WHITE PAPER

# Building MSI Updates and Patches

by Robert Dickau
Principal Technical Training Writer, Flexera Software

FLEXERA SOFTWARE

# Building MSI Updates and Patches

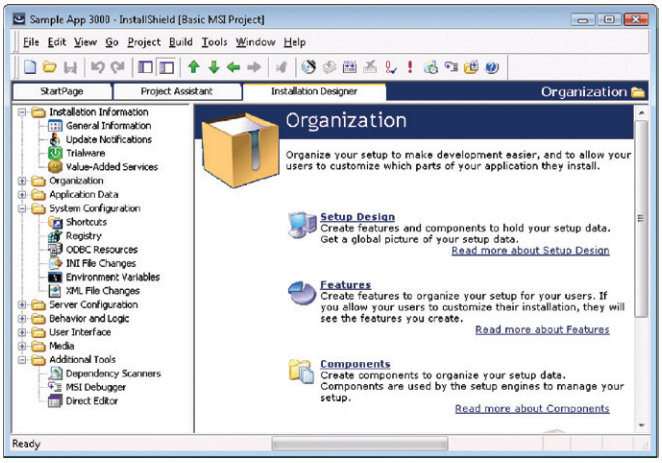
## Introduction

This white paper describes the changes you make to a Windows Installer (MSI) installation package for it to behave as the desired type of update or patch. It begins by discussing information about settings to change in the MSI database. It also highlights the tools in InstallShield® by Flexera™ Software that simplify the process.

Also, if you haven't read the white paper, "*Designing an Update-Friendly MSI Installation*," it has valuable tips on how to build your initial install. You can download it at www.flexerasoftware.com/whitepapers.

## Using the InstallShield Environment

This white paper frequently refers to the InstallShield development environment. It is assumed you are familiar with the general layout of the InstallShield interface, which contains a list of views with which you can modify different portions of your installation project.




For example, the General Information view is where you set general product and project properties; the Setup Design view enables you to edit the features, components, and component data used by your project; the Registry view enables you to modify the registry data installed by your installation program; and the Direct Editor view gives you access to the raw MSI database tables.

It is also assumed you are familiar with some of the wizards available with InstallShield, such as the Release Wizard and Component Wizard.

- **The Release Wizard**, available under the Build menu and also from the Releases view, lets you describe the properties—media type, compression settings, and so forth—of a release, and then builds the specified release image.
- **The Component Wizard**, available by right-clicking a feature in the Setup Design view, lets you create special types of components, such as components for COM servers, fonts, and Windows services.

The InstallShield Help Library contains information about using every view and wizard in the InstallShield environment. The InstallShield Help Library is available when you press F1 with any view selected; you can also select Contents from the Help menu to view the help library.

In addition to the graphical environment, InstallShield provides several tools for modifying and building projects from the command line or an external script. For example, to build a project from the command line, batch file, or other automated process, you can use the executable IsCmdBld.exe. The IsCmdBld executable is located in the System subdirectory of the InstallShield distribution directory.

To rebuild a project, you pass IsCmdBld the project file path, the product configuration name, and the release name that you want to rebuild. A sample command appears as follows:

**iscmdbld -p C:\ProductName.ism -a BuildConfig -r ReleaseName**

In addition, InstallShield provides an Automation interface, with which you can modify the contents of a project file without using the graphical environment.

**Learn More about InstallShield:**
If you wish to learn more about the capabilities of InstallShield, please visit the Flexera Software Web site at http://www.flexerasoftware.com/installshield

## Review of Upgrade Types

Windows Installer supports three types of product upgrades: small updates, minor upgrades, and major upgrades. The three types of upgrades are defined as follows.

- **A small update** consists of product changes, such as hot fixes, so small that no change to the product version is necessary or desired. (A drawback to small updates is that external programs, including installers for later versions of your product, will not be able to distinguish a product with the small update applied from one without the small update.)
- **A minor upgrade** is a change to the product large enough to merit a change to the product version, such as updating version 1.1 to 1.2, but in which there have been no significant changes to the setup organization between versions. The install-time behavior of a minor upgrade is to install over the existing product.
- **A major upgrade** includes substantial product changes, such as updating version 1.2 to 2.0. A major upgrade can contain significant changes to the setup architecture; later. The install-time behavior of a major upgrade can be to uninstall the earlier version and install the new one, or to install over the earlier version and then remove any leftover data.

*NOTE: For an earlier product version that was installed with a legacy (non–Windows Installer) setup, a custom action will normally be required to uninstall or modify the existing product installation.*

## MSI Codes and Updates

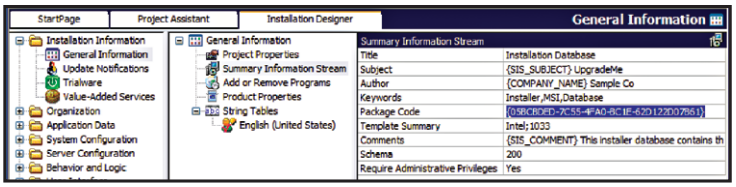
Every MSI database contains a handful of codes that identify the product being installed.

