**Version: v3**

# How To: Implement a Major Upgrade In Your Installer

When creating an .msi-based installer, you are strongly encouraged to include logic that supports Windows Installer major upgrades. Major upgrades are the most common form of updates for .msi's, and including support in your initial .msi release gives you flexibility in the future. Without including support for major upgrades you risk greatly complicating your distribution story if you ever need to release updates later on.

You can use the following steps to enable major upgrades in your .msi, build multiple versions of your .msi and test major upgrade scenarios.

## Step 1: Add upgrade information needed to cause new versions to upgrade older versions

In order to allow major upgrades, you must include the following information in your .msi:

### Add a unique ID to identify that the product can be upgraded

To accomplish this, you must include an UpgradeCode attribute in your Product element. This looks like the following:

```
<Product Id="*"
         UpgradeCode="PUT-GUID-HERE"
         Name="My Application Name"
         Language="1033"
         Version="1.0.1"
         Manufacturer="My Manufacturer Name"/>
```

## Schedule the removal of old versions and handle out-of-order installations

The MajorUpgrade element upgrades all older versions of the .msi. By default, it prevents out-of-order installations: installing an older version after installing a newer version.

```
<MajorUpgrade
  DowngradeErrorMessage="A later version of [ProductName] is already installed. Setup will now exit.">
```

There are several options for where you can schedule the RemoveExistingProducts action to remove old versions of the .msi. You need to review the options and choose the one that makes the most sense for your scenarios. You can find a summary of the options in the RemoveExistingProducts documentation.

By default, MajorUpgrade schedules RemoveExistingProducts after InstallValidate. You can change the scheduling using the Schedule attribute. For example, If you choose to schedule it after InstallInitialize, it will look like the following:

Windows Installer looks for other installed .msi files with the same UpgradeCode value during the FindRelatedProducts action. If you do not specifically schedule the FindRelatedProducts action in your setup authoring, WiX will automatically schedule it for you

```
<MajorUpgrade
  Schedule="afterInstallInitialize"
  DowngradeErrorMessage="A later version of [ProductName] is already installed. Setup will now exit.">
```

## Step 2: Build version 1 and version 2 of your .msi

Creating version 1 of your .msi is as simple as running your standard build process - this means you compile and link it with the WiX toolset. In order to create version 2 of your .msi, you must make the following changes to your setup authoring, then re-run your build process to create a new .msi:

- Increment the Version value in your Product element to be higher than any previous versions that you have shipped. Windows Installer only uses the first 3 parts of the version in upgrade scenarios, so make sure to increment your version such that one of the first 3 parts is higher than any previously shipped version. For example, if your version 1 uses Version value 1.0.1.0, then version 2 should have a Version value of 1.0.2.0 or higher (1.0.1.1 will not work here).
- Generate a new Id value in the Product element of the new version of the .msi.

# Step 3: Test upgrade scenarios before you ship version 1

This step is very important and is too often ignored. In order to make sure that upgrade scenarios will behave the way you expect, you should test upgrades before you ship the first version of your .msi. There are some upgrade-related bugs that can be fixed purely by making fixes in version 2 or higher of your .msi, but there are some bugs that affect the uninstall of version 1 that must be fixed before you ship version 1. Once version 1 ships, you are essentially locked into the uninstall behavior that you ship with version 1, and that impacts major upgrade scenarios because Windows Installer performs an uninstall of version 1 behind the scenes during version 2 installation.

Here are some interesting scenarios to test:

- Install version 1, then install version 2. Make sure that version 1 is correctly removed and version 2 functions correctly. Make

sure version 2 cleanly uninstalls afterwards.

- Install version 2, then try to install version 1. Make sure that version 1 correctly detects that version 2 is already installed and either blocks or silently exits, depending on what behavior you choose to implement for your out-of-order installation scenarios.

When testing major upgrade scenarios, make sure to pay particular attention to the conditions on custom actions in your .msi because you may run into issues caused by custom actions running during a major upgrade uninstall and leaving your product in a partially installed state. The UPGRADINGPRODUCTCODE property can be useful to prevent actions from running during an uninstall that is invoked by the RemoveExistingProducts action.

In addition, pay attention to assemblies that need to be installed to the GAC or the Win32 WinSxS store. There is some information about a sequence of events that can remove assemblies from the GAC and the WinSxS store during some major upgrades in this knowledge base article.

Edit this page