InstallShield
PROFESSIONAL SERVICES

Upgrades Using Windows Installer

Technical White Paper

By Hitesh Pandya

Table of Contents

1. Abstract

This white paper explains the different types of updates available in Windows Installer.

2. Description

2.1 Types of Updates

The Windows Installer service supports three types of upgrades:

- Small Upgrade
- Minor Upgrade
- Major Upgrade

The three types of updates can be defined as follows:

2.1.1 Small Update

A small update only makes changes to one or more application files. These changes are judged to be too minor to warrant changing the Product Code and Product Version property. Because a small update changes the information inside the MSI file, changing the package code is necessary. The package code is stored in the Revision Number Summary Property of the Summary Information Stream.

As the product code is never changed while using the small update, all of the changes introduced by the small update should be consistent with the guidelines described in Section 2.2.

2.1.2 Minor Update

A minor update makes changes to the product database and files. The changes are large enough to merit a change to the Product Version property, but not to the Product Code property. Minor upgrades provide product differentiation without actually defining a different product.

Changing the product version indicates that there is an order to the different updates to the same product. For example, if a patch existed to update v9.0 to v9.1, and another patch existed to patch v9.1 to v9.2, the installer can enforce the correct order by checking the product version before applying the patch. This also prevents the v9.1 to v9.2 patch from being applied to v9.0.

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.

2.1.3 Major Upgrade

A major upgrade includes changes large enough to merit changes to both the Product Version and Product Code properties. Major upgrades are available in Microsoft Windows Installer version 1.1 and later versions. Installer version 1.0 does not support updates that change the product code.

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 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 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.

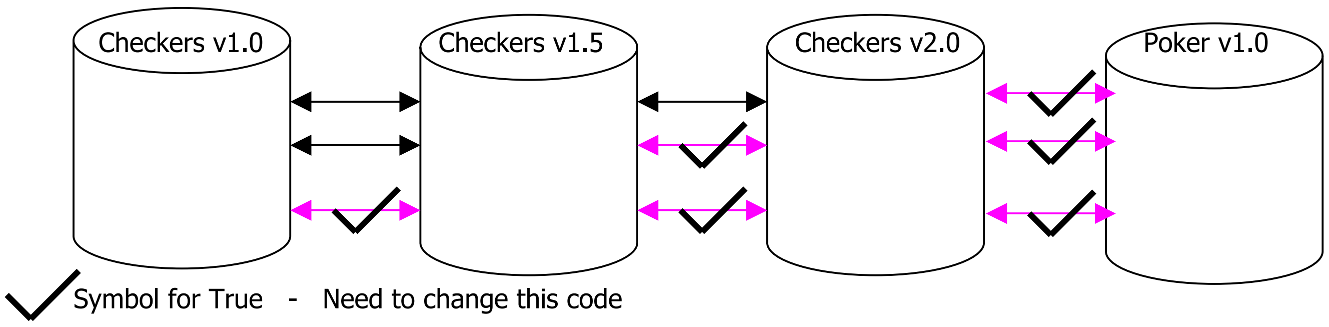
A major upgrade can be shipped as a patch package or as a full product installation package.

2.2 What GUID to Change and When

There are three very important GUIDs in any project:

- Upgrade code
- Product code
- Package code

These GUIDs are used by Windows Installer service to decide what to do with this package, whether it has been installed before, and how to handle upgrading previous versions. To explain in more depth the rules that are applied when Windows Installer finds a GUID in your package, we will look at a fictitious example, using two programs from a single vendor.



2.2.1 Upgrade Code

The upgrade code uniquely identifies the family of products for any of your future releases, and is used as an anchor point for any future versions of your software so it should NEVER be changed. The upgrade code is used in the Upgrade table to search for related versions of the product that are already installed. The Upgrade table is used to control automated un-installation of previous versions of your software. The upgrade code only changes between products.

2.2.2 Product Code

The product code uniquely identifies a particular product release. Product code is unique on each machine. Once you run the MSI on a machine, the database is cached on that particular machine. So if Windows Installer recognizes the product code in a new MSI file, then by default it runs the cached database.

2.2.3 Package Code

The package code is a GUID identifying a particular Microsoft Windows Installer package. No two non-identical MSI 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 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.

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.