- **Package Code**: part of the Summary Information Stream, the package code identifies a particular database. Any two MSI databases with identical package codes must have identical contents, and therefore you should change the package code for each build.
- **ProductVersion**: MSI property storing the product version. Note that MSI uses only the first three fields of the ProductVersion property for version comparisons: in *a.b.c.d*, the field *d* is ignored. (Note that this is true just for comparisons of ProductVersion values, and not for file versions. File-version comparisons can use all four fields of a file's version.)
- **ProductCode**: MSI property containing the GUID for the current product. MSI treats two products with different ProductCode GUIDs as unrelated, even if ProductName is the same. (As you will see, the major-upgrade process instructs MSI to treat two products with different GUIDs as related.)
- **UpgradeCode**: MSI property containing a GUID representing the current product "family"; in the major-upgrade process, if two products have different ProductCode values but the same UpgradeCode value, MSI knows that the products are related, and are therefore candidates for the major-upgrade process. In general, the upgrade code value never changes.

When you design different types of upgrades, you must change different combinations of these codes. The following table summarizes the required changes.

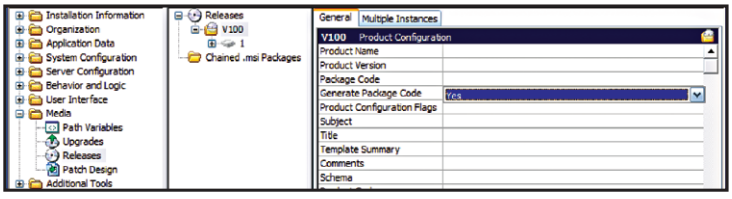
| | Package code (SIS) | Product Version | Product Code | Upgrade Code |
|---|---|---|---|---|
| Small update | X | | | |
| Minor upgrade | X | X | | |
| Major upgrade | X | X (usually) | X | |

Inside InstallShield, the package code is exposed in the Summary Information Stream view, under General Information.



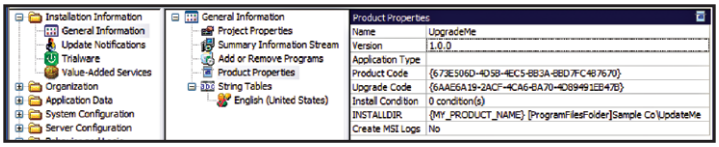

In this view, you can select the Package Code setting and click the Generate GUID button (not pictured).

Because the package code should change for every new release, the default behavior of InstallShield is to generate a new package code each time you perform a build. You can modify this behavior by selecting a product configuration icon in the Releases view, and setting Generate Package Code to No, though doing so is strongly discouraged.




Note that you can also use this view to specify a product version, product code, and other properties for the current product-build configuration. If you enter a specific value here, it automatically overrides the value in the Product Properties view.

The other settings—Version, Product Code, and Upgrade Code—are exposed in the Product Properties view, also under General Information.




To change the Version setting, simply type in the updated value. To change Product Code (for a major upgrade), you can select the property and then click the Generate GUID button (not pictured). These settings are also available in the Property Manager, in the Behavior and Logic view group.

**TIP:** An advanced technique is to modify a project's version and GUIDs using the InstallShield Automation interface, which provides a GenerateGUID method. For more information, see the InstallShield Help Library.

## Creating Update Packages

This section describes the database changes required for different types of updates, and describes how to use the tools provided in InstallShield to simplify the update-development process.

### Minor Upgrades

As listed in the table above, in addition to the new files and other data in your update package, to create a minor upgrade you must change the package code and product version of your installer.

*NOTE: By default, when you run an installation from the InstallShield environment by clicking the Run toolbar button, InstallShield will quietly uninstall any previous version of your product present on the development machine. To change this setting so that deploying an installation from the environment will use the default MSI behavior, pull down the Tools menu, select Options, select the Preferences tab, and clear the check box labeled "Uninstall before installing".*

To deploy the minor upgrade, a typical command line is the following:

**msiexec /i product.msi REINSTALLMODE=voums REINSTALL=ALL**

If the update contains features that you do not want to update, you should set REINSTALL to a comma-separated list of the features that you do want to update, as in the following command (names you use in the REINSTALL property are case-sensitive):

**msiexec /i product.msi REINSTALLMODE=voums REINSTALL=F1,F3,F5**

The important setting for deploying a minor upgrade is the REINSTALLMODE flag "v", which indicates to run the installer using the updated MSI database and to re-cache the package based on the new database. Without the "v" flag, Windows Installer runs the installer based on the cached version of the existing earlier product version.

