Create an installer for website with WIX, part 1

46 Replies

For a customer we created a new web application in MVC4 with an underlying SQL database. One of the requirements was to provide an installer to install this website at their customers local installations. The installer had to do a few tasks:

- Install the .NET 4.5 framework if that isn't installed already
- Install the MVC 4 framework if that isn't installed already.
- Create a folder and copy all needed files to run the application
- Create a new database on an existing SQL server and prefill the database with the correct tables and values. (the connection details and database name should be entered by the end user running the installer)
- Create a new website in IIS 7.5 (create website and application pool running under .NET 4.5)
- Alter the config file so the correct connection settings are used (entered by the end user)

From Visual Studio 2012 on there is no Windows Installer project available any more. You can use the InstallShield Express edition with limited capabilities or the Windows Installer XML (WiX) open source package created by Rob Mensching when he was working for Microsoft. (It's actually the oldest open source project from Microsoft and now under the OuterCurve foundation)

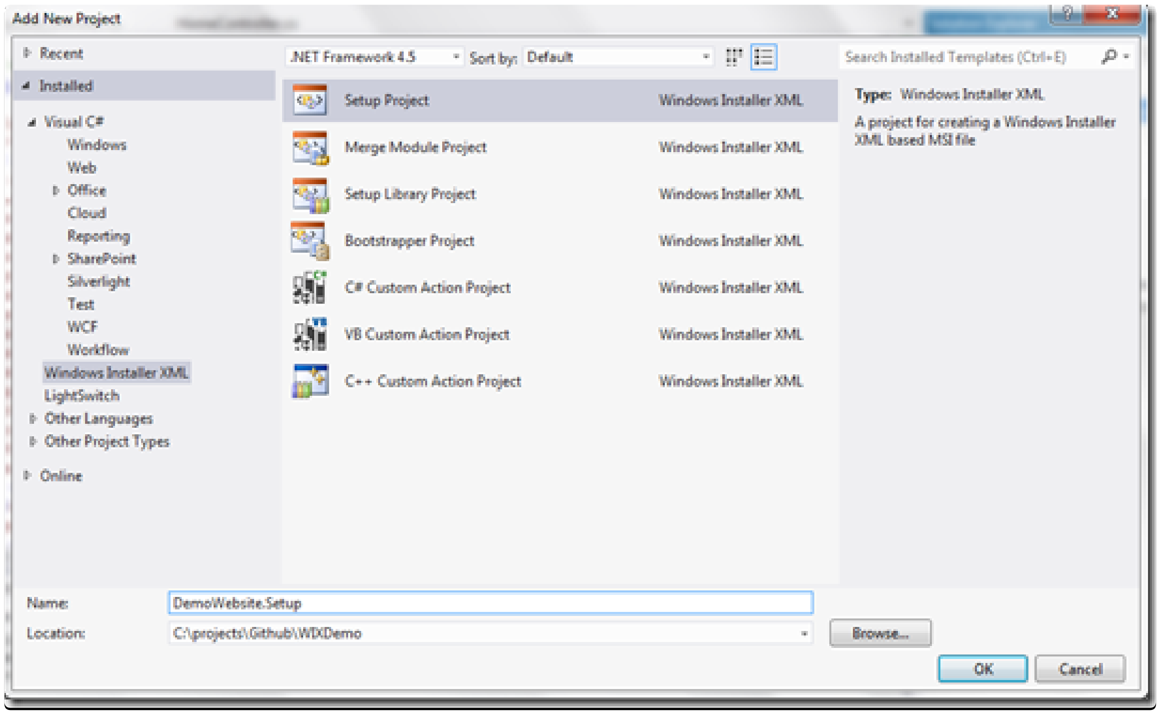
The Installshield express version doesn't support IIS installations and falls out of the boat. WiX does support all actions that we have to do but has a steep learning curve. I used WIX in a previous project and still had some hassle to put all of it together. This series of posts will walk us through the creation of the installer.

Start project

Start by downloading the WiX toolset from http://wixtoolset.org/releases/. For this demo I used the 3.8.826.0 version. This is not the latest stable published release but I haven't got any problems with this version.

