the Error table.
1704	An install for [2] is currently suspended. You must undo the changes made by that install to continue. Do you want to undo those changes?	
1705	A previous install for this product is in progress. You must undo the changes made by that install to continue. Do you want to undo those changes?	
1706	No valid source could be found for product [2].	
1707	Installation operation completed successfully.	
1708	Installation operation failed.	
1709	Product: [2] -- [3]	

1710	You may either restore your computer to its previous state or continue the install later. Would you like to restore?	
1711	An error occurred while writing installation information to disk. Check to make sure enough disk space is available, and click <b>Retry</b> , or <b>Cancel</b> to end the install.	
1712	One or more of the files required to restore your computer to its previous state could not be found. Restoration will not be possible.	
1713	[2] cannot install one of its required products. Contact your technical support group. System Error: [3].	
1714	The older version of [2] cannot be removed. Contact your technical support group. System Error [3].	
1715	Installed [2].	
1716	Configured [2].	
1717	Removed [2].	
1718	File [2] was rejected by digital signature policy.	A very large installation may cause the operating system to run out of

		memory.
1719	Windows Installer service could not be accessed. Contact your support personnel to verify that it is properly registered and enabled.	
1720	There is a problem with this Windows Installer package. A script required for this install to complete could not be run. Contact your support personnel or package vendor. Custom action [2] script error [3], [4]: [5] Line [6], Column [7], [8]	
1721	There is a problem with this Windows Installer package. A program required for this install to complete could not be run. Contact your support personnel or package vendor. Action: [2], location: [3], command: [4]	
1722	There is a problem with this Windows Installer package. A program run as part of the setup did not finish as expected. Contact your support personnel or package vendor. Action [2], location: [3], command: [4]	
1723	There is a problem with this	Ensure that the functions used by

	Windows Installer package. A DLL required for this install to complete could not be run. Contact your support personnel or package vendor. Action [2], entry: [3], library: [4]	custom actions are actually exported. For more information about custom actions based upon a DLL, see <a href="#">Dynamic-Link Libraries</a> .
1724	Removal completed successfully.	
1725	Removal failed.	
1726	Advertisement completed successfully.	
1727	Advertisement failed.	
1728	Configuration completed successfully.	
1729	Configuration failed.	
1730	You must be an Administrator to remove this application. To remove this application, you can log on as an administrator, or contact your technical support group for assistance.	
1731	The source installation package for the product [2] is out of sync with the client package. Try the installation again using a valid copy of the installation package '[3]'.	Available beginning with Windows Installer for Windows Server 2003.
1732	In order to complete the	The scheduled system restart

	<p>installation of [2], you must restart the computer. Other users are currently logged on to this computer, and restarting may cause them to lose their work. Do you want to restart now?</p>	<p>message when other users are logged on the computer. For more information, see <a href="#">System Reboots</a> and ScheduleReboot Action. This message may be customized using the Error table.</p> <p>Available beginning with Windows Installer for Windows Server 2003.</p>
1801	The path [2] is not valid	
1802	Out of memory	
1803	There is no disk in drive [2]. Please, insert one and click <b>Retry</b> , or click <b>Cancel</b> to go back to the previously selected volume.	
1804	There is no disk in drive [2]. Please, insert one and click <b>Retry</b> , or click <b>Cancel</b> to return to the browse dialog and select a different volume.	
1805	The path [2] does not exist	
1806	You have insufficient privileges to read this folder.	
1807	A valid destination folder for the install could not be determined.	
1901	Error attempting to read from the source install database: [2]	

1902	Scheduling restart operation: Renaming file [2] to [3]. Must restart to complete operation.	An file being updated by the installation is currently in use. Windows Installer renames the file to update it and removes the old version at the next restart of the system.
1903	Scheduling restart operation: Deleting file [2]. Must restart to complete operation.	A system restart may be required because the file that is being updated is also currently in use. Users may be given the opportunity to avoid some system restarts by using the FilesInUse Dialog or MsiRMFilesInUse Dialog. For more information, see <a href="#">System Reboots</a> and Logging of Reboot Requests.
1904	Module [2] failed to register. HRESULT [3].	
1905	Module [2] failed to unregister. HRESULT [3].	
1906	Failed to cache package [2]. Error: [3]	
1907	Could not register font [2]. Verify that you have sufficient permissions to install fonts, and that the system supports this font.	
1908	Could not unregister font [2]. Verify that you have sufficient permissions to remove fonts.	
1909	Could not create shortcut	

	[2]. Verify that the destination folder exists and that you can access it.	
1910	Could not remove shortcut [2]. Verify that the shortcut file exists and that you can access it.	
1911	Could not register type library for file [2]. Contact your support personnel.	Error loading a type library or DLL.
1912	Could not unregister type library for file [2]. Contact your support personnel.	Error loading a type library or DLL.
1913	Could not update the .ini file [2][3]. Verify that the file exists and that you can access it.	
1914	Could not schedule file [2] to replace file [3] on restart. Verify that you have write permissions to file [3].	
1915	Error removing ODBC driver manager, ODBC error [2]: [3]. Contact your support personnel.	
1916	Error installing ODBC driver manager, ODBC error [2]: [3]. Contact your support personnel.	
1917	Error removing ODBC driver: [4], ODBC error [2]:	

	[3]. Verify that you have sufficient privileges to remove ODBC drivers.	
1918	Error installing ODBC driver: [4], ODBC error [2]: [3]. Verify that the file [4] exists and that you can access it.	
1919	Error configuring ODBC data source: [4], ODBC error [2]: [3]. Verify that the file [4] exists and that you can access it.	
1920	Service '[2]' ([3]) failed to start. Verify that you have sufficient privileges to start system services.	
1921	Service '[2]' ([3]) could not be stopped. Verify that you have sufficient privileges to stop system services.	
1922	Service '[2]' ([3]) could not be deleted. Verify that you have sufficient privileges to remove system services.	
1923	Service '[2]' ([3]) could not be installed. Verify that you have sufficient privileges to install system services.	
1924	Could not update environment variable '[2]'. Verify that you have	

	sufficient privileges to modify environment variables.	
1925	You do not have sufficient privileges to complete this installation for all users of the machine. Log on as administrator and then retry this installation.	
1926	Could not set file security for file '[3]'. Error: [2]. Verify that you have sufficient privileges to modify the security permissions for this file.	
1927	The installation requires COM+ Services to be installed.	
1928	The installation failed to install the COM+ Application.	
1929	The installation failed to remove the COM+ Application.	
1930	The description for service '[2]' ([3]) could not be changed.	
1931	The Windows Installer service cannot update the system file [2] because the file is protected by Windows. You may need to	Windows Installer protects critical system files. For more information, see <a href="#">Using Windows Installer and Windows Resource Protection</a> . For Windows Me, see the

	<p>update your operating system for this program to work correctly. Package version: [3], OS Protected version: [4]</p>	<p><a href="#">InstallSFPCatalogFile action</a>, the <a href="#">FileSFPCatalog table</a>, and the <a href="#">SFPCatalog table</a>.</p>
1932	<p>The Windows Installer service cannot update the protected Windows file [2]. Package version: [3], OS Protected version: [4], SFP Error: [5]</p>	<p>Windows Installer protects critical system files. For more information, see <a href="#">Using Windows Installer and Windows Resource Protection</a>. For Windows Me, see the <a href="#">InstallSFPCatalogFile action</a>, the <a href="#">FileSFPCatalog table</a>, and the <a href="#">SFPCatalog table</a>.</p>
1933	<p>The Windows Installer service cannot update one or more protected Windows files. SFP Error: [2]. List of protected files:\r\n[3]</p>	<p>Windows Installer protects critical system files. For more information, see <a href="#">Using Windows Installer and Windows Resource Protection</a>. For Windows Me, see the <a href="#">InstallSFPCatalogFile action</a>, the <a href="#">FileSFPCatalog table</a>, and the <a href="#">SFPCatalog table</a>.</p>
1934	<p>User installations are disabled through policy on the machine.</p>	
1935	<p>An error occurred during the installation of assembly component [2]. HRESULT: [3]. {{assembly interface: [4], function: [5], assembly name: [6]}}</p>	<p>For more information, see <a href="#">Assemblies</a>.</p> <p>Help and Support may have published a KB article that discusses the installation of this assembly. Go to the <a href="#">Search the Support Knowledge Base</a> page and search for articles that discuss this Windows Installer error message.</p>