*NOTE: You must take care not to set REINSTALL=ALL for a first-time installation. Later you will see how InstallShield can create a setup launcher that sets REINSTALL dependent on whether an earlier version of the product is present on the target system.*

### Major Upgrades

For a major upgrade, as indicated in the earlier table, you must change the package code (in the Summary Information Stream); you will usually change the product version (in the ProductVersion property in the Property table); and now must also change the product code (the ProductCode property in the Property table).

When you change the product code, Windows Installer treats your latest and previous product versions as unrelated, even though the ProductName values are likely the same. If you want both versions of your product to be installable on the same system, you can simply change the product code and the main installation directory (often INSTALLDIR). (Naturally, the application must have been developed to support multiple simultaneous instances by separating each instance's configuration settings and data files.)

If instead you want your latest product version to supersede an earlier installed version, you will populate records in the Upgrade table. A record in the Upgrade table contains the following fields:

- The UpgradeCode value of the product(s) you want to update.
- A range of versions of products to update.
- Optional language information for the product to update.
- An optional list of features to remove.
- A public property (called an "action property") associated with any products to update.

A sample record in the Upgrade table might appear as follows.

| UpgradeCode | {11111111-2222-3333-4444-555555555555} |
|---|---|
| VersionMin | 1.0.0 |
| VersionMax | 2.0.0 |
| Language | |
| Attributes | 1025 |
| Remove | |
| ActionProperty | OLDPRODUCTS |

The bit flags used in the Attributes field are described in the MSI Help Library page "Upgrade Table". The value 1025 used here means that all languages should be detected (1024) and that feature states should be migrated (1) from the installed product to the latest product.

In addition to adding records to the Upgrade table, some further authoring steps required are to:

- Add the action property name to the value of the SecureCustomProperties property.
- Ensure the value of ALLUSERS is the same in the before and after versions, through a custom action or from the command line.

**TIP:** Because the packages involved in a major upgrade have different product codes, you cannot use the MSI property "Installed" to determine if the earlier version of your product is present on the target system. Instead, the older product version can determine if it is being removed because of a major upgrade using the UPGRADINGPRODUCTCODE property; the newer version can use the action property (say, OLDPRODUCTS) that you defined in the Upgrade table.

At run time, the standard MSI action FindRelatedProducts reads the records in the Upgrade table, and, if a related product is found, the product code for that product is added to the action property. FindRelatedProducts is by default placed early in both the User Interface and Execute sequences.

The MigrateFeatureStates action (if appropriate) attempts to migrate the feature-selection states from an installed product version to the newer version. Finally, the standard RemoveExistingProducts action reads the product codes stored in the Upgrade table action property, and in effect performs silent nested-uninstallation actions on those products.

The RemoveExistingProducts action can be placed in different locations to define different types of upgrade behavior. In short, placing RemoveExistingProducts early in the Execute sequence instructs Windows Installer to completely remove the existing product data before installing the new product. Placing RemoveExistingProducts late in the Execute sequence causes the upgrade to install the new product data, and then remove old data.

**TIP:** The Attributes field of the Upgrade table also supports a detect-only bit (value 2). If an Upgrade record has the detect-only bit set, the action property defined in that record will be populated with the product codes of corresponding product versions, but the RemoveExistingProducts action will not remove those products. The detect-only bit is useful, for example, when creating a custom action that prevents an older version of the product from installing over a newer version. Recent InstallShield versions automatically create such an Upgrade-table record and custom action for new Basic MSI projects. For more information about creating such a custom action, see the MSI Help Library page "How do I prevent an old package from installing over a newer version?"

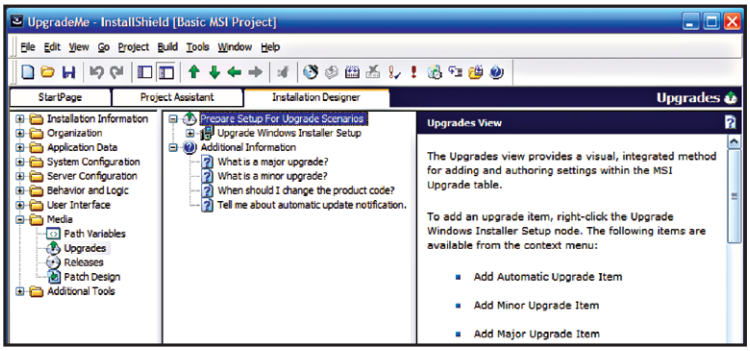
**TIP:** If you use the same UpgradeCode value for every release of your product, an external program or custom action can determine the product codes of any installed copies of your product using the MSI API MsiEnumRelatedProducts, or the RelatedProducts property of the MSI Automation interface.

