This is documentation for WiX Toolset **v3**, which is no longer actively maintained.

For up-to-date documentation, see the **latest version** (v4.0).

Version: v3

How To: Specify source files

WiX provides three ways of identifying a setup package's payload - the files that are included in the setup and installed on the user's machine.

- By file name and directory tree.
- By explicit source file.
- Via named binder paths.

Compiling, linking, and binding

The WiX toolset models a typical C/C++ compiler in how authoring is built, with a compiler that parses the WiX source authoring to object files and a linker that combines the object files into an output. For WiX, the output is an .msi package, .msm merge module, or .wixlib library, which have a third phase: binding payload files into the output. Light.exe includes both the linker and binder.

Though WiX source authoring refers to payload files, the compiler never looks at them; instead, only the binder does, when it creates cabinets containing them or copies them to an uncompressed layout.

You can provide the binder with one or more *binder input paths* it uses to look for files. It also looks for files relative to the current working directory. Light.exe's -b switch and the BindInputPaths .wixproj property let you specify one or more binder input paths.

Binder input paths can also be prefixed with a *name* which will append that path to the identified binder input path bucket (unprefixed paths will be added to the unnamed binder paths bucket). The bucket name must be more than two characters long and be followed by an equal sign ("="). See an example in the *Identifying payload via named binder paths* section

Identifying files by name and directory tree

When you use the File/@Name attribute and don't use the File/@Source attribute, the compiler constructs an implicit path to the file based on the file's parent component directory plus the name you supply. So, for example, given the partial authoring

```
<Directory Id="TARGETDIR">
  <Directory Name="foo">
    <Directory Name="bar">
      <Component>
        <File Name="baz.txt" />
```

the binder looks for a file *foo\bar\baz.txt* in the unnamed binder input paths.

Overriding implicit payload directories

The FileSource attribute for the Directory and DirectoryRef elements sets a new directory for files in that directory or any child directories. For example, given the partial authoring

```
<Directory Id="TARGETDIR">
  <Directory Name="foo" FileSource="build\retail\x86">
    <Directory Name="bar">
```

```
          <Component>
            <File Name="baz.txt" />
```

the binder looks for a file *build\retail\x86\bar\baz.txt* in the unnamed binder input paths.

The FileSource attribute can use preprocessor variables or environment variables. If the value is an absolute path, the binder's unnamed input paths aren't used.

Preferred use

If the build tree serving as your payload source is almost identical to the tree of your installed image and you have a moderate-to-deep directory tree, using implicit paths will avoid repetition in your authoring.

Source directories

The Directory/@SourceName attribute controls both the name of the directory where Light.exe looks for files and the "source directory" in the .msi package. Unless you also want to control the source directory, just use FileSource.

Identifying payload by source files

The File/@Source attribute is a path to the payload file. It can be an absolute path or relative to any unnamed binder input path. If File/@Source is present, it takes precedence over the implicit path created by Directory/@Name, Directory/@FileSource, and File/@Name.

If you specify File/@Source, you can omit File/@Name because the compiler automatically sets it to the filename portion of the source path.

Preferred use

If the build tree serving as your payload source is different from the tree of your installed image, using File/@Source makes it easy to pick explicit paths than are different than the .msi package's directory tree. You can use multiple unnamed binder input paths to shorten the File/@Source paths.

For example, the WiX setup .wixproj project points to the output tree for the x86, x64, and ia64 platforms WiX supports and the WiX source tree. Unique filenames can be referred to with just their filenames; files with the same name across platforms use relative paths.

See the WiX authoring in src\Setup\*.wxs for examples.

Identifying payload via named binder paths

This is similar in authoring style to "Identifying payload by source files" while searching multiple paths like "Identifying files by name and directory tree". As such, it is sort of a hybrid between the two.

Named bind paths uses the File/@Source path prefixed with a bindpath variable like !(bindpath.*bucketname*). As with the unnamed binder paths used when the File/@Source is not present each path tagged with the same bucket name will be tested until a matching file is found. If the resulting path is not an absolute filepath, the unnamed binder file paths will be searched for each string in the bucket.

```
<File Source="!(bindpath.foo)bar\baz.txt" />
<File Source="!(bindpath.bar)baz\foo.txt" />

light -b foo=C:\foo\ -b bar=C:\bar\ -b foo=D:\
```

will look for the baz.txt file first at *C:\foo\bar\baz.txt* and then at *D:\bar\baz.txt*, using the first one found, while looking for the foo.txt file at *C:\bar\baz\foo.txt*; while

```
<File Source="!(bindpath.foo)bar\baz.txt" />
<File Source="!(bindpath.bar)baz\foo.txt" />

light -b foo=foo\ -b bar=bar\ -b foo=baz\
```

will search for the baz.txt file as if looking for two files having File/@Source values of *foo\bar\baz.txt* and *baz\bar\baz.txt* and will search for the foo.txt file as if the File/@Source was *bar\baz\foo.txt*.

Preferred use

If the build tree serving as your payload source places the same category of files in several locations and you need to search those locations differently for different categories of payload source files, using File/@Source with the "!(bindpath.*bucketname*)" prefix makes it easy to pick explicit groups of search paths. You can use multiple unnamed binder input paths to shorten the File/@Source paths and/or the unnamed binder paths.

For example, a partial build system may separate binary and non-binary files to different paths stored on a network share while the local override build may not have them separated. By prefixing the File/@Source values with the appropriate bindpath variable unique filenames can be referred to with just their filenames while files with the same name across platforms use relative paths.

Edit this page