1935	<p>An error occurred during the installation of assembly '[6]'. Please refer to Help and Support for more information. HRESULT: [3]. {{assembly interface: [4], function: [5], component: [2]}}</p>	<p>For more information, see <a href="#">Assemblies</a>.</p> <p>Help and Support may have published a KB article that discusses the installation of this assembly. Go to the <a href="#">Search the Support Knowledge Base</a> page and search for articles that discuss this Windows Installer error message.</p> <p>Available beginning with Windows Installer for Windows Server 2003.</p>
1936	<p>An error occurred during the installation of assembly '[6]'. The assembly is not strongly named or is not signed with the minimal key length. HRESULT: [3]. {{assembly interface: [4], function: [5], component: [2]}}</p>	<p>For more information, see <a href="#">Assemblies</a>.</p> <p>Help and Support may have published a KB article that discusses the installation of this assembly. Go to the <a href="#">Search the Support Knowledge Base</a> page and search for articles that discuss this Windows Installer error message.</p> <p>Available beginning with Windows Installer for Windows Server 2003.</p>
1937	<p>An error occurred during the installation of assembly '[6]'. The signature or catalog could not be verified or is not valid. HRESULT: [3]. {{assembly interface: [4], function: [5], component: [2]}}</p>	<p>For more information, see <a href="#">Assemblies</a>.</p> <p>Help and Support may have published a KB article that discusses the installation of this assembly. Go to the <a href="#">Search the Support Knowledge Base</a> page and search for articles that discuss this Windows Installer error message.</p> <p>Available beginning with Windows Installer for Windows Server 2003.</p>

1938	<p>An error occurred during the installation of assembly '[6]'. One or more modules of the assembly could not be found. HRESULT: [3]. {{assembly interface: [4], function: [5], component: [2]}}</p>	<p>For more information, see <a href="#">Assemblies</a>.</p> <p>Help and Support may have published a KB article that discusses the installation of this assembly. Go to the <a href="#">Search the Support Knowledge Base</a> page and search for articles that discuss this Windows Installer error message.</p> <p>Available beginning with Windows Installer for Windows Server 2003.</p>
1939	<p>Service '[2]' ([3]) could not be configured. This could be a problem with the package or your permissions. Verify that you have sufficient privileges to configure system services.</p>	<p>For information, see <a href="#">Using Services Configuration</a>.</p> <p>Available beginning with Windows Installer 5.0 for Windows 7 and Windows Server 2008 R2.</p>
1940	<p>Service '[2]' ([3]) could not be configured. Configuring services is supported only on Windows Vista/Server 2008 and above.</p>	<p>For information, see <a href="#">Using Services Configuration</a>.</p> <p>Available beginning with Windows Installer 5.0 for Windows 7 and Windows Server 2008 R2.</p>
1941	<p>Both LockPermissions and MsiLockPermissionsEx tables were found in the package. Only one of them should be present. This is a problem with the package.</p>	<p>A package cannot contain both the <a href="#">MsiLockPermissionsEx Table</a> and the <a href="#">LockPermissions Table</a>.</p> <p>Available beginning with Windows Installer 5.0 for Windows 7 and Windows Server 2008 R2.</p>
1942	<p>Multiple conditions ('[2]' and '[3]') have resolved to true while installing Object [4] (from table [5]). This</p>	<p>Available beginning with Windows Installer 5.0 for Windows 7 and Windows Server 2008 R2.</p>

	may be a problem with the package.	
1943	SDDL string '[2]' for object [3](in table [4]) could not be resolved into a valid Security Descriptor.	See <a href="#">Securing Resources</a> for information on using <a href="#">MsiLockPermissionsEx</a> table.  Available beginning with Windows Installer 5.0 for Windows 7 and Windows Server 2008 R2.
1944	Could not set security for service '[3]'. Error: [2]. Verify that you have sufficient privileges to modify the security permissions for this service.	Available beginning with Windows Installer 5.0 for Windows 7 and Windows Server 2008 R2.
1945	You do not have sufficient privileges to complete the re-advertisement of this product. Re-advertisement requires initiation by a local system account calling the MsiAdvertiseScript API	The process calling <a href="#">MsiAdvertiseScript</a> must be running under the LocalSystem account.  Available beginning with Windows Installer 5.0 for Windows 7 and Windows Server 2008 R2.
2101	Shortcuts not supported by the operating system.	
2102	Invalid .ini action: [2]	
2103	Could not resolve path for shell folder [2].	
2104	Writing .ini file: [3]: System error: [2].	
2105	Shortcut Creation [3] Failed. System error: [2].	

2106	Shortcut Deletion [3] Failed. System error: [2].	
2107	Error [3] registering type library [2].	
2108	Error [3] unregistering type library [2].	
2109	Section missing for .ini action.	
2110	Key missing for .ini action.	
2111	Detection of running applications failed, could not get performance data. Registered operation returned : [2].	
2112	Detection of running applications failed, could not get performance index. Registered operation returned : [2].	
2113	Detection of running applications failed.	
2200	Database: [2]. Database object creation failed, mode = [3].	
2201	Database: [2]. Initialization failed, out of memory.	
2202	Database: [2]. Data access failed, out of memory.	
2203	Database: [2]. Cannot open	

	database file. System error [3].	
2204	Database: [2]. Table already exists: [3].	
2205	Database: [2]. Table does not exist: [3].	
2206	Database: [2]. Table could not be dropped: [3].	
2207	Database: [2]. Intent violation.	
2208	Database: [2]. Insufficient parameters for Execute.	
2209	Database: [2]. Cursor in invalid state.	
2210	Database: [2]. Invalid update data type in column [3].	
2211	Database: [2]. Could not create database table [3].	
2212	Database: [2]. Database not in writable state.	
2213	Database: [2]. Error saving database tables.	
2214	Database: [2]. Error writing export file: [3].	
2215	Database: [2]. Cannot open import file: [3].	
2216	Database: [2]. Import file	

	format error: [3], Line [4].	
2217	Database: [2]. Wrong state to CreateOutputDatabase [3].	
2218	Database: [2]. Table name not supplied.	
2219	Database: [2]. Invalid Installer database format.	
2220	Database: [2]. Invalid row/field data.	
2221	Database: [2]. Code page conflict in import file: [3].	
2222	Database: [2]. Transform or merge code page [3] differs from database code page [4].	
2223	Database: [2]. Databases are the same. No transform generated.	
2224	Database: [2]. GenerateTransform: Database corrupt. Table: [3].	
2225	Database: [2]. Transform: Cannot transform a temporary table. Table: [3].	
2226	Database: [2]. Transform failed.	
2227	Database: [2]. Invalid identifier '[3]' in SQL query:	

	[4].	
2228	Database: [2]. Unknown table '[3]' in SQL query: [4].	
2229	Database: [2]. Could not load table '[3]' in SQL query: [4].	
2230	Database: [2]. Repeated table '[3]' in SQL query: [4].	
2231	Database: [2]. Missing ')' in SQL query: [3].	
2232	Database: [2]. Unexpected token '[3]' in SQL query: [4].	
2233	Database: [2]. No columns in SELECT clause in SQL query: [3].	
2234	Database: [2]. No columns in ORDER BY clause in SQL query: [3].	
2235	Database: [2]. Column '[3]' not present or ambiguous in SQL query: [4].	
2236	Database: [2]. Invalid operator '[3]' in SQL query: [4].	
2237	Database: [2]. Invalid or missing query string: [3].	
2238	Database: [2]. Missing FROM clause in SQL query: [3].	