When you define a major upgrade, it can be useful to run MSI validation on your upgrade package. The rules ICE61, ICE63, and ICE74 can detect package-authoring errors related to major upgrade packages. By default, InstallShield performs upgrade validation during the build process to flag common authoring problems with the different types of upgrades.

To deploy a major upgrade package, the user can simply launch the MSI database or setup launcher, without needing to set any special values for REINSTALLMODE or REINSTALL.

## InstallShield Upgrades View

To simplify the process of creating minor and major upgrades, InstallShield provides the Upgrades view, in which you specify your latest product version and the earlier packages you want it to update.

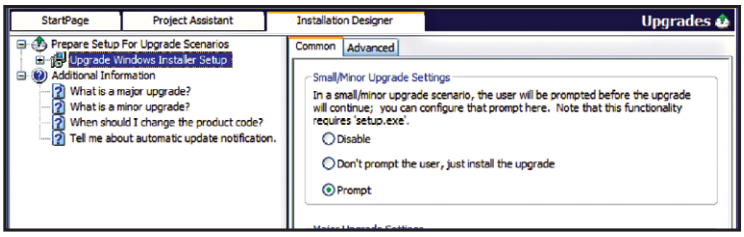



In the Upgrades view, you can define a "minor upgrade item" or a "major upgrade item". (There is also an "automatic upgrade item", which determines which type of upgrade to create based on the earlier and later packages you specify. Automatic upgrade items will not be discussed here.) The two types of upgrade items are described in the following sections.

For more information about the different types of upgrades, you can select the appropriate help icon under the Additional Information icon, pictured in the figure above.

### Creating Minor Upgrades with InstallShield

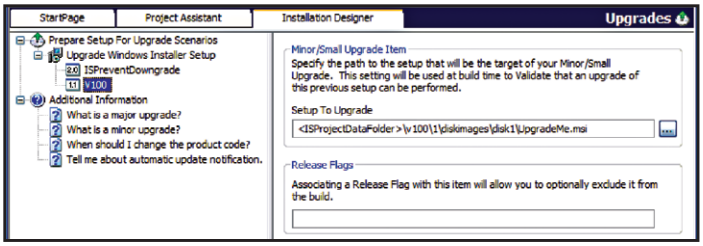
When you create a minor upgrade item, a general setting you can modify is how a setup launcher should behave if an earlier product version is present. You can modify this setting by selecting the Upgrade Windows Installer Setup icon and reviewing the Small/Minor Upgrade Settings section of the IDE, as pictured in the following figure.



As you have seen, a minor upgrade package requires the REINSTALLMODE and REINSTALL properties to be set during deployment. If your build configuration included a Setup.exe setup launcher, you can select one of the following settings:

- **Disable**: this setting causes Setup.exe not to set REINSTALLMODE and REINSTALL if the launcher discovers an earlier version of the product on the target system. With this setting, you must manually set REINSTALLMODE and REINSTALL through other means.
- **Don't prompt the user, just install the upgrade**: this setting causes Setup.exe to set REINSTALLMODE to "voums" and REINSTALL to "ALL" if an earlier product version is detected, and otherwise just behaves as a first-time installation.
- **Prompt**: this setting (the default) causes Setup.exe to display a message to the user if an earlier version exists, prompting whether to update the setup. If the user agrees, Setup.exe sets REINSTALLMODE and REINSTALL as in the previous setting. If no earlier version exists, no prompt is displayed, and the setup behaves as a first-time installation.

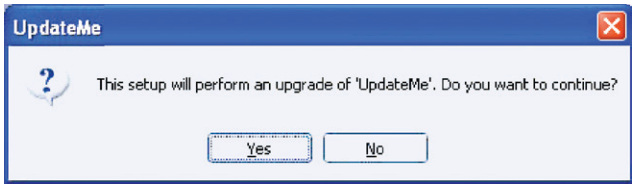
To create a minor upgrade item, right-click the Upgrade Windows Installer Setup icon and select Add Minor Upgrade Item. If desired, you can rename the new icon to represent the earlier version of your product, as in "Version100". In the "Setup to Upgrade" field, browse for the MSI database for the earlier product version that you want to update.




If you are creating an upgrade that can update more than one previous version, you can right-click the Upgrade Windows Installer Setup icon and create additional minor-upgrade items.

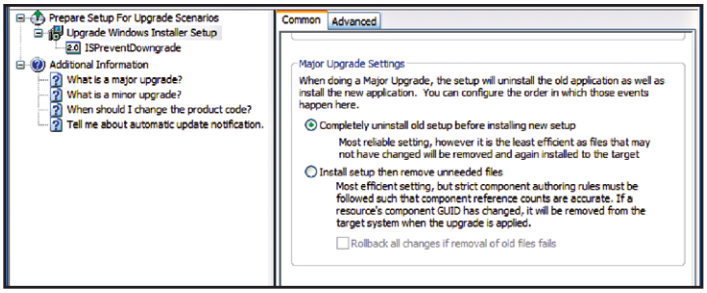
After having created one or more minor-upgrade items, whenever you build a media set, by default InstallShield will perform validation on the latest and earlier product versions, alerting you to possible errors in the upgrade settings.

