How to Create an Application Installer using Windows Installer XML Toolset

After putting many hours of labor into a software application, the last thing that you want to do is spend a large amount of time creating an installer so that you can share your creation with the world. Creating a reliable, robust, and maintainable installation package can be a daunting and time consuming task, and many software developers are employed as specialists in this area for that very reason. Fortunately, there are options available that the non-specialists can take advantage of. Creating simple installation packages can be done with minimal effort, yet still take advantage of the powerful installation services provided with Microsoft Windows. In this article, we introduce the Windows Installer XML (WiX) toolset and put it into perspective with the other installer technology options available.

Contents

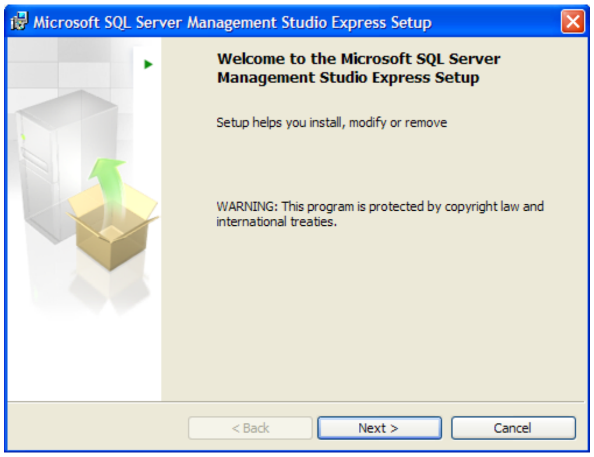
Choosing the right technology for the job

There are a few different installation methods typically used to setup software on the machines of users. The first method, which may not constitute a true installer, is to simply copy the needed files to a computer. This file copy method can be performed over a network, CD, or even the Web. If you have an application that can stand alone and does not require any additional setup, this may be the way to go – the simplest solution might be best here.

Another option is to use Microsoft's ClickOnce technology, which you can access through the publication wizard in Express. ClickOnce is a great option for managed applications that have little need for customizing the install process. However, if you need to perform install-time tasks such as create a virtual Web directory, create a new user account, or add registry keys, then you will need to consider other installation options.

You could create your own custom installer or provide scripts and instructions to your users, but this is probably not a viable option for a number of reasons. Creating your own robust installer would be a difficult and time-consuming endeavor, not to mention confusing to users who are used to more common products. Providing step-by-step instructions or scripts to your users is also not desirable from both usability and maintainability perspectives.

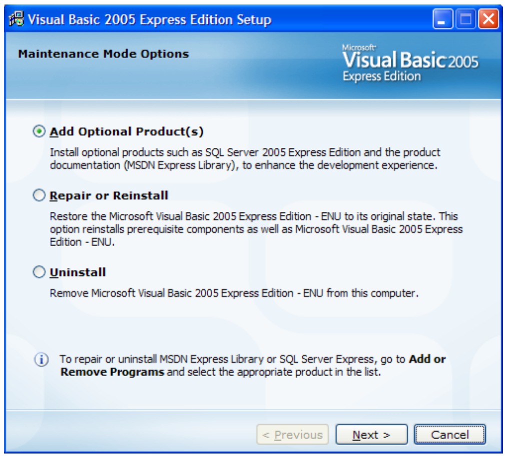
Today, the use of Microsoft Windows Installer technology is quite common. Most Windows users are quite familiar with the common interfaces that enable them to choose an installation location, choose features, repair, uninstall, and so on.


Microsoft SQL Server Management Studio Express Setup

Welcome to the Microsoft SQL Server Management Studio Express Setup

Setup helps you install, modify or remove

WARNING: This program is protected by copyright law and international treaties.


Visual Basic 2005 Express Edition Setup

Maintenance Mode Options

[✓ **Add Optional Product(s)**]
Install optional products such as SQL Server 2005 Express Edition and the product documentation (MSDN Express Library), to enhance the development experience.

Repair or Reinstall
Restore the Microsoft Visual Basic 2005 Express Edition - ENU to its original state. This option reinstalls prerequisite components as well as Microsoft Visual Basic 2005 Express Edition - ENU.

