About component, package, product and upgrade codes in Windows Installer

Abstract

Example GUID:
{50EFC3E0-8AF8-11D4-94C7-00E09876D9C4}

You can seen these alphanumeric strings in curly braces - also known as GUIDs - in many places in InstallShield Professional - Windows Installer Edition. This article describes the purposes of these component IDs, package codes, product codes and upgrade codes. It gives guidelines when you should change these codes and when not. And it explains why you can't use human readable strings in these places.

About component, package, product and upgrade codes in Windows Installer

What good is a GUID

Windows installer makes heavy use of these almost unreadable alphanumeric strings in curly braces, also known as "GUID", like this:

```
{50EFC3E0-8AF8-11D4-94C7-00E09876D9C4}
```

GUID is the abbreviation for Globally Unique Identifier. It is a 128 bit number, represented as a string of hexadecimal digits. There is an operating system function that can create such unique numbers. To a "very high degree of certainty" (as Microsoft puts it), this function returns a unique value – no other invocation, on the same or any other system, should return the same value. To ensure uniqueness across machines, the ID of the network card is used (among others) to compute the number. Therefore it is advisable that your development machine be equipped with a network card, else there is a slight chance that another computer could generate an identical GUID.

GUIDs are nothing new. For instance they are used to identify classes, objects and interfaces in ActiveX applications. However Windows Installer has introduced a new requirement for GUIDs: While tools like Visual Studio often use lower case letters to represent hex digits, GUIDs used in Windows Installer must be in all upper case characters. I have no idea why Microsoft decided to build in such a pitfall, but if you use the "Generate GUID" buttons that InstallShield has placed in all relevant locations of the IPWI development environment, you don't have to worry about this.

The sections below describe some of the locations where GUIDs are used, and why it is vital to have identifiers that are unique across development teams and companies.

Component ID

Components are the building blocks of an msi package. They can include files, registry entries and shortcuts. The Windows Installer reference counting mechanism is based on this component code: 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. Therefore, 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. If you can't guarantee this rule, you must isolate this resource as its own component and set its "shared" flag to "yes". The same should be done if the file is also used in legacy setup packages that don't use the Windows Installer service.

If you change the component ID, you must also change the names and/or locations of all included resources, and vice versa: if you change the name of an application file, you must also change its component ID.

Package Code

As the name implies, the package code identifies a specific msi file. I want to emphasize: not a product, but an msi file. No two msi files that are not identical copies of each other should ever have the same package code, even if they install (different versions of) the same product. Windows Installer keeps copies of all installed msi files in a cache. If you start a Windows Installer setup, the runtime engine first checks to see if an msi file with the same package code already exists in the cache, and uses this copy instead. So whenever you rebuild your msi file you should give it a new package code. You must do this at least for each package that you release to the wild (including beta testers), but in my opinion it is a good idea to do this for each build.

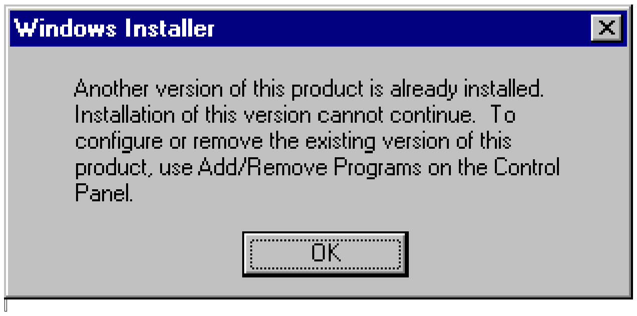
To change the package code, go to the Summary Information Stream panel, put the cursor in the Package Code field, and press the "Generate GUID" button in the lower part of the screen. In an automated build environment you can also change the package code using the ISWiAutomation object.

About component, package, product and upgrade codes in Windows Installer

Product Code

This code should only be changed if significant changes are made to the application - changes that warrant to call it a different product. If you are just releasing a new version of your program, the product code should not be changed. Of course, two different products should never have the same product code. If you have two flavors of a product, say "YourApp Express" and "YourApp Professional", these should have different product codes. The same is true for different language versions of a program - they must use different product codes. Windows Installer cannot install two instances of the same product (i.e. two msi packages with the same product code) on the same machine.

Using different product codes is required to allow two applications to coexist on the same computer. If you try to install two msi packages with different package codes, but with the same product code, you will get an error message as shown in figure 1.


Windows Installer

Another version of this product is already installed. Installation of this version cannot continue. To configure or remove the existing version of this product, use Add/Remove Programs on the Control Panel.

Figure 1: Reinstallation error message

To overwrite the existing product with a newer version, you should perform a small or minor update. This requires that you set the following properties on the

```
msiexec command line: msiexec /i Yourapp.msi REINSTALLMODE=vomus
REINSTALL=ALL
```

The important part is the "v" in the reinstall mode settings. It forces the use of the new msi file instead of the cached copy. Therefore it can't be set inside the package. The rest of the REINSTALLMODE flags make sure that existing files get updated, new files get installed, registry entries are re-written and shortcuts are created. REINSTALL=ALL means that only those features, that were selected by the user during the install of the old version get updated. Unselected features should not be added.

If you selected the option "Create installation launcher (setup.exe)" when you built the release, you can easily specify these parameters in the generated Setup.ini file:

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

Small and minor updates are very similar. The only difference is that in a minor update the product version is increased (in any of the first thee fields), while a small update leaves the version number unchanged or increases only the fourth field of the number. You can update your product with a small or minor update package only if the product code is unchanged. To replace an existing application with a package that has a different product code, a major upgrade is required.

The following changes in your setup project *require* that you change the product code:

- The name of the .msi file has been changed.
- The component code of an existing component has changed.
- A component has been added or 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.

Note that adding a new feature (top level or child), consisting entirely of new components, does not require changing the product code.

Upgrade Code

All applications in a product family shared the same upgrade code. Such a group of related applications can consist of different versions and different language versions of the same product. You should never change this code, unless you want to prevent major upgrades. A major upgrade could replace YourApp Express with YourApp Professional. Therefore both members of the YourApp family should have the same upgrade code. When you install YourApp Professional, Windows Installer will detect that another family member is already installed, and automatically remove YourApp Express before installing YourApp Professional. Windows Installer is smart enough to keep any components that a shared in both editions on the system, thus reducing installation time to a minimum. Of course this requires that you properly populated the Upgrade Table.

About component, package, product and upgrade codes in Windows Installer

Summary

Table 1 summarizes when to change the package, product and upgrade codes, and the product version.

Update Type	Package Code	Product Version	Product Code	Upgrade Code
Small update	change	don't change	don't change	don't change
Minor update	change	change	don't change	don't change
Major upgrade change	change	change	change	don't change

Further Reading

The following links will take you to other web sites. They will open in a new browser window.

- Organizing applications into components (Microsoft Platform SDK)
- Changing the product code (Microsoft Platform SDK)
- Updates and patches (InstallSite.org)

These hyperlinks were accurate as of this writing. If they appear to be broken, please use the search functions at the respective sites.

About the Author

Stefan Krueger is working as freelance setup consultant. Besides his contract work, he answers questions in various newsgroups and maintains the InstallSite.org web site, a place where setup developers share resources and information among peers.

If you have any comments about this article, or want to suggest a topic that Stefan should discuss in a future article, please write to isnewsarticle@installsite.org. To read Stefan's articles from previous issues of the InstallShield Newsletter, please visit http://www.installsite.org/isnews.htm.