If you built your updated version to include Setup.exe, and set the minor upgrade settings to prompt the user whether to update a product, the user will see the following prompt, if appropriate.



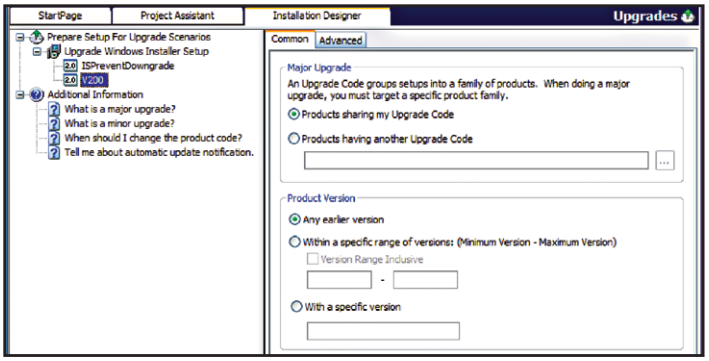

## Creating Major Upgrades with InstallShield

When you create a major upgrade item, a general setting you can modify is where to schedule the RemoveExistingProducts action. You can modify this setting by selecting the Upgrade Windows Installer Setup icon and reviewing the Major Upgrade Settings section of the IDE, as pictured in the following figure.




To create a major-upgrade item, right-click the Upgrade Windows Installer Setup icon and select Add Major Upgrade Item. You can rename the icon to describe the latest version of your setup, as in "V200".

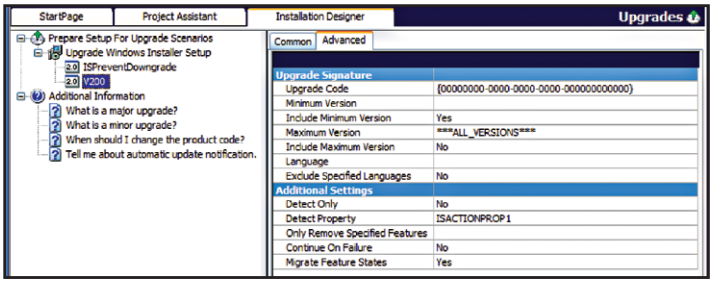
The properties for your major-upgrade item are a more user-friendly view of the Upgrade table settings. In the Common tab for a major-upgrade item (pictured below), you specify the Upgrade Code value of the products you want to update, as well as the versions of product you want to update.




## Designing an Update-Friendly Installation

(The existing item ISPreventDowngrade is provided by InstallShield to prevent an older version of a product from being installed over a newer version.)

In the Advanced tab for a major-upgrade item, you can specify other settings to be written to the Upgrade table, such as the name of the action property (by default "ISACTIONPROP1"), whether to remove only certain features, and so forth.



At build time, InstallShield by default includes an ISSetAllUsers custom action to manage setting ALLUSERS correctly during a major upgrade. (In addition, ISSetAllUsers sets the custom property IS_MAJOR_UPGRADE if a major upgrade is taking place.) To control whether to include the ISSetAllUsers action, pull down the Tools menu, select Options, and activate the General tab. You can clear or select the "Automatically create ISSetAllUsers action" check box.

To deploy a major upgrade, a user can simply launch the MSI file or setup launcher. No special command-line settings are required. If a related earlier product version is detected, it will be removed; if no earlier version is detected, the setup will behave as a first-time installation.

## Creating Patches

A patch is a packaging mechanism for an upgrade. After you have created an upgrade—usually a minor upgrade—as described above, you can build a patch for it. The earlier product databases for which you want to build a patch are commonly called the target databases, and the latest version is commonly called the upgraded package.

### PCP Files

InstallShield uses Patch Creation Property (PCP) files to store settings for a patch to be created. These settings include, among many others:

- A GUID for the patch
- ProductCode values for products the patch will update
- A list of patches that the current patch supersedes
- Locations of uncompressed source and target MSI packages

A PCP file uses the MSI format, and can be edited directly with InstallShield or Orca; the standard tables and fields of a PCP file are described in the MSI Help Library.