2239	Database: [2]. Insufficient values in INSERT SQL statement.	
2240	Database: [2]. Missing update columns in UPDATE SQL statement.	
2241	Database: [2]. Missing insert columns in INSERT SQL statement.	
2242	Database: [2]. Column '[3]' repeated.	
2243	Database: [2]. No primary columns defined for table creation.	
2244	Database: [2]. Invalid type specifier '[3]' in SQL query [4].	
2245	IStorage::Stat failed with error [3].	
2246	Database: [2]. Invalid Installer transform format.	
2247	Database: [2] Transform stream read/write failure.	
2248	Database: [2] GenerateTransform/Merge: Column type in base table does not match reference table. Table: [3] Col #: [4].	

2249	Database: [2] GenerateTransform: More columns in base table than in reference table. Table: [3].	
2250	Database: [2] Transform: Cannot add existing row. Table: [3].	
2251	Database: [2] Transform: Cannot delete row that does not exist. Table: [3].	
2252	Database: [2] Transform: Cannot add existing table. Table: [3].	
2253	Database: [2] Transform: Cannot delete table that does not exist. Table: [3].	
2254	Database: [2] Transform: Cannot update row that does not exist. Table: [3].	
2255	Database: [2] Transform: Column with this name already exists. Table: [3] Col: [4].	
2256	Database: [2] GenerateTransform/Merge: Number of primary keys in base table does not match reference table. Table: [3].	
2257	Database: [2]. Intent to modify read only table: [3].	

2258	Database: [2]. Type mismatch in parameter: [3].	
2259	Database: [2] Table(s) Update failed	Queries must adhere to the restricted Windows Installer <a href="#">SQL syntax</a> .
2260	Storage CopyTo failed. System error: [3].	
2261	Could not remove stream [2]. System error: [3].	
2262	Stream does not exist: [2]. System error: [3].	
2263	Could not open stream [2]. System error: [3].	
2264	Could not remove stream [2]. System error: [3].	
2265	Could not commit storage. System error: [3].	
2266	Could not rollback storage. System error: [3].	
2267	Could not delete storage [2]. System error: [3].	
2268	Database: [2]. Merge: There were merge conflicts reported in [3] tables.	
2269	Database: [2]. Merge: The column count differed in the '[3]' table of the two databases.	

2270	Database: [2]. GenerateTransform/Merge: Column name in base table does not match reference table. Table: [3] Col #: [4].	
2271	SummaryInformation write for transform failed.	
2272	Database: [2]. MergeDatabase will not write any changes because the database is open read- only.	
2273	Database: [2]. MergeDatabase: A reference to the base database was passed as the reference database.	
2274	Database: [2]. MergeDatabase: Unable to write errors to Error table. Could be due to a non- nullable column in a predefined Error table.	
2275	Database: [2]. Specified Modify [3] operation invalid for table joins.	
2276	Database: [2]. Code page [3] not supported by the system.	
2277	Database: [2]. Failed to save table [3].	
2278	Database: [2]. Exceeded	

	number of expressions limit of 32 in WHERE clause of SQL query: [3].	
2279	Database: [2] Transform: Too many columns in base table [3].	
2280	Database: [2]. Could not create column [3] for table [4].	
2281	Could not rename stream [2]. System error: [3].	
2282	Stream name invalid [2].	
2302	Patch notify: [2] bytes patched to far.	
2303	Error getting volume info. GetLastError: [2].	
2304	Error getting disk free space. GetLastError: [2]. Volume: [3].	
2305	Error waiting for patch thread. GetLastError: [2].	
2306	Could not create thread for patch application. GetLastError: [2].	
2307	Source file key name is null.	
2308	Destination file name is null.	
2309	Attempting to patch file [2] when patch already in	

	progress.	
2310	Attempting to continue patch when no patch is in progress.	
2315	Missing path separator: [2].	
2318	File does not exist: [2].	
2319	Error setting file attribute: [3] GetLastError: [2].	
2320	File not writable: [2].	
2321	Error creating file: [2].	
2322	User canceled.	
2323	Invalid file attribute.	
2324	Could not open file: [3] GetLastError: [2].	
2325	Could not get file time for file: [3] GetLastError: [2].	
2326	Error in FileToDosDateTime.	
2327	Could not remove directory: [3] GetLastError: [2].	
2328	Error getting file version info for file: [2].	
2329	Error deleting file: [3]. GetLastError: [2].	
2330	Error getting file attributes:	

	[3]. GetLastError: [2].	
2331	Error loading library [2] or finding entry point [3].	
2332	Error getting file attributes. GetLastError: [2].	
2333	Error setting file attributes. GetLastError: [2].	
2334	Error converting file time to local time for file: [3]. GetLastError: [2].	
2335	Path: [2] is not a parent of [3].	
2336	Error creating temp file on path: [3]. GetLastError: [2].	
2337	Could not close file: [3] GetLastError: [2].	
2338	Could not update resource for file: [3] GetLastError: [2].	
2339	Could not set file time for file: [3] GetLastError: [2].	
2340	Could not update resource for file: [3], Missing resource.	
2341	Could not update resource for file: [3], Resource too large.	

2342	Could not update resource for file: [3] GetLastError: [2].	
2343	Specified path is empty.	
2344	Could not find required file IMAGEHLP.DLL to validate file:[2].	
2345	[2]: File does not contain a valid checksum value.	
2347	User ignore.	
2348	Error attempting to read from cabinet stream.	
2349	Copy resumed with different info.	
2350	FDI server error	
2351	File key '[2]' not found in cabinet '[3]'. The installation cannot continue.	
2352	Could not initialize cabinet file server. The required file 'CABINET.DLL' may be missing.	
2353	Not a cabinet.	
2354	Cannot handle cabinet.	
2355	Corrupt cabinet.	
2356	Could not locate cabinet in stream: [2].	When troubleshooting embedded streams, you may use

		<a href="#">WiStream.vbs</a> to list the streams and use <a href="#">Msidb.exe</a> to export the streams.
2357	Cannot set attributes.	
2358	Error determining whether file is in-use: [3]. GetLastError: [2].	
2359	Unable to create the target file - file may be in use.	
2360	Progress tick.	
2361	Need next cabinet.	
2362	Folder not found: [2].	
2363	Could not enumerate subfolders for folder: [2].	
2364	Bad enumeration constant in CreateCopier call.	
2365	Could not BindImage exe file [2].	
2366	User failure.	
2367	User abort.	
2368	Failed to get network resource information. Error [2], network path [3]. Extended error: network provider [5], error code [4], error description [6].	

2370	Invalid CRC checksum value for [2] file. { Its header says [3] for checksum, its computed value is [4]. }	
2371	Could not apply patch to file [2]. GetLastError: [3].	
2372	Patch file [2] is corrupt or of an invalid format. Attempting to patch file [3]. GetLastError: [4].	
2373	File [2] is not a valid patch file.	
2374	File [2] is not a valid destination file for patch file [3].	
2375	Unknown patching error: [2].	
2376	Cabinet not found.	
2379	Error opening file for read: [3] GetLastError: [2].	
2380	Error opening file for write: [3]. GetLastError: [2].	
2381	Directory does not exist: [2].	
2382	Drive not ready: [2].	
2401	64-bit registry operation attempted on 32-bit operating system for key [2].	
2402	Out of memory.	

