Windows Installer patches

Windows Installer patching allows you to update an installed product without a full upgrade. Patches contain the differences between an older version of a package (called the target) and the newer version (the update). Because they contain only the differences, patches are typically smaller than the full update package and typically install more quickly.

That said, patching is an advanced topic and requires strict adherence to all the rules that come with Windows Installer. Using major upgrades with "early" scheduling, such as using the default `Schedule` attribute of the `MajorUpgrade` element is far simpler. As Bob Arnson discussed way back in 2008:

> If you don't absolutely need to ship patches, you can avoid the costs of minor upgrades by simply using major upgrades. You can remove files without worrying about component-rule violations if you use an "early" scheduling of the RemoveExistingProducts standard action – before or immediately after the InstallInitialize action.

> The MSI SDK notes that scheduling RemoveExistingProducts early is "inefficient" because the files that are same between the two product versions are removed and then reinstalled. But that inefficiency is what lets you remove files and components.

Authoring patches

Patch authoring starts with the `Patch` element. Attributes supply some metadata about the patch and child elements define the cabinet containing updated files and the target and update packages. The target and update packages can be specified as .wixpdb or as .msi files.

Authoring patches using .wixpdb files

.wixpdb files are produced during the WiX build and, like .pdb files from your favorite Visual Studio compiler, contain symbol information produced during the build process beyond what's recorded in the final output (like an .msi file). For patches, one bit of that extra information is the paths that were used when building the target and update packages to choose the payload files. When use you bind paths, WiX can use path information from the .wixpdb files to locate files when building the patch content.

Here's what authoring for a patch might look like:

```
<Wix xmlns="http://wixtoolset.org/schemas/v4/wxs">
  <Patch
    AllowRemoval="yes"
    DisplayName="Product Patch v$(Version)"
    Description="Product Patch v$(Version)"
    MoreInfoURL="https://wixtoolset.org/"
    Manufacturer="WiX Toolset"
    Classification="Update">

    <Media Id="1" Cabinet="patch.cab">
      <PatchBaseline
        Id="RTM"
        BaselineFile="PackageV1.wixpdb"
        UpdateFile="PackageV2.wixpdb" />
    </Media>

    <PatchFamily Id="SequenceFamily" Version="$(Version)" />
  </Patch>
</Wix>
```

Building it would look like this:

```
wix build -define Version=2.0 -bindpath path\to\V1PackageOutput -bindpath path\to\V2PackageOutput
-bindpath:target path\to\V1PackagePayloads -bindpath:update path\to\V2PackagePayloads -out path\to
\patch.msp Patch.wxs
```

The following table describes the switches used for building the patch:

Switch	Description
`-define`	Passes the patch version into the patch authoring.
`-bindpath path\to\V1PackageOutput`	Provides a bind path to find the target package.
`-bindpath path\to\V2PackageOutput`	Provides a bind path to find the update package.
`-bindpath:target path\to\V1PackagePayloads`	Tells WiX to use the specified path when locating files from target .wixpdb.
`-bindpath:update path\to\V2PackagePayloads`	Tells WiX to use the specified path when locating files from update .wixpdb.
`-out path\to\patch.msp`	Specify the output is a patch.

Authoring patches using .msi files

You can also use .msi files as the target and update packages. The primary advantage of using .msi files over .wixpdb files is that the .msi files are the source of both the changes between the target and update packages but also of the payload files themselves. So rather than supplying .wixpdb files and paths for the payload files, you need to supply just the .msi files. The primary disadvantage of using .msi files is that WiX needs to extract files from the .msi packages, which typically increases the time needed to build the patch.

The patch authoring itself is identical other than file paths:

```
<PatchBaseline
  Id="RTM"
  BaselineFile="PackageV1.msi"
  UpdateFile="PackageV2.msi" />
```

Building it would look like this:

```
wix build -define Version=2.0 -bindpath path\to\V1PackageOutput -bindpath path\to\V2PackageOutput  -out
PatchV1V2.msp Patch.wxs
```

The following table describes the switches used for building the patch:

Switch	Description
`-define`	Passes the patch version into the patch authoring.
`-bindpath path\to\V1PackageOutput`	Provides a bind path to find the target package.

Switch	Description
`-bindpath path\to\V2PackageOutput`	Provides a bind path to find the update package.
`-out path\to\patch.msp`	Specify the output is a patch.

Edit this page