The four required tables in a PCP file are the following:

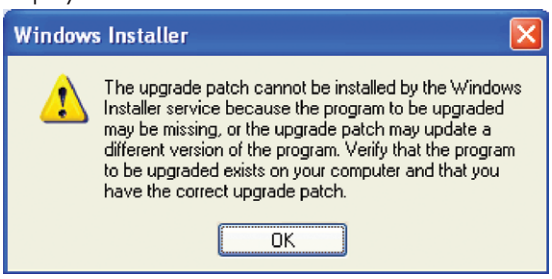
- Properties: contains global settings for the patch, such as the patch GUID, the path to the patch-creation output directory, whether to include only entire files, and so forth.
- ImageFamilies: contains an identifier for a group of related target and updated images.
- TargetImages: describes the target MSI packages involved in the patch.
- UpgradedImages: describes the upgraded packages involved in the patch.

**TIP:** Given a product's ProductCode value, you can determine which patches have been applied to it by using the MSI API function MsiEnumPatches, or the MSI Automation interface property Patches.

You run a patch using a command line similar to the following:

**msiexec /p patch.msp REINSTALLMODE=oums REINSTALL=ALL**

It is not necessary to specify the product to which the patch applies; if the user does not have an appropriate target product version installed, the following error dialog box will be displayed.



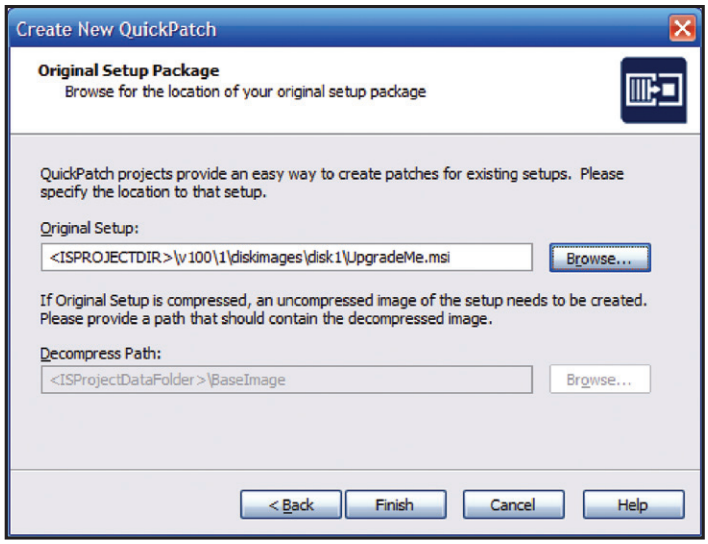

### QuickPatch® Projects

A QuickPatch project enables you to quickly create a patch for an existing MSI package, where the patch contains only file and registry changes. In a QuickPatch project, you can add, modify, or remove files and registry data; if you need the patch to include additional changes, you should use the Patch Design view, described in the following section.

To create a new QuickPatch project in InstallShield, you can pull down the File menu, select New, and then select QuickPatch Project, entering the name of the InstallShield project (ISM) file to create.

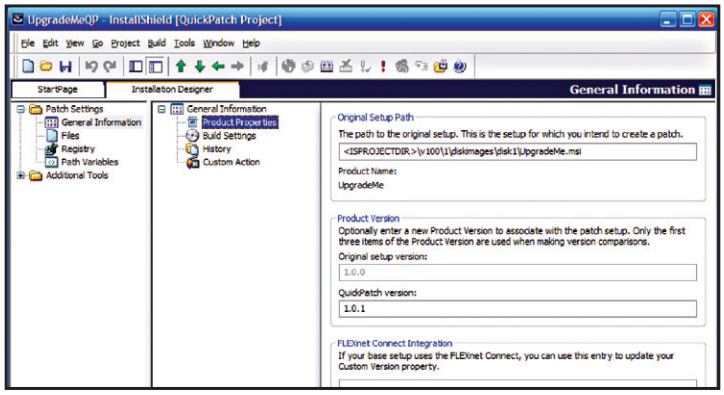
In the QuickPatch Project Base panel, you are prompted to base the patch on an existing MSI package or on an existing QuickPatch project.


You will then be prompted for the original database or project, creating an uncompressed image of your installation package if necessary.




In the IDE for a QuickPatch project, the Patch Settings view group contains views in which you can edit the general information, files, and registry data to be included in the patch.

In the General Information view, the Product Properties section is where you can view and modify the location of the Windows Installer package for which the patch is being generated, the patch version information, and optional Update Service information.