Uninstall
Remove Microsoft Visual Basic 2005 Express Edition - ENU from this computer.

To repair or uninstall MSDN Express Library or SQL Server Express, go to **Add or Remove Programs** and select the appropriate product in the list.

Screenshot - Common user interface screens during installation

Windows Installer is tightly integrated with the operating system so that developers have a common platform that is guaranteed to be available on their customers' computers. Even some of the most commonly used third-party installer creation software, such as InstallShield and Wise, use Windows Installer technology under the covers. Although many of the third-party tools that make Windows Installer easier to use have a fee associated with them, there are some community project efforts that may be worth looking into. In this article, we will take a look at one of those community projects called the Windows Installer XML toolset, or WiX for short.

Introduction to Windows Installer

Let's take a few moments to describe Windows Installer so that we have a better understanding of the platform that WiX is built upon. Windows Installer, which is distributed with the various versions of Microsoft Windows, is implemented as a service for high availability. Setup files using the MSI extension contain all of the data files and structure necessary for Windows Installer to install a software application. When targeting Windows Installer, we are basically responsible for telling the installer service "what" to install more so than "how" to install it.

The benefits of using Windows Installer include installation on demand, rollback on failure, self-repair, patching, uninstall, and more. For more information on Windows

Installer, see the references and resources section at the end of this article. Now we are ready to look at what the WiX toolset can do for us. Prepare yourself for some XML!

WiX toolset background

WiX actually started out as an internal tool to Microsoft, where is still broadly used by product teams today. In a blog entry by Rob Mensching (posted April 5, 2004) where the release of the WiX toolset is announced as an open source project, Microsoft Office, SQL Server, BizTalk, Virtual PC, and Instant Messenger are listed as products which are packaged using the WiX toolset. This evidence proves an important point – that WiX can help you do anything your setup needs require!

The WiX toolset provides a schema that describes a Windows Installer database file (the MSI file we previously discussed), plus a number of tools to compile and link the WiX source files into a final working database. From a high level, there really isn't much more to it than that. There are many details that you need to be aware of that relate to both WiX and Windows Installer databases, but once you have a acquired some basic understanding of the tools and the other resources available, you will be fully equipped to take your application installation requirements and translate them into full-featured installers.

Download and setup of the WiX toolset

The official WiX project is currently hosted on SourceForge at http://wix.sourceforge.net/, where you will find the latest news, documentation, and updates. For the purposes of this article, download the version 2.0 binaries package which, at the time of this writing, was listed as the current stable version. As you browse the version 2.0 branch of the WiX download page, you will notice two other packages that are available – one containing the source code and another containing a project named Votive, which adds integrated WiX support to Visual Studio Standard or higher. You only need to download the binaries package to utilize the WiX toolset.

Once you have downloaded the WiX 2.0 binaries to your development system, extract the contents of the package to a location of your choosing. The main folder of the WiX toolset includes the command-line tools that you will use to generate, compile, and link WiX code into fully functional MSI files. In addition, documentation and examples are both available in their respective directories. The WiX.chm help file contains all of the available documentation and is a great reference tool to keep open as you work with the WiX toolset.

Introduction to WiX

All WiX source files start with an opening root element named <Wix/>. After that, you must have either one <Product/> or one <Module/> child element, plus zero or more optional <Fragment/> elements. The use of the <Product/> element leads to an .msi, which is a Windows Installer database file. You can also create a Merge Module, which uses the .msm extension, with the <Module/> element. Merge Modules are self-contained packages that can be merged into .msi products. Finally, the <Fragment/> element allows you to define pieces of WiX that can be referenced from other WiX code. The ability to create fragments becomes useful when the size of your installations increase.

Take a look at the following WiX code, which must be used for any WiX-related construct:

```
<?xml version='1.0'?>
<Wix xmlns='http://schemas.microsoft.com/wix/2003/01/wi'>
</Wix>
```

Once you have created the skeleton for a WiX source file, the creation of your installer actually begins. Typical tasks of an installer include deploying files, creating a directory structure, registering with Add/Remove Programs, executing custom actions, adding user accounts, SQL databases, and Web applications. Take a look at the available elements from the WiX schema in the WiX Help file (under the WiX Help | Authoring | Wix Schema node). As you can see, with over 230 elements available, there is some complexity associated with creating installers. The good news is that you typically only need to use a small subset of these elements when creating an installer.