2501	Could not create rollback script enumerator.	
2502	Called InstallFinalize when no install in progress.	
2503	Called RunScript when not marked in progress.	
2601	Invalid value for property [2]: '[3]'	
2602	The [2] table entry '[3]' has no associated entry in the Media table.	
2603	Duplicate table name [2].	
2604	[2] Property undefined.	
2605	Could not find server [2] in [3] or [4].	
2606	Value of property [2] is not a valid full path: '[3]'.	
2607	Media table not found or empty (required for installation of files).	
2608	Could not create security descriptor for object. Error: '[2]'.	
2609	Attempt to migrate product settings before initialization.	
2611	The file [2] is marked as compressed, but the	

	associated media entry does not specify a cabinet.	
2612	Stream not found in '[2]' column. Primary key: '[3]'.	
2613	RemoveExistingProducts action sequenced incorrectly.	
2614	Could not access IStorage object from installation package.	
2615	Skipped unregistration of Module [2] due to source resolution failure.	
2616	Companion file [2] parent missing.	
2617	Shared component [2] not found in Component table.	
2618	Isolated application component [2] not found in Component table.	
2619	Isolated components [2], [3] not part of same feature.	
2620	Key file of isolated application component [2] not in File table.	
2621	Resource DLL or Resource ID information for shortcut [2] set incorrectly.	Available with Windows Installer version 4.0.
2701	The depth of a feature	The maximum depth of any feature

	exceeds the acceptable tree depth of [2] levels.	is 16. This error is returned if a feature that exceeds the maximum depth exists.
2702	A Feature table record ([2]) references a non-existent parent in the Attributes field.	
2703	Property name for root source path not defined: [2]	
2704	Root directory property undefined: [2]	
2705	Invalid table: [2]; Could not be linked as tree.	
2706	Source paths not created. No path exists for entry [2] in Directory table.	
2707	Target paths not created. No path exists for entry [2] in Directory table.	
2708	No entries found in the file table.	
2709	The specified Component name ('[2]') not found in Component table.	
2710	The requested 'Select' state is illegal for this Component.	
2711	The specified Feature name ('[2]') not found in Feature table.	

2712	Invalid return from modeless dialog: [3], in action [2].	
2713	Null value in a non-nullable column ('[2]' in '[3]' column of the '[4]' table.	
2714	Invalid value for default folder name: [2].	
2715	The specified File key ('[2]') not found in the File table.	
2716	Could not create a random subcomponent name for component '[2]'.  May occur if the first 40 characters of two or more component names are identical. Ensure that the first 40 characters of component names are unique to the component.	
2717	Bad action condition or error calling custom action '[2]'.  May occur if the first 40 characters of two or more component names are identical. Ensure that the first 40 characters of component names are unique to the component.	
2718	Missing package name for product code '[2]'.  May occur if the first 40 characters of two or more component names are identical. Ensure that the first 40 characters of component names are unique to the component.	
2719	Neither UNC nor drive letter path found in source '[2]'.  May occur if the first 40 characters of two or more component names are identical. Ensure that the first 40 characters of component names are unique to the component.	
2720	Error opening source list key. Error: '[2]'  May occur if the first 40 characters of two or more component names are identical. Ensure that the first 40 characters of component names are unique to the component.	
2721	Custom action [2] not found in Binary table stream.  May occur if the first 40 characters of two or more component names are identical. Ensure that the first 40 characters of component names are unique to the component.	
2722	Custom action [2] not found in File table.  May occur if the first 40 characters of two or more component names are identical. Ensure that the first 40 characters of component names are unique to the component.	
2723	Custom action [2] specifies unsupported type.  May occur if the first 40 characters of two or more component names are identical. Ensure that the first 40 characters of component names are unique to the component.	

2724	The volume label '[2]' on the media you're running from does not match the label '[3]' given in the Media table. This is allowed only if you have only 1 entry in your Media table.	
2725	Invalid database tables	
2726	Action not found: [2].	
2727	The directory entry '[2]' does not exist in the Directory table.	
2728	Table definition error: [2]	
2729	Install engine not initialized.	
2730	Bad value in database. Table: '[2]'; Primary key: '[3]'; Column: '[4]'	
2731	Selection Manager not initialized.	The selection manager is responsible for determining component and feature states. It is initialized during the costing actions ( <a href="#">CostInitialize action</a> , <a href="#">FileCost action</a> , and <a href="#">CostFinalize action</a> .) A standard action or custom action made a call to a function requiring the selection manager before the initialization of the selection manager. This action should be sequenced after the costing actions.

2732	Directory Manager not initialized.	The directory manager is responsible for determining the target and source paths. It is initialized during the costing actions ( <a href="#">CostInitialize action</a> , <a href="#">FileCost action</a> , and <a href="#">CostFinalize action</a> ). A standard action or custom action made a call to a function requiring the directory manager before the initialization of the directory manager. This action should be sequenced after the costing actions.
2733	Bad foreign key ('[2]') in '[3]' column of the '[4]' table.	
2734	Invalid reinstall mode character.	
2735	Custom action '[2]' has caused an unhandled exception and has been stopped. This may be the result of an internal error in the custom action, such as an access violation.	This error is caused by a custom action that is based on <a href="#">Dynamic-Link Libraries</a> . When troubleshooting the DLL you may need to use one or more of the tools described in <a href="#">KB198038</a> .
2736	Generation of custom action temp file failed: [2].	This error is caused by a custom action that is based on <a href="#">Dynamic-Link Libraries</a> . When troubleshooting the DLL you may need to use one or more of the tools described in <a href="#">KB198038</a> .
2737	Could not access custom action [2], entry [3], library [4]	This error is caused by a custom action that is based on <a href="#">Dynamic-Link Libraries</a> . When troubleshooting the DLL you may need to

		use one or more of the tools described in <a href="#">KB198038</a> .
2738	Could not access VBScript run time for custom action [2].	This error is caused by a custom action that is based on <a href="#">Dynamic-Link Libraries</a> . When troubleshooting the DLL you may need to use one or more of the tools described in <a href="#">KB198038</a> .
2739	Could not access JScript run time for custom action [2].	This error is caused by a custom action that is based on <a href="#">Dynamic-Link Libraries</a> . When troubleshooting the DLL you may need to use one or more of the tools described in <a href="#">KB198038</a> .
2740	Custom action [2] script error [3], [4]: [5] Line [6], Column [7], [8].	This error is caused by a custom action that is based on <a href="#">Dynamic-Link Libraries</a> . When troubleshooting the DLL you may need to use one or more of the tools described in <a href="#">KB198038</a> .
2741	Configuration information for product [2] is corrupt. Invalid info: [2].	This error is caused by a custom action that is based on <a href="#">Dynamic-Link Libraries</a> . When troubleshooting the DLL you may need to use one or more of the tools described in <a href="#">KB198038</a> .
2742	Marshaling to Server failed: [2].	This error is caused by a custom action that is based on <a href="#">Dynamic-Link Libraries</a> . When troubleshooting the DLL you may need to use one or more of the tools described in <a href="#">KB198038</a> .
2743	Could not execute custom	This error is caused by a custom

	action [2], location: [3], command: [4].	action that is based on <a href="#">Dynamic-Link Libraries</a> . When troubleshooting the DLL you may need to use one or more of the tools described in <a href="#">KB198038</a> .
2744	EXE failed called by custom action [2], location: [3], command: [4].	This error is caused by a custom action that is based on <a href="#">Dynamic-Link Libraries</a> . When troubleshooting the DLL you may need to use one or more of the tools described in <a href="#">KB198038</a> .
2745	Transform [2] invalid for package [3]. Expected language [4], found language [5].	The language ID that is specified by the <b>ProductLanguage</b> property must be contained in the <b>Template Summary</b> property. Perform <a href="#">package validation</a> and check for <a href="#">ICE80</a> .
2746	Transform [2] invalid for package [3]. Expected product [4], found product [5].	
2747	Transform [2] invalid for package [3]. Expected product version < [4], found product version [5].	
2748	Transform [2] invalid for package [3]. Expected product version <= [4], found product version [5].	
2749	Transform [2] invalid for package [3]. Expected product version == [4], found product version [5].	