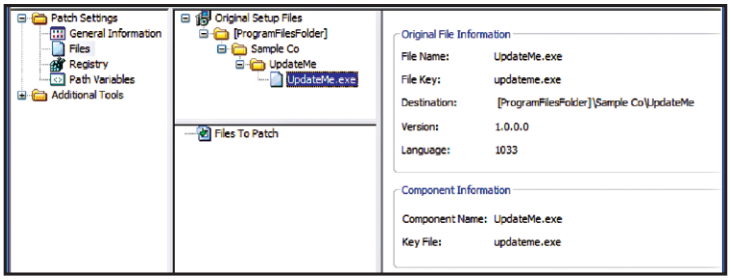



In the Build Settings view (not pictured), you specify such settings as:

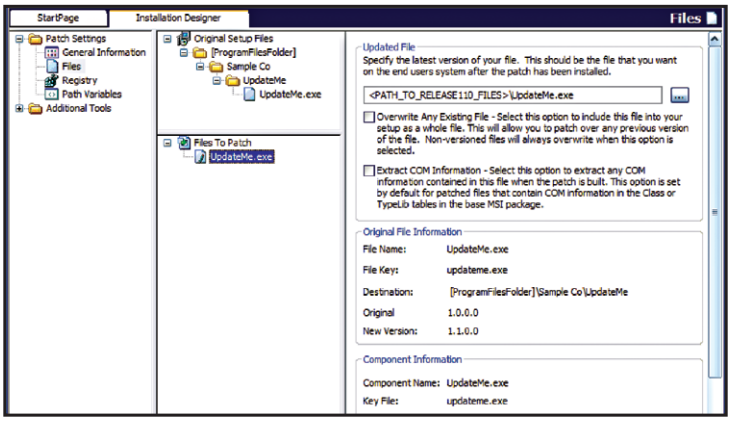
- Where to place the build output (the MSP file, and so forth).
- Whether to build an Update.exe launcher, which launches the patch file with the appropriate REINSTALLMODE and REINSTALL values.
- Whether to include the MSI, InstallScript, or .NET redistributables with your patch launcher.
- Where to obtain any redistributables you selected: either built into Update.exe or downloaded from a URL.

In the History and Custom Actions views (not pictured), you can view and modify the base and intermediate projects that your current patch applies to, and specify if any of your custom actions should be set up not to run while your patch is being applied.

In the Files view, you see a tree of the destination folders used by your project, and the files in them. Selecting a file icon in the Original Setup Files tree displays the file's destination and version information, as pictured in the following figure.




In this view, you can change, add, and remove files. To update a file, drag its icon from the Original Setup Files tree to the Files to Patch tree. When you select the file icon in the Files to Patch tree, in the Update File area you then browse for the newer version of the file.




Note that a file installed by a patch by default obeys the file-overwrite rules. If you want, you can select the Overwrite Any Existing File check box to include the entire updated file in the patch (as opposed to just the byte-level binary differences). This setting causes your file to overwrite any earlier version of the file on the target system in the case of a versioned file, or overwrite any existing file in the case of an unversioned file.

You can also delete the file from the setup using your patch by selecting the "delete this file" check box at the bottom of the file-information view.

To add a new file to the product using your patch, right-click the Files to Patch icon and select Insert New File. You will be prompted to browse for the new file; afterward, you can specify the file's destination and the features with which the new file is associated.

Use of the Registry view (not pictured) follows similarly. In the Registry view, you see a tree of registry data contained in the earlier product version, as well as a tree representing the registry data on the development system. You can then add, remove, and modify registry keys and values relative to the original package, and the patch will include the appropriate changes.

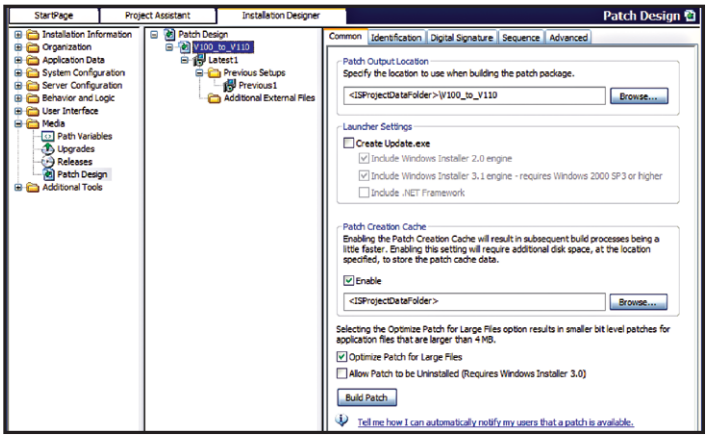
To build the patch, simply click the Build toolbar button. By default, recent versions of InstallShield will perform upgrade and patch validation. When the build is complete, you can open the folder containing the build output by clicking the Open Release Folder toolbar button. The output directory includes the patch (MSP file) or Update.exe, the build log, and so forth.

Using the Patch Design View

The InstallShield Patch Design view provides a more sophisticated environment for creating patches. Unlike QuickPatch projects, which are a separate project type, the Patch Design view is associated with an existing project. In the Patch Design view, it is assumed you have already built the projects for your earlier and later product versions.