The following WiX code, taken directly from the “examples\first” directory of the WiX binaries package, shows how to create an installer that will create a directory in “Program Files”, copy a text file, and register with Add/Remove Programs.

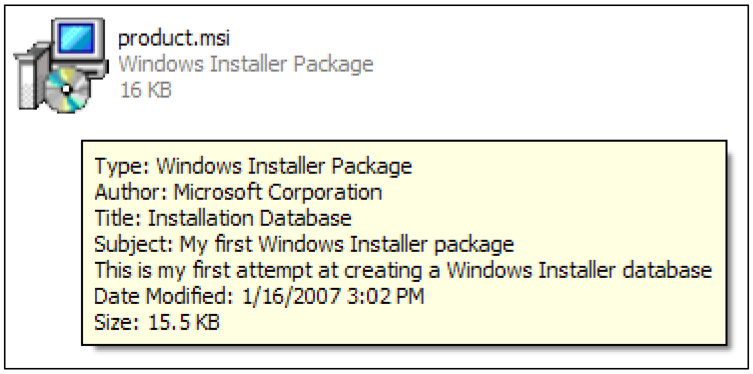
```
<Wix xmlns='http://schemas.microsoft.com/wix/2003/01/wi'>
  <Product Id='12345678-1234-1234-1234-123456789012' Name='Test Package'
          Language='1033' Version='1.0.0.0' Manufacturer='Microsoft Corporation'>
    <Package Id='12345678-1234-1234-1234-123456789012'
                   Description='My first Windows Installer package'
                   Comments='This is my first attempt at creating a Windows Installer database'
                   InstallerVersion='200' Compressed='yes' />

    <Media Id='1' Cabinet='product.cab' EmbedCab='yes' />

    <Directory Id='TARGETDIR' Name='SourceDir' >
      <Directory Id='ProgramFilesFolder' Name='PFiles'>
        <Directory Id='MyDir' Name='TestProg' LongName='Test Program'>
          <Component Id='MyComponent' Guid='12345678-1234-1234-1234-123456789012'>
            <File Id='readme' Name='readme.txt' DiskId='1' src='readme.txt' />
          </Component>
        </Directory>
      </Directory>
    </Directory>

    <Feature Id='MyFeature' Title='My 1st Feature' Level='1'>
      <ComponentRef Id='MyComponent' />
    </Feature>
  </Product>
</Wix>
```

It is worthwhile to explain some of the details shown within the WiX sample above. As you would expect, the <Wix\> element contains just one child, the <Product\> element. Take a look at the Product Element page in the WiX Schema section of the WiX Help file. If you scroll down to the Attributes section you can see the attributes defined for the Product element, including the ones required. You can verify that the example WiX code does supply all required attributes for the Product element.

Table - Product element attributes (partial list)

Name	Type	Description	Required
Id	Autogenuuid	The product code GUID for the product.	Yes
Language	LocalizableInteger	The decimal language ID (LCID) for the product.	Yes
Manufacturer	String	The manufacturer of the product.	Yes
Name	String	The descriptive name of the product.	Yes
Version	String	The product's version string.	Yes

The <Package\> element is a required child for both Product and Module elements. The Package describes properties that are made available as part of the .msi's summary stream, which can be seen when you mouse over the .msi in Explorer. The Description and Comments properties can be seen in the screenshot below.


product.msi
Windows Installer Package
16 KB

Type: Windows Installer Package
Author: Microsoft Corporation
Title: Installation Database
Subject: My first Windows Installer package
This is my first attempt at creating a Windows Installer database
Date Modified: 1/16/2007 3:02 PM
Size: 15.5 KB

Take a look at the other available attributes of the Package element in the WiX Help file documentation. Note that we can specify the installer version, file compression, and even installation privileges if targeting Windows Vista.