2750	Transform [2] invalid for package [3]. Expected product version $\geq$ [4], found product version [5].	
2751	Transform [2] invalid for package [3]. Expected product version $>$ [4], found product version [5].	
2752	Could not open transform [2] stored as child storage of package [4].	
2753	The File '[2]' is not marked for installation.	
2754	The File '[2]' is not a valid patch file.	
2755	Server returned unexpected error [2] attempting to install package [3].	
2756	The property '[2]' was used as a directory property in one or more tables, but no value was ever assigned.	
2757	Could not create summary info for transform [2].	
2758	Transform [2] does not contain an MSI version.	
2759	Transform [2] version [3] incompatible with engine; Min: [4], Max: [5].	

2760	Transform [2] invalid for package [3]. Expected upgrade code [4], found [5].	
2761	Cannot begin transaction. Global mutex not properly initialized.	
2762	Cannot write script record. Transaction not started.	The <a href="#">InstallExecuteSequence</a> may have been authored incorrectly. Actions that change the system must be sequenced between the <a href="#">InstallInitialize</a> and <a href="#">InstallFinalize</a> actions. Perform <a href="#">package validation</a> and check for <a href="#">ICE77</a> .
2763	Cannot run script. Transaction not started.	
2765	Assembly name missing from AssemblyName table : Component: [4].	
2766	The file [2] is an invalid MSI storage file.	
2767	No more data{ while enumerating [2]}.	
2768	Transform in patch package is invalid.	
2769	Custom Action [2] did not close [3] MSIHANDLES.	The <a href="#">InstallExecuteSequence</a> may have been authored incorrectly. Actions that change the system must be sequenced between the <a href="#">InstallInitialize</a> and <a href="#">InstallFinalize</a> actions. Perform <a href="#">package validation</a> and check for <a href="#">ICE77</a> .

2770	Cached folder [2] not defined in internal cache folder table.	
2771	Upgrade of feature [2] has a missing component. .	Available beginning with Windows Installer version 3.0.
2772	New upgrade feature [2] must be a leaf feature.	Available beginning with Windows Installer version 3.0.
2801	Unknown Message -- Type [2]. No action is taken.	
2802	No publisher is found for the event [2].	
2803	Dialog View did not find a record for the dialog [2].	
2804	On activation of the control [3] on dialog [2] CMsiDialog failed to evaluate the condition [3].	
2805		
2806	The dialog [2] failed to evaluate the condition [3].	
2807	The action [2] is not recognized.	
2808	Default button is ill-defined on dialog [2].	
2809	On the dialog [2] the next control pointers do not form a cycle. There is a pointer from [3] to [4], but there is	

	no further pointer.	
2810	On the dialog [2] the next control pointers do not form a cycle. There is a pointer from both [3] and [5] to [4].	
2811	On dialog [2] control [3] has to take focus, but it is unable to do so.	
2812	The event [2] is not recognized.	
2813	The EndDialog event was called with the argument [2], but the dialog has a parent	
2814	On the dialog [2] the control [3] names a nonexistent control [4] as the next control.	
2815	ControlCondition table has a row without condition for the dialog [2].	
2816	The EventMapping table refers to an invalid control [4] on dialog [2] for the event [3].	
2817	The event [2] failed to set the attribute for the control [4] on dialog [3].	
2818	In the ControlEvent table EndDialog has an unrecognized argument [2].	

2819	Control [3] on dialog [2] needs a property linked to it.	
2820	Attempted to initialize an already initialized handler.	
2821	Attempted to initialize an already initialized dialog: [2].	
2822	No other method can be called on dialog [2] until all the controls are added.	
2823	Attempted to initialize an already initialized control: [3] on dialog [2].	
2824	The dialog attribute [3] needs a record of at least [2] field(s).	
2825	The control attribute [3] needs a record of at least [2] field(s).	
2826	Control [3] on dialog [2] extends beyond the boundaries of the dialog [4] by [5] pixels.	
2827	The button [4] on the radio button group [3] on dialog [2] extends beyond the boundaries of the group [5] by [6] pixels.	
2828	Tried to remove control [3] from dialog [2], but the	

	control is not part of the dialog.	
2829	Attempt to use an uninitialized dialog.	
2830	Attempt to use an uninitialized control on dialog [2].	
2831	The control [3] on dialog [2] does not support [5] the attribute [4].	
2832	The dialog [2] does not support the attribute [3].	
2833	Control [4] on dialog [3] ignored the message [2].	
2834	The next pointers on the dialog [2] do not form a single loop.	
2835	The control [2] was not found on dialog [3].	
2836	The control [3] on the dialog [2] cannot take focus.	
2837	The control [3] on dialog [2] wants the winproc to return [4].	
2838	The item [2] in the selection table has itself as a parent.	
2839	Setting the property [2] failed.	

2840	Error dialog name mismatch.	
2841	No OK button was found on the error dialog.	
2842	No text field was found on the error dialog.	
2843	The ErrorString attribute is not supported for standard dialogs.	
2844	Cannot execute an error dialog if the Errorstring is not set.	
2845	The total width of the buttons exceeds the size of the error dialog.	
2846	SetFocus did not find the required control on the error dialog.	
2847	The control [3] on dialog [2] has both the icon and the bitmap style set.	
2848	Tried to set control [3] as the default button on dialog [2], but the control does not exist.	
2849	The control [3] on dialog [2] is of a type, that cannot be integer valued.	
2850	Unrecognized volume type.	

2851	The data for the icon [2] is not valid.	
2852	At least one control has to be added to dialog [2] before it is used.	
2853	Dialog [2] is a modeless dialog. The execute method should not be called on it.	
2854	On the dialog [2] the control [3] is designated as first active control, but there is no such control.	
2855	The radio button group [3] on dialog [2] has fewer than 2 buttons.	
2856	Creating a second copy of the dialog [2].	
2857	The directory [2] is mentioned in the selection table but not found.	
2858	The data for the bitmap [2] is not valid.	
2859	Test error message.	
2860	Cancel button is ill-defined on dialog [2].	
2861	The next pointers for the radio buttons on dialog [2] control [3] do not form a cycle.	

2862	The attributes for the control [3] on dialog [2] do not define a valid icon size. Setting the size to 16.	
2863	The control [3] on dialog [2] needs the icon [4] in size [5]x[5], but that size is not available. Loading the first available size.	
2864	The control [3] on dialog [2] received a browse event, but there is no configurable directory for the present selection. Likely cause: browse button is not authored correctly.	
2865	Control [3] on billboard [2] extends beyond the boundaries of the billboard [4] by [5] pixels.	
2866	The dialog [2] is not allowed to return the argument [3].	
2867	The error dialog property is not set.	
2868	The error dialog [2] does not have the error style bit set.	
2869	The dialog [2] has the error style bit set, but is not an error dialog.	
2870	The help string [4] for control [3] on dialog [2]	

	does not contain the separator character.	
2871	The [2] table is out of date: [3].	
2872	The argument of the CheckPath control event on dialog [2] is invalid.	Where "CheckPath" can be the <a href="#">CheckTargetPath</a> , <a href="#">SetTargetPath</a> or the <a href="#">CheckExistingTargetPath</a> control events.
2873	On the dialog [2] the control [3] has an invalid string length limit: [4].	
2874	Changing the text font to [2] failed.	
2875	Changing the text color to [2] failed.	
2876	The control [3] on dialog [2] had to truncate the string: [4].	
2877	The binary data [2] was not found	
2878	On the dialog [2] the control [3] has a possible value: [4]. This is an invalid or duplicate value.	
2879	The control [3] on dialog [2] cannot parse the mask string: [4].	
2880	Do not perform the remaining control events.	