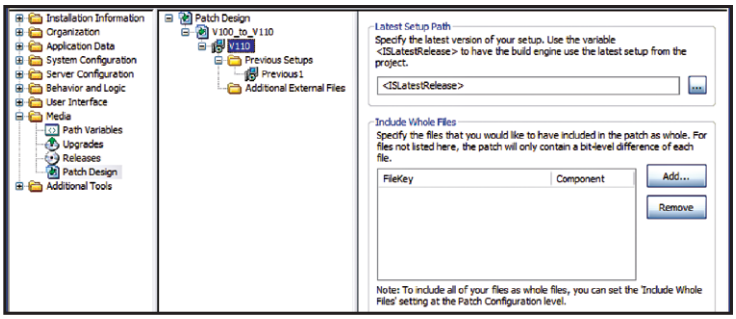
*NOTE: When running the Release Wizard for your new product version, you will want to use patch optimization, especially if any of your components have used dynamic file linking that includes subfolders.*

To create a patch, you begin by right-clicking the Patch Design icon and selecting Add New Patch Configuration. A patch configuration consists of one "Latest Setup" and one or more "Previous Setups". Before you specify the latest and previous setups, you can specify overall patch properties, such as where to store the built patch data, whether to create a patch launcher Update.exe, and so forth.




In the Advanced tab for a patch configuration (not pictured), you can specify additional settings to write to the PCP file, such as a fixed patch GUID, whether to include only entire files, and so forth. In addition, the Identification, Digital Signature, and Sequence tabs enable you to take advantage of uninstallable patches and patch sequences, introduced in recent versions of Windows Installer.

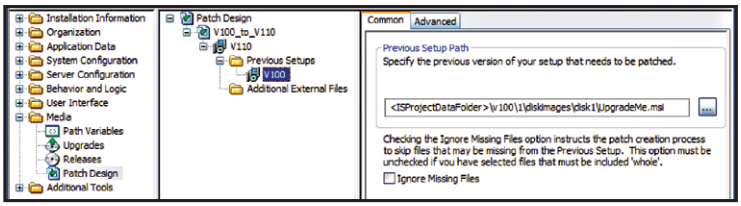
To specify the latest version (the updated image) of your package, right-click the patch configuration icon and select Add New Latest Setup. InstallShield automatically provides one latest setup project, which you can rename if desired.




In the Latest Setup Path field, you can use the variable <ISLatestRelease> for InstallShield always to use the latest build of your setup as the upgraded image. Alternatively, you can browse for a specific latest version's MSI database or Setup.exe launcher. The release you select here should be an uncompressed release, ideally created with an administrative installation if the previous product versions were compressed.

In addition, you can use the Include Whole Files section to specify particular files to include as whole files; the default behavior is to include the byte-level file differences, where possible. Including some files as whole files can suppress certain prompts for the original installation source while the patch is being applied.

Under the latest-setup icon are folders labeled Previous Setups and Additional External Files. In the Previous Setups folder, you add the target images for the current patch. You can begin by right-clicking the Previous Setups icon and selecting Add New Previous Setup, if desired renaming the new icon. InstallShield automatically provides one icon for a previous setup.




In the Previous Setup Path field, you browse for the earlier version's MSI database or Setup.exe launcher. Because the patch-creation process requires an uncompressed release, you will be prompted for a location to decompress the previous version, if necessary.

In the Advanced tab for a previous version (not pictured), you can specify settings such as the version relationship required between the latest and previous versions, and which product-version fields to check to determine if a patch is appropriate for a particular product version on the target system. Recent InstallShield versions support streamlining a QuickPatch package, which removes subfeatures and custom actions that are unnecessary for simple QuickPatch projects.

In the Additional External Files folder, you can specify files that you want to patch, but which were not installed by your previous setups. The data you enter here is written to the optional ExternalFiles table of your PCP file.

Finally, to build the patch, select the patch-configuration icon and click the Build the Patch button. The build will return an error and exit if your latest setup version was built compressed. By default, InstallShield will also perform upgrade and patch validation.

Moreover, recent versions of IsCmdBld support a -patch_config switch for building patches from the command line.

**TIP:** In Windows, you can right-click a built .msp patch file and select Edit with InstallShield to launch the Open MSP Wizard, with which you can view and edit the package changes contained in a patch file.

*NOTE: Deploying an effective minor upgrade—either as a full installation package or as a patch—requires the REINSTALL and REINSTALLMODE properties to be set. You have seen how these properties can be set at the command line, and how InstallShield can create a setup launcher that populates the properties. In addition, the properties can be set by custom actions or control events inside the running installation. The one exception is that the "v" flag for REINSTALLMODE, which indicates to re-cache the updated MSI database during a minor upgrade packaged as a full installation database, must be set outside the installation program, and cannot be changed from within a running installation.*

## Summary

This white paper discusses how to create minor and major upgrades, and how to package updates as full installation databases or as patches.