Table - Package element attributes (partial list)

Name	Type	Description	Required
Id	Autogenuuid	Package code GUID for SKU.	Yes
AdminImage	YesNoType	Set to 'yes' if the source is an admin image.	
Comments	String	Optional comments for browsing.	
Compressed	YesNoType	Set to 'yes' to have compressed files in the source.	
Description	String	The product full name or	

		description.	
InstallerVersion	Integer	The minimum installer version (major*100 + minor).	
InstallPrivileges	Enumeration	Use this attribute to specify the privileges required to install the package on Windows Vista and above. This attribute's value should be one of the following: *limited* Set this value to declare that the package does not require elevated privileges to install. *elevated* Set this value to declare that the package requires elevated privileges to install. This is the default value.	
Manufacturer	String	The vendor releasing the package.	
Platforms	String	The list of platforms supported in the package.	

For many installers, like the example we are looking at, it is a good idea to keep all installation source files packaged together within the MSI. The Media element allows us to do just that by specifying our desire to have files stored within a compressed and embedded .cab file. Larger installers, however, may need to span multiple disks. In this case, you would define multiple Media elements within your WiX source file and have your resources assigned to each disk as appropriate.

Table - Media element attributes (partial list)

Name	Type	Description	Required
Id	Integer	Disk identifier for Media table. This number must be equal to or greater than 1.	Yes
Cabinet	String	The name of the cabinet if some or all of the files stored on the media are compressed into a cabinet file. If no cabinets are used, this attribute must be blank.	
DiskPrompt	String	The disk name, which is usually the visible text printed on the disk. This localizable text is used to prompt the user when this disk needs to be inserted. This value will be used in the "[1]" of the DiskPrompt Property. Using this attribute will require you to define a DiskPrompt Property.	
EmbedCab	YesNoType	Instructs the binder to embed the cabinet in the product if 'yes'. This attribute can only be specified in conjunction with the Cabinet attribute.	
VolumeLabel	String	The label attributed to the volume. This is the	

		volume label returned by the GetVolumeInformation function. If the SourceDir property refers to a removable (floppy or CD-ROM) volume, then this volume label is used to verify that the proper disk is in the drive before attempting to install files. The entry in this column must match the volume label of the physical media.	

The next step of the example shows how to define the location and file to be installed. Even without prior experience with WiX or Windows Installer, you can probably surmise that the result of this step is a copy of the readme.txt file sitting in the "\Program Files\Test Program" folder. Looking at the series of nested Directory elements can be confusing at first glance, so let's go over that in more detail.

```
<Directory Id='TARGETDIR' Name='SourceDir' >
    <Directory Id='ProgramFilesFolder' Name='PFiles'>
        <Directory Id='MyDir' Name='TestProg' LongName='Test Program'>
```

MSI databases require that a single root destination directory be defined to be <Directory Id='TARGETDIR' Name='SourceDir' >. Both "TARGETDIR" and "SourceDir" are properties that are recognized by Windows Installer. The "TARGETDIR" property, if defined via command line or user interface, will be used when resolving the destination path. In the case of our simple installer, this will not be defined and Windows Installer will default to using the "ROOTDRIVE" property instead, which will likely be something like "C:\". The "SourceDir" property simply represents the directory that contains the installation package.

The next Directory element is a child of the root destination directory, defined to be <Directory Id='ProgramFilesFolder' Name='PFiles'>. "ProgramFilesFolder" is another Windows Installer property that refers to the full path to the predefined Program Files folder on the system. See http://msdn2.microsoft.com/en-us/library/aa372057.aspx for examples of other system folder properties that are available. At this point, we are likely looking at a destination install folder of "C:\Program Files\".

Finally, we have the last Directory element defined to be <Directory Id='MyDir' Name='TestProg' LongName='Test Program'>. The "MyDir" Id property does not refer to a defined property this time, so the default Name, or LongName, property will be used. This completes the directory structure for this sample, which will likely be "C:\Program Files\Test Program\".

Table - Directory element attributes (partial list)

Name	Type	Description	Required
Id	String	This value is the unique identifier of the directory entry.	Yes
FileSource	String	Used to set the file system source for this directory's child elements.	