2881	CMsiHandler initialization failed.	
2882	Dialog window class registration failed.	
2883	CreateNewDialog failed for the dialog [2].	
2884	Failed to create a window for the dialog [2].	
2885	Failed to create the control [3] on the dialog [2].	
2886	Creating the [2] table failed.	
2887	Creating a cursor to the [2] table failed.	
2888	Executing the [2] view failed.	
2889	Creating the window for the control [3] on dialog [2] failed.	
2890	The handler failed in creating an initialized dialog.	
2891	Failed to destroy window for dialog [2].	
2892	[2] is an integer only control, [3] is not a valid integer value.	
2893	The control [3] on dialog [2]	

	can accept property values that are at most [5] characters long. The value [4] exceeds this limit, and has been truncated.	
2894	Loading RICHED20.DLL failed. GetLastError() returned: [2].	
2895	Freeing RICHED20.DLL failed. GetLastError() returned: [2].	
2896	Executing action [2] failed.	
2897	Failed to create any [2] font on this system.	
2898	For [2] textstyle, the system created a '[3]' font, in [4] character set.	
2899	Failed to create [2] textstyle. GetLastError() returned: [3].	
2901	Invalid parameter to operation [2]: Parameter [3].	
2902	Operation [2] called out of sequence.	May indicate that the <a href="#">installation of Win32 assemblies</a> was authored incorrectly. A Win32 side-by-side component may need a key path.
2903	The file [2] is missing.	
2904	Could not BindImage file [2].	

2905	Could not read record from script file [2].	
2906	Missing header in script file [2].	
2907	Could not create secure security descriptor. Error: [2].	
2908	Could not register component [2].	
2909	Could not unregister component [2].	
2910	Could not determine user's security ID.	
2911	Could not remove the folder [2].	
2912	Could not schedule file [2] for removal on restart.	
2919	No cabinet specified for compressed file: [2].	
2920	Source directory not specified for file [2].	
2924	Script [2] version unsupported. Script version: [3], minimum version: [4], maximum version: [5].	
2927	ShellFolder id [2] is invalid.	
2928	Exceeded maximum number of sources. Skipping source	

	'[2]'. [2].	
2929	Could not determine publishing root. Error: [2].	
2932	Could not create file [2] from script data. Error: [3].	
2933	Could not initialize rollback script [2].	
2934	Could not secure transform [2]. Error [3].	
2935	Could not unsecure transform [2]. Error [3].	
2936	Could not find transform [2].	
2937	Windows Installer cannot install a system file protection catalog. Catalog: [2], Error: [3].	Windows Installer protects critical system files. For more information, see <a href="#">Using Windows Installer and Windows Resource Protection</a> . For Windows Me, see the <a href="#">InstallSFPCatalogFile action</a> , the <a href="#">FileSFPCatalog table</a> , and the <a href="#">SFPCatalog table</a> .
2938	Windows Installer cannot retrieve a system file protection catalog from the cache. Catalog: [2], Error: [3].	Windows Installer protects critical system files. For more information, see <a href="#">Using Windows Installer and Windows Resource Protection</a> . For Windows Me, see the <a href="#">InstallSFPCatalogFile action</a> , the <a href="#">FileSFPCatalog table</a> , and the <a href="#">SFPCatalog table</a> .
2939	Windows Installer cannot delete a system file	Windows Installer protects critical system files. For more information,

	protection catalog from the cache. Catalog: [2], Error: [3].	see <a href="#">Using Windows Installer and Windows Resource Protection</a> . For Windows Me, see the <a href="#">InstallSFPCatalogFile</a> action, the <a href="#">FileSFPCatalog</a> table, and the <a href="#">SFPCatalog</a> table.
2940	Directory Manager not supplied for source resolution.	
2941	Unable to compute the CRC for file [2].	
2942	BindImage action has not been executed on [2] file.	
2943	This version of Windows does not support deploying 64-bit packages. The script [2] is for a 64-bit package.	
2944	GetProductAssignmentType failed.	
2945	Installation of ComPlus App [2] failed with error [3].	
3001	The patches in this list contain incorrect sequencing information: [2][3][4][5][6][7][8][9][10][11][12][13][14][15][16].	Available beginning with Windows Installer version 3.0
3002	Patch [2] contains invalid sequencing information.	Available beginning with Windows Installer version 3.0

Community content may be also be available for some Windows Installer error messages. If you are viewing the documentation using the online MSDN library, the Community content tool may be displayed at the bottom of this page.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Error Codes

These error codes are returned by the Windows Installer functions MsiExec.exe and InstMsi.exe. Note that any error in Winerror.h (such as ERROR\_INVALID\_DATA, included here) can be returned as well.

**Note** The error codes ERROR\_SUCCESS, ERROR\_SUCCESS\_REBOOT\_INITIATED, and ERROR\_SUCCESS\_REBOOT\_REQUIRED are indicative of success. If ERROR\_SUCCESS\_REBOOT\_REQUIRED is returned, the installation completed successfully but a reboot is required to complete the installation operation.

See [Windows Installer Error Messages](#) for errors returned by the Windows Installer.

**Note** If you are a user experiencing difficulty with your computer either during or after installing or uninstalling an application, you should contact customer support for the software you are trying to install or remove. If you feel you are in need of support for a Microsoft product, please go to our technical support site at [support.microsoft.com](http://support.microsoft.com).

Error code	Value	Description
ERROR_SUCCESS	0	The action
ERROR_INVALID_DATA	13	The data i
ERROR_INVALID_PARAMETER	87	One of the
ERROR_CALL_NOT_IMPLEMENTED	120	This value action atte cannot be The functi ERROR_ Available Installer v
ERROR_APPHELP_BLOCK	1259	If Window may be in operating informing

		try to install returned if installation
ERROR_INSTALL_SERVICE_FAILURE	1601	The Windows be accessed personnel Installer s
ERROR_INSTALL_USEREXIT	1602	The user c
ERROR_INSTALL_FAILURE	1603	A fatal err
ERROR_INSTALL_SUSPEND	1604	Installatio
ERROR_UNKNOWN_PRODUCT	1605	This action are curren
ERROR_UNKNOWN_FEATURE	1606	The featur
ERROR_UNKNOWN_COMPONENT	1607	The comp registered.
ERROR_UNKNOWN_PROPERTY	1608	This is an
ERROR_INVALID_HANDLE_STATE	1609	The handl
ERROR_BAD_CONFIGURATION	1610	The config corrupt. C
ERROR_INDEX_ABSENT	1611	The comp
ERROR_INSTALL_SOURCE_ABSENT	1612	The instal not availal and that y
ERROR_INSTALL_PACKAGE_VERSION	1613	This instal installed b service. Y service pa of the Wir
ERROR_PRODUCT_UNINSTALLED	1614	The produ
ERROR_BAD_QUERY_SYNTAX	1615	The SQL c unsupport

ERROR_INVALID_FIELD	1616	The record
ERROR_INSTALL_ALREADY_RUNNING	1618	Another in progress. ( before pro For inform <a href="#">_MSIExec</a>
ERROR_INSTALL_PACKAGE_OPEN_FAILED	1619	This instal opened. V is accessit vendor to Windows
ERROR_INSTALL_PACKAGE_INVALID	1620	This instal opened. C verify that Installer p
ERROR_INSTALL_UI_FAILURE	1621	There was Installer s your supp
ERROR_INSTALL_LOG_FAILURE	1622	There was log file. V location e
ERROR_INSTALL_LANGUAGE_UNSUPPORTED	1623	This langu is not supp
ERROR_INSTALL_TRANSFORM_FAILURE	1624	There was Verify that are valid.
ERROR_INSTALL_PACKAGE_REJECTED	1625	This instal policy. Co
ERROR_FUNCTION_NOT_CALLED	1626	The functi
ERROR_FUNCTION_FAILED	1627	The functi
ERROR_INVALID_TABLE	1628	An invalid specified.