For a particular product release, the following table explains what codes to change and when:

	Small Update	**Minor Update**	**Major Upgrade**
Package Code	*Change*	*Change*	*Change*
Product Code	*Do Not Change*	*Do Not Change*	*Change*

Upgrade Code	*Never Change*	*Never Change*	*Never Change*
Product Version	*Do Not Change*	*Change*	*Change*

2.3 Deploying Updates

2.3.1 Small Update and Minor Update Using Reinstalling

The simplest method of the performing small update or minor update for an installed product is to reinstall the product. By design, launching of an installation package a second or later time causes the Windows Installer service to refer to the cached package on the target system. So updates cannot be deployed from control panel applet on a target machine. To instruct Windows Installer not to use the cached package, we pass REINSTALLMODE property value that includes a letter v to MsiExec.

```
MsiExec /I C:\NewPackage.msi REINSTALLMODE=voums REINSTALL=ALL
```

The other letters used in the value of REINSTALLMODE specifies Windows Installer to:
O = Reinstall if the file is missing or an older version is present
U = Rewrite all required user-specific registry entries
M = Rewrite all machine-specific registry entries
S = Overwrite all existing shortcuts
(For more command-line options, please refer to MSI help for command-line parameters)

We can also use /f option to Msiexec (again including the letter v) to repair the existing installation, as follows:

```
MsiExec /fvoums C:\NewPackage.msi
```

The updated MSI will be cached on the end-user's machine.

If the setup program uses Setup.exe generated by ISWI or InstallShield Developer, we can pass the desired command line to MsiExec using /v argument.

```
Setup.exe /v"REINSTALLMODE=voums REINSTALL=ALL"
```

Similarly, we can specify always to set the REINSTALLMODE and REINSTALL properties with Setup.exe by modifying the CmdLine entry in the [Startup] section of the Setup.ini file.

```
[Startup]
CmdLine=REINSTALLMODE=voums REINSTALL=ALL
```

Specifying REINSTALL=ALL at the command line will only install the features that are previously being installed on that system. So running the installation will install no files on a system that does not already have the product installed. To enable the setup to work on a system that does not already have the product installed, we create a Type-51 custom action that Undefines the REINSTALL if the condition "Not Installed" is true.

If the update contains any new features that we wish to install, we set the ADDLOCAL, ADDSOURCE, and ADDDEFAULT properties at the command line with the desired feature names; and for new components we use COMPADDLOCAL or COMPADDSOURCE with the desired component codes. If the update contains new feature along with new component, we need to specify ADDLOCAL or ADDSOURCE and COMPADDLOCAL or COMPADDSOURCE in the command line.

2.3.2 Major Upgrade

Performing a major upgrade requires the value of the Product Code property changed. Therefore running the updated package on top of the previous version will by default result in first time installation. A major upgrade is performed using the Upgrade table inside the Windows Installer database. If we wish to uninstall the previous versions of the application, we need to populate the upgrade table of the updated project. The fields of the upgrade table store the following values:

Upgrade Code: This column contains the list of the upgrade codes of all the products that are to be detected by FindRelatedProducts action.

VersionMin: This column contains the lower bound on the range of product versions that needs to be detected by FindRelatedProducts action.

VersionMax: This column contains the upper bound on the range of product versions that needs to be detected by FindRelatedProducts action.

Language: Specifies the set of languages detected by FindRelatedProducts. Enter a list of numeric language identifiers (LANGID) separated by commas.

Attributes: Normal attribute is 1025 (1024 for all the languages to be detected and 1 for that features state should be migrated).

Remove: This column contains a comma-delimited list of feature names that needs to be removed by the upgrade. For example: [Feature1], [Feature2]. The installer sets the REMOVE property to features specified in this column. If this field is left empty, the installer will set the REMOVE=ALL.

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 (for example: OLDPRODUCTS) and the package author must add the property to the SecureCustomProperties property. Go to Property Manager and add a property named SecureCustomProperties and set it to the value of the public property you created (OLDPRODUCTS). Each row in the ActionProperty 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.

2.4 Update Using a Patch

2.4.1 Small and Minor Update Using a Patch

Small and minor updates can also be achieved by applying a patch. A Windows Installer patch package (.msp file) has a unique patch code, binary patches or entire files to be updated, and transforms streams to modify the target package.

To apply a patch to the existing installation of a product, we use the following command line:

```
Msiexec /p C:\Patch.msp REINSTALLMODE=oums REINSTALL=ALL
```

The letter v is not required in the REINSTALLMODE property value, as /p automatically ignores the cached database. You can also run the MSP file by just double-clicking on it. A patch package contains a list of product codes it applies to, and therefore it is unnecessary to specify the code for the product being patched.

Using InstallShield Professional for Windows Installer 1.5 or later, you can also create "Update.exe" file to apply the patch. Update.exe runs the command line mentioned above. But this is easy to distribute, as most of the customers know how to run the executable file.

3. Resources

Windows Installer Online Help
MSI help documentation for Small Upgrade, Minor Upgrade and Major Upgrade