LongName	LongFileNameType	Set this value to the non-8.3 name for the directory. This attribute cannot be specified unless the Name attribute is used to set the short name for this directory.	
LongSource	LongFileNameType	Set this value to the non-8.3 name of the directory on the source media for systems supporting long names. This attribute cannot be specified unless the SourceName attribute sets the short source name for this directory.	
Name	String	The 8.3 name of the directory. Do not specify this attribute (or the LongName attribute) if this directory represents the same directory as the parent (see the Windows Installer SDK's Directory table topic for more information about the "." operator).	
SourceName	ShortFileNameType	The 8.3 name of the directory on the source media. If this attribute is not specified, the Windows Installer will default to the Name attribute.	

After the directory structure is laid out, we can finally specify the file component to be installed. For convenience, here is the relevant snippet from the sample:

```xml
<Component Id='MyComponent' Guid='12345678-1234-1234-1234-123456789012'>
    <File Id='readme' Name='readme.txt' DiskId='1' src='readme.txt' />
</Component>
```

Components, which are also uniquely defined by a GUID, can represent resources such as files, certificates, environmental variables, COM registrations, registry keys, services, SQL databases, and Web sites. In our example, we are simply installing a text file, with the ‘src’ property referring to the location of the readme.txt file during the build process. The DiskId property refers to the Media Id property that was defined earlier.

Table - Component element attributes (partial list)

Name	Type	Description	Required
Id	String	Component identifier; this is the primary key for identifying components.	Yes
Guid	Component Guid	This value should be a guid that uniquely identifies this component's contents, language, platform, and version. It's also possible to set the value to an empty string to specify an unmanaged component. Unmanaged components are a security vulnerability because the component cannot be removed or repaired by Windows Installer (it is essentially an unpatchable, permanent component). Therefore, a guid should always be specified for any component which contains resources that may need to be patched in the future.	Yes

Location	Enumeration	This attribute's value should be one of the following: *local* Prevents the component from running from the source or the network (this is the default behavior if this attribute is not set). *source* Enforces that the component can only be run from the source (it cannot be run from the user's computer). *either* Allows the component to run from source or locally.	
Permanent	YesNoType	If this attribute is set to 'yes', the installer does not remove the component during an uninstall. The installer registers an extra system client for the component in the Windows Installer registry settings (which basically just means that at least one product is always referencing this component). Note that this option differs from the behavior of not setting a guid because although the component is permanent, it is still patchable (because Windows Installer still tracks it), it's just not uninstallable.	
Transitive	YesNoType	If this attribute is set to 'yes', the installer reevaluates the value of the statement in the Condition upon a reinstall. If the value was previously False and has changed to True, the installer installs the component. If the value was previously True and has changed to False, the installer removes the component even if the component has other products as clients.	

It is important to note that any resources added to a component are considered to be an atomic unit by Windows Installer. Everything in it will be installed and uninstalled as a group. The recommended practice here is to put each resource into its own component unless you see a specific need to do otherwise. For example, if you wanted to add a link to the text file on the desktop, that would be a prime candidate for being included in the same component.

Since there is at least one component in this installer, we need to define a parent Feature element for it. This step may seem superfluous at the moment, but that is because our installer has just one component and no UI. Creating sets of features becomes necessary once you have multiple components and logical application pieces to install, some of which may be optional. For example, installers typically separate the core functionality of an application out from optional help files.

As a reminder, the Feature element from our simple example looks like the following:

```
<Feature Id='MyFeature' Title='My 1st Feature' Level='1'>
        <ComponentRef Id='MyComponent' />
</Feature>
```

The Feature element defines "MyFeature" to include the single component that was defined earlier. Again, since we are not offering a UI to the user at this time, this feature is essentially mandatory.