ERROR_DATATYPE_MISMATCH	1629	The data s
ERROR_UNSUPPORTED_TYPE	1630	Data of th
ERROR_CREATE_FAILED	1631	The Wind start. Cont
ERROR_INSTALL_TEMP_UNWRITABLE	1632	The Temp inaccessib exists and
ERROR_INSTALL_PLATFORM_UNSUPPORTED	1633	This instal on this pla vendor.
ERROR_INSTALL_NOTUSED	1634	Componer
ERROR_PATCH_PACKAGE_OPEN_FAILED	1635	This patch Verify that accessible vendor to Windows
ERROR_PATCH_PACKAGE_INVALID	1636	This patch Contact th that this is patch pack
ERROR_PATCH_PACKAGE_UNSUPPORTED	1637	This patch by the Win must insta contains a Installer s
ERROR_PRODUCT_VERSION	1638	Another v installed. I cannot cor the existin <b>Add/Rem Panel.</b>
ERROR_INVALID_COMMAND_LINE	1639	Invalid co the Windo command-

ERROR_INSTALL_REMOTE_DISALLOWED	1640	The current performance of a server role service
ERROR_SUCCESS_REBOOT_INITIATED	1641	The installation message is
ERROR_PATCH_TARGET_NOT_FOUND	1642	The installation patch became upgraded to patch update program. The upgraded program that you have
ERROR_PATCH_PACKAGE_REJECTED	1643	The patch system policy
ERROR_INSTALL_TRANSFORM_REJECTED	1644	One or more operations are not permitted
ERROR_INSTALL_REMOTE_PROHIBITED	1645	Windows installation Connection
ERROR_PATCH_REMOVAL_UNSUPPORTED	1646	The patch removal patch pack Windows
ERROR_UNKNOWN_PATCH	1647	The patch Available Installer v
ERROR_PATCH_NO_SEQUENCE	1648	No valid set of patches Windows
ERROR_PATCH_REMOVAL_DISALLOWED	1649	Patch removal Available Installer v
ERROR_INVALID_PATCH_XML	1650	The XML beginning

		3.0.
ERROR_PATCH_MANAGED_ADVERTISED_PRODUCT	1651	Administr for a per-u applicatio Available Installer v
ERROR_INSTALL_SERVICE_SAFEBOOT	1652	Windows the compu Mode and Restore to previous s Windows
ERROR_ROLLBACK_DISABLED	1653	Could not transactio disabled. I cannot rur Available Installer v
ERROR_SUCCESS_REBOOT_REQUIRED	3010	A restart i: install. Th success. T where the

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Glossary

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [I](#) [M](#) [P](#) [Q](#) [R](#) [S](#) [T](#) [U](#) [V](#)

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# A

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

## **accessibility**

Design implementation for making the installer's user interface accessible to all users. For more information about accessibility, see the overview topic: [Accessibility](#).

## **acquisition phase**

Phase of installation during which the installer determines procedure. Acquisition phase begins when an application or user instructs *Windows Installer* to install an application or feature. The installer then queries the *database* for information as it generates the *execution script* for the installation. For more information about the phases of an installation, see [Installation Mechanism](#).

## **action**

Many of the functions performed by Windows Installer are encapsulated into actions. Each action specifies the execution of a particular function and the total procedural flow of the installation is prescribed by the sequence of actions in the *Sequence tables*. *Standard actions* are built into Windows Installer. *Custom actions* are written by the author of the installation *package*.

## **Admin Approval Mode**

The approval state enabled by User Account Protection (UAC) that runs all users with least privilege, including administrators. Users are required to provide consent to elevate installations that require administrator privileges.

## **advertising**

Capability to make the interfaces required for loading and to make an application available without installing the application. When a user or application activates an advertised interface, the installer then proceeds to install the necessary components. The two types of advertising are *assigning* and *publishing*. For more information, see also *install-on-demand*. For more information about how the installer advertises applications, see [Advertisement](#).

## **Application Information Service (AIS)**

A system service of Windows Vista that facilitates starting installations that require elevated privileges to run. Provides the Consent UI used by User Account Control to prompt a user for administrator authorization.

### **assigning**

Makes an application available, and makes it appear as if it has been installed to a user, without actually installing it. Assigning adds shortcuts and icons to the **Start** menu, associates appropriate files, and writes registry entries for the application. When a user tries to open an assigned application, then the installer installs the application. Assigning and publishing are two methods of *advertising*. For more information, see [Advertisement](#).

### **asynchronous execution**

Custom action during which the installer runs separate threads of the custom action and the current installation simultaneously. For more information, see [Synchronous and Asynchronous Custom Actions](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# B

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [K](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#) [Z](#)

## **basic UI**

One level of the installer's *internal user interface* capabilities. The basic user interface (UI) level supports a UI for the installation that has simple progress and error handling. For more information about user interface levels, see [User Interface Levels](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# C

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

## **cabinet file**

Single file, usually with a .cab extension, that stores compressed files in a file library. The cabinet format is an efficient way to package multiple files because compression is performed across file boundaries, significantly improving compression ratio. For information about creating cabinets, see [Cabinet Files](#).

## **checksum**

Computed value that depends on the contents of a file. It is stored with data to detect file corruption. The system checks this value to ensure that the data is uncorrupted.

## **component**

Smallest piece of an installation handled by the installer, also a building-block of an application or feature from the programmer's perspective. Examples of components are a group of related files, a COM object, or a library. For more information, see [Components and Features](#).

## **committing databases**

Accumulated changes made in a database. The changes are not reflected in the actual database until the database is committed. For more information, see [Committing Databases](#).

## **ControlEvent**

Aspect of functionality of the installer's user interface. A typical ControlEvent triggers some action by the installer upon the activation of a dialog box control or other event. For more information, see [ControlEvent Overview](#).

## **costing**

Method used by the installer to determine disk space requirements for installation. Costing takes into account factors such as whether existing files need to be overwritten. For more information, see [File Costing](#).

## **.cub file**

Validation module that stores and provides access to ICE custom

actions. For more information, see [Sample .cab File](#). For more information, see also [Windows Installer File Extensions](#).

### **custom action**

Written by the author of the *package* but not built into the installer as a standard action. For more information, see [Custom Actions](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# D

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

## **database function**

Accesses the installer database. These functions are provided primarily for use by *installer package authoring tools* and they are not intended to be used to call installer services. For more information, see [Database Functions](#) and the [Installer Function Reference](#).

## **database handle**

Required for working with a database. For more information, see [Obtaining a Database Handle](#).

## **delta patch**

A delta patch is a delta-compressed Windows Installer patch created using a tool, such as Patchwiz.dll, that supports delta compression. Patches that use delta compression can reduce the size of an update by providing only the differences (deltas) between existing files on a target computer and the desired new files. The desired new files are then synthesized from the existing files and the downloaded deltas. For more information about delta compression technology, see the [Delta Compression Application Programming Interface](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# E

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

## **elevated**

Actions performed with system privileges are called elevated.

## **execution phase**

When the installer executes a script of installer actions. In Microsoft Windows 2000, this process is performed by the installer service. In a managed application, the script is executed with system privileges. For more information, see [Installation Mechanism](#).

## **execution script**

Installer actions for an installation. The execution script is generated during the [acquisition phase](#) of installation and executed during the [execution phase](#). For more information, see [Installation Mechanism](#).

## **external source files**

Source files of the application being installed that are stored outside of the [.msi file](#). Multiple external source files can be compressed and stored together inside of a [cabinet file](#) included in the [package](#).

## **external user interface**

UI provided by the author of an installation package. It does not use the internal UI capabilities of the installer. For more information, see [About the User Interface](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# F

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

## **feature**

Functionality from the user's perspective. For more information, see [Components and Features](#).

## **full UI**

Characteristic of the installer's *internal user interface* capabilities. The full UI level supports a user interface for the installation, including dialog boxes, progress bars, and error messages. For more information, see [User Interface Levels](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# G

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [K](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#) [Z](#)

## **globally unique identifier (GUID)**

Data type representing a class identifier. For more information, see [GUID](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# I

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

## **.idt file**

Exported installer database table. For more information, see [Importing and Exporting](#) and [Windows Installer File Extensions](#).

## **Import Address table (IAT)**

Where the computed virtual address is saved from each function imported from all DLLs.

## **internal user interface**

Built-in capabilities of the installer that can be used to author a graphical UI for installation. An author may choose an [external user interface](#). For more information, see [About the User Interface](#).

## **install level**

Specified installation value. An application is installed only if its own level is less than or equal to the install level. The set of applications chosen for installation can therefore be changed by altering the install level. For more information, see [Feature Table](#).

## **installation-on-demand**

Service that installs applications or features as requested by the user or another application. [Advertising](#) makes a feature or application available for install-on-demand installation. For more information, see: [Installation-on-Demand](#).

## **installation context**

The combination of installation rights and installation types produces three different installation contexts: per-user non-managed, per-user managed, and per-machine managed. There is no such thing as per-machine non-managed.

## **installer**

The [Windows Installer](#).

## **installer database**

Relational database containing setup instructions for an installation. The installer database is stored in the [.msi file](#). For more information, see [Installer Database](#).

## **installer function**

Called by a user or application to obtain installer services and functionality. This is the application programming interface of the installer. For information, see [Installer Function Reference](#).

## **installer package authoring tool**

Software that enables a developer to create and edit an .msi file. This together with any [external source files](#) comprise a [package](#) containing all the information required for an installation.

## **internal source files**

Stored in the .msi file. Multiple internal source files can be compressed and stored together in a [cabinet file](#) that is stored within an .msi file.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# M

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

## managed application

System privileges, also known as elevated privileges, are used to install the application. An application is managed on a system if it is a per-machine installation on Windows 2000, or if the application is assigned or published using Group Policy on Windows 2000.

## merge

Combines two installation databases to form a single database. For more information, see [Merging Databases](#).

## merge module

Database merged with an installation package to deliver new components to an application. Merge modules cannot be installed alone. For more information, see [Merge Modules](#).

## Microsoft Windows Installer

Client-side installer service for managing the installation of applications on 32-bit platforms. The application must be encapsulated in a *package*. For more information, see [About Windows Installer](#).

## .msi file

COM-structured storage file containing the instructions and data required to install an application. Every package contains at least one .msi file. The .msi file contains the *installer database*, a *summary information stream*, and possibly one or more *transforms* and *internal source files*. For more information, see [Windows Installer File Extensions](#).

## .msm file

[Merge module](#). For more information, see [Windows Installer File Extensions](#).

## .msp file

[Patch package](#). For more information, see [Windows Installer File Extensions](#).

## .mst file

Transform package. For more information, see [Merges and Transforms](#). For more information, see [Windows Installer File Extensions](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# O

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

## **Over The Shoulder**

User experience of standard user in Windows Vista that attempts to install an application that requires administrator privileges. The Application Information Service prompts for consent from an administrator before elevating the installation.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# P

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

## package

*.msi file* and any *external source files* that may be pointed to by this file. A package therefore contains all the information the Windows Installer needs to run the UI and to install or uninstall the application. For more information, see also *installer database*.

## package code

GUID that identifies a particular package. A unique package code is required because some transforms or patch packages can be applied to more than one application or can upgrade an application into another application or version. For more information, see [Package Codes](#).

## patching

Method of updating a file that replaces only the bits being changed, rather than the entire file. This means that users can download an upgrade patch for a product that is much smaller than the entire product. For more information, see [Patch Packages](#).

## patch file

[Patch package](#) used for patching. For more information, see [Patching and Upgrades](#).

## preview mode

Mode for viewing the design of the UI. For more information, see [Previewing the User Interface](#).

## progress bar

Control the installer can display during script execution time representing the progress of the installation. For more information, see [Authoring a ProgressBar Control](#).

## property

Global variables used during an installation. (See [About Properties](#).)

The term "property" also refers to an attribute of an automation object. (See [Automation Interface](#).)

## publishing

Method of [advertising](#) a feature or application. Publishing does not populate the UI. However, if another application attempts to open a published application, there is enough information present for the installer to assign the published application. For more information, see [assigning](#) and [Components and Features](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# Q

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

## **qualified component**

Method of single-level indirection that is similar to a pointer. Qualified components are primarily used to group components with parallel functionality into categories. For more information, see [Qualified Components](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# R

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

## **reduced UI**

Level of the installer's *internal user interface* capabilities. Displays only authored modeless dialog boxes. For more information, see [User Interface Levels](#).

## **reinstall**

Partial or complete reinstallation of an installed application. Commonly done to repair an application if any files or registry entries have become corrupted or are missing. For more information, see [Reinstalling a Feature or Application](#).

## **resiliency**

Application capability of graceful recovery from error without generating error messages. For more information, see [Resiliency](#).

## **rollback**

Automatic restoration of the original state of the computer should the installation fail. For more information, see [Rollback Installation \(Windows Installer\)](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# S

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

## **sequence tables**

Tables in the *installer database* that list actions sequentially. For more information, see [Using a Sequence Table](#).

## **source list**

Specifies the locations where the installer is to search for *external source files*.

## **standard action**

Built-in *action* of the Windows Installer. For more information, see [Standard Actions](#).

## **standard user**

User without administrator privileges.

## **Structured Query Language (SQL)**

Language used to process data in a relational database. You can use SQL to make queries to the installer database. For more information, see [Working with Queries](#).

## **summary information stream**

Storage location in the *.msi file* for information that can be viewed with Microsoft Windows Explorer. For more information, see [About the Summary Information Stream](#).

## **synchronous execution**

When the installer waits for the separate thread of a custom action to complete before continuing the thread of the current installation. For more information, see [Synchronous and Asynchronous Custom Actions](#)

## **system policy**

Used by system administrators to control user and computer configurations from a single location on a network. System policies propagate registry settings to a large number of computers without requiring the administrator to have detailed knowledge of the registry. The system policy of the installer can be configured using the Group Policy Editor (GPE), included in Windows 2000. For more

information, see [System Policy](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# T

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

## **transaction processing**

One or more operations processed together as a single indivisible whole called a transaction. All the constituent operations must succeed for the transaction to succeed, otherwise all the operations are rolled back to the original state.

## **transform**

Template of the differences between two *installer databases* that can be applied to produce similar changes in other databases. For example, a localization transform can be used to change the language of an application. For more information, see [Merges and Transforms](#).

## **transform error condition flags**

Set of properties used to flag the error conditions of a *transform*. For more information, see [MsiCreateTransformSummaryInfo](#).

## **transform validation flags**

Set of properties used to verify that the transform can be applied to the database. For a list of these properties, see [MsiCreateTransformSummaryInfo](#).

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# U

[A](#) [B](#) [C](#) [D](#) [E](#) [F](#) [G](#) [H](#) [I](#) [J](#) [K](#) [L](#) [M](#) [N](#) [O](#) [P](#) [Q](#) [R](#) [S](#) [T](#) [U](#) [V](#) [W](#) [X](#) [Y](#) [Z](#)

## **upgrade**

Recent and improved version of an existing application. For more information, see [Patching and Upgrades](#).

## **User Account Control (UAC)**

UAC is a new access control technology in Windows Vista that causes all users to run with limited privileges. For more information, see the [User Account Control](#) documentation.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.

# V

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

## **volume**

Storage medium, such as a disk or tape, that is formatted to contain files and directories.

[Send comments about this topic to Microsoft](#)

Build date: 8/13/2009

© 2009 Microsoft Corporation. All rights reserved.