Now let's take the sample WiX source code and produce a Windows Installer Package that could be installed on virtually any Windows machine by following these steps:

1. **Open** a command prompt window by selecting **Start | All Programs | Accessories | Command Prompt**.
2. **Navigate** to the **"\examples\first"** folder of your WiX binaries installation location in the Command Prompt window.
3. **Verify** that product.wxs contains the example code that we previously discussed by issuing the command:
   ```
   >notepad product.wxs
   ```
4. **Add** the WiX toolset binaries folder to the environmental path by issuing the command (replacing <WiXbinaries> with the installation location on your machine):
   ```
   >path = %path%;<WiXbinaries>
   ```
5. **Compile** the product.wxs source file using the Candle.exe compiler tool using the command:
   ```
   >candle product.wxs
   ```
   The result of the compilation step is a WiX object file named product.wixobj.
6. **Link** the product.wixobj object file using the Light.exe linker tool using the command:
   ```
   >light product.wixobj
   ```
   The result of the linking step is a Windows Installer database file named product.msi.
7. **Install** the sample product by typing the full name of the product in the Command Prompt window or by double-clicking on it using Explorer. Notice that we are shown some progress UI information, but that the entire process completes without further interaction.
8. **Verify** that the installer did its job by navigating to the "Program Files" folder and looking for a **"Test Program"** folder with the **readme.txt** file in it.
9. **Verify** that un-installation works as well by performing the necessary steps in the **Add or Remove Programs** control panel applet. Alternatively, you could also issue the following command in the Command Prompt window:
   ```
   >msiexec /x product.msi
   ```

Using WiX within the Visual Studio Express IDE

Before we finish up with our WiX discussion, let's take a moment to customize our Visual Studio Express environment so that we have both WiX Intellisense support and a few WiX template files available. Intellisense is the technology in Visual Studio that allows you to type part of a word, such as a class name or XML tag, and then receive a list of possible matches and descriptions. As you will see shortly, this is an invaluable tool when hand editing WiX code!

To add Intellisense support for WiX:

1. **Navigate** to the **"Doc"** folder of your WiX installation.
2. **Copy** all of the **XML schema files** (XSD extension) to the **"\Xml\Schemas"** folder of your Visual Studio installation location (typically in "\Program Files\Microsoft Visual Studio 8\").

To add WiX template files:

1. **Download** the WiX template Zip file package (WiXTemplates.zip) from the same location as the word document.
2. **Extract** the contents of the WiX template Zip package to your "\My Documents\Visual Studio 2005\Templates\ItemTemplates" folder.
3. **Verify** that **"WiX Merge Module.zip"** and **"WiX Product.zip"** are in the **ItemTemplates** folder.

Working with Windows Installer database files, and by extension WiX source files, necessitates the generation of globally unique identifiers (GUIDs). There are multiple formats in which GUIDs can be represented, depending upon their usage, but we will be using one that looks like the following:

> "01234567-89AB-CDEF-0123-456789ABCDEF"
> or optionally
> {"01234567-89AB-CDEF-0123-456789ABCDEF"}

Because of our need for unique GUID generation, we should use a tool that is capable of doing this for us. Here are a few suggestions:

1. Install Microsoft Visual C++ 2005 Express Edition and use the Guidgen.exe utility (in \Program Files\Microsoft Visual Studio 8\Common7\Tools). Use the "Registry Format" option to create the GUID in a WiX compatible format.
2. Create your own application. You can use the System.Guid class and its static NewGuid() method to generate new GUIDs.
3. Look for other GUID generation tools online. For example, try http://www.guidgen.com.

Putting it all together

In this section, we will take what we have learned so far and put it to use by creating an installer for a simple Windows Application.

Task 1: *Create the WiX source.*

1. **Open** up an instance of your **Visual Studio Express** version of choice.

2. **Create** a new project by selecting **File | New Project**. This will load the New Project window.
3. **Select** the **Windows Application** project template and provide a name that you will remember later.
4. **Press** the **OK** button to generate the new Windows Application project.
5. **Add** a new **WiX source file** to the project by right-clicking on the project in Solution Explorer and selecting **Add | New Item** from the context menu.
6. **Scroll down** to the **My Templates** section in the Add New Item window.
7. **Select** the **WiX Product** template.
8. **Press** the **Add** button to add the new WiX Product source file to the project.
9. **Select File | Save All** to save the project to disk. Remember the name of the project as this will be important later.
10. **Choose** an appropriate **installation location** within the Program Files folder and **assign** it to the **LongName** property of the Directory element with an Id of "INSTALLLOCATION".
11. **Change** the **Component** element **Id** to a more appropriate name, such as "MyApp". Note: spaces are not legal here.
12. **Change** the **ComponentRef** element **Id** to match the Component element Id that you modified in the previous step.
13. **Add** a new **File** element as a child to the Component element with the following:
    ```
    <File Id="app" Name="MyApp.exe" DiskId="1" src="WiXInstallTest.exe" />
    ```
    Note: The File element's 'src' property should match the name of the executable produced, which is the name of your project by default.
14. **Add** a new **Shortcut** element as a child to the File element with the following:
    ```
    <Shortcut Id="desktopShortcut"
              Name="WiXTest" LongName="WiX Test 1.0"
              Directory="DesktopFolder" />
    ```
    Note: It is important to make sure that the Shortcut is a child of the File element, not the Component element. We want the desktop shortcut to point directly to the file. "DesktopFolder" is a property understood by Windows Installer to represent the user desktop.
15. **Add** the following **Directory** element just before the last Directory element closing tag:
    ```
          <Directory Id="DesktopFolder" Name="Desktop" />
    </Directory>
    ```
    Note: The closing </Directory> tag shown above is not new; it's just a reference marker to show where the new Directory element needs to be located.
16. **Generate** a new **GUID** and **assign** it to the **Component Guid** property where it says "PUT-GUID-HERE".
17. **Generate** a new **GUID** and **assign** it to the **Product Id** property where it says "PUT-GUID-HERE".

Task 2: *Modify the project to perform post-built steps*

Note: This task is for C#, J#, and C++ users only. If you are using VB then you will need to compile and link the WiX code yourself.

1. **Right-click** on your **project** in Solution Explorer and select **Properties** from the context menu.
2. **Select** the **Build Events** tab (for C#, J#).
   Or
   Select the **Configuration Properties | Build Events | Post-Build Event** node (C++).
3. In the **"Post-build event command line"** text box, enter the following:
```
"C:\wix 2.0.4820.0-binaries\candle.exe" "WiXProduct1.wxs"
"C:\wix 2.0.4820.0-binaries\light.exe" "WiXProduct1.wixobj"
```
4. **Select** the **WiX source file** in Solution Explorer and view the **Properties** window.
5. **Change** the **"Copy to Output Directory"** property to **"Copy if newer"**.

Task 3: *Build and test deploy*

1. **Select Build | Build Solution** from the main menu.
2. **Navigate** to the **\Build\Debug** folder and **execute** the **installer**.
3. **Verify** that the **desktop link** to the application works.
4. **Uninstall** the application using **Add/Remove Programs** in the Control Panel.

Summary

In this article, we introduced the Windows Installer XML toolset and provided a small taste of what it can do to satisfy your installation needs. Once you have the WiX toolset installed on your development machine and integrated with your build environment, the creation of a new installer is straightforward, albeit somewhat complicated considering the many details involved.

Always remember to weigh your options against your installation needs to determine if creating a full Windows Installer database is truly necessary. For example, if you have created a self-contained Windows Forms or Windows Presentation Framework application, using ClickOnce technology would make deployment easier for the developer and consumer.

Keep in mind that we have only scratched the surface of this topic, and that WiX allows you to execute custom actions, define user interfaces, perform COM component registration, and much more. Refer to the WiX documentation and the official WiX site, which can both be found in the references section below, for more details.

References and additional resources

Official Windows Installer XML toolset site:
http://wix.sourceforge.net/

Rob Menching blog entry (WiX contributor)
http://blogs.msdn.com/robmen/archive/2004/04/05/107709.aspx

Justin Rockwood blog (WiX contributor)
http://blogs.msdn.com/jrock/

Choosing a Deployment Strategy
http://msdn2.microsoft.com/en-us/library/e2444w33(VS.80).aspx

Windows Installer Deployment
http://msdn2.microsoft.com/en-us/library/2kt85ked(VS.80).aspx

Windows Installer Best Practices
http://msdn2.microsoft.com/en-us/library/bb204770.aspx

Windows Installer Wikipedia article
http://en.wikipedia.org/wiki/Windows_Installer