Next step is creating a MVC 4 web project and choose for an internet application so we have some default files (javascript, css, views, controllers, …).

Right click on the solution in the solution explorer and choose to add another project. In the ‘Add New Project’ dialog select Windows Installer XML on the left hand side. Choose for ‘Setup Project’ and click the OK button.


Add New Project
Recent
Installed
Visual C#
Windows
Web
Office
Cloud
Reporting
SharePoint
Silverlight
Test
WCF
Workflow
Windows Installer XML
LightSwitch
Other Languages
Other Project Types
Online
.NET Framework 4.5
Sort by: Default
Search Installed Templates (Ctrl+E)
Setup Project Windows Installer XML
Merge Module Project Windows Installer XML
Setup Library Project Windows Installer XML
Bootstrapper Project Windows Installer XML
C# Custom Action Project Windows Installer XML
VB Custom Action Project Windows Installer XML
C++ Custom Action Project Windows Installer XML
Browse...
OK
Cancel

Type: Windows Installer XML

A project for creating a Windows Installer XML based MSI file

Name: DemoWebsite.Setup

Location: C:\projects\Github\WIXDemo

You should get a new project with only one file (Product.wxs).

WiX flow

WiX source files are written in XML and have the wxs or wxi (for variables) extensions. Those files have to be compiled to wixobj files. This can be done in Visual Studio or by command line by using the candle.exe tool in the WiX toolset. After compiling the wixobj files another tool is needed to create the msi (installer) file, the light.exe tool.

The most simple installer can be created by just using one wxs file. You will notice that it will make your project more clear to use different wxs files. One for every part of the installation.

You can use the default UI that is available in the WiX toolset but with bigger projects (and in this demo) you can create your own UI and flow. Even the UI is defined in XML and has the same wxs extension. Another reason to split up your installer code in different files to keep the overview.

Step 1: install all needed files

Open up the Product.wxs file in Visual Studio. You'll see already a few standard values filled out.

```
<?xml version="1.0" encoding="UTF-8"?>
<Wix xmlns="http://schemas.microsoft.com/wix/2006/wi">
  <Product Id="*" Name="Demo website setup" Language="1033" Version="1.0.0.0" Manufacturer="" Up
                <Package InstallerVersion="200" Compressed="yes" InstallScope="perMachine" />

                <MajorUpgrade DowngradeErrorMessage="A newer version of [ProductName] is already
                <MediaTemplate />

                <Feature Id="ProductFeature" Title="DemoWebsite.Setup" Level="1">
                        <ComponentGroupRef Id="ProductComponents" />
                </Feature>
        </Product>

        <Fragment>
                <Directory Id="TARGETDIR" Name="SourceDir">
                        <Directory Id="ProgramFilesFolder">
                                <Directory Id="INSTALLFOLDER" Name="DemoWebsite.Setup" />
                        </Directory>
                </Directory>
        </Fragment>

        <Fragment>
                <ComponentGroup Id="ProductComponents" Directory="INSTALLFOLDER">
                        <!-- TODO: Remove the comments around this Component element and the Com
                        <!-- <Component Id="ProductComponent"> -->
                                <!-- TODO: Insert files, registry keys, and other resources here
                        <!-- </Component> -->
                </ComponentGroup>
        </Fragment>
</Wix>
```

gistfile1.xml hosted with ❤ by **GitHub** **view raw**

At line 3 you get the product attributes that have to be set. Leave the asterix (*) at the Id tag. WiX will replace this with an unique Guid when compiling the source. Set the version, language and the name to the desired values.

IIS default location for websites is the c:\inetpub directory. We'll alter the installer so this default location is used. In one of the next chapters we'll be able to change this folder. Navigate to line 15 and alter the Directory tag.

```
  <Fragment>
    <!-- Will default to C:\ if that is the main disk-->
```

```
                <Directory Id="TARGETDIR" Name="SourceDir">
      <!-- Will reference to C:\inetpub-->
                        <Directory Id="INETPUB" Name="Inetpub">
        <!-- Will reference to c:\Inetpub\Demowebsite-->
                                <Directory Id="INSTALLFOLDER" Name="DemoWebsite" />
                        </Directory>
                </Directory>
        </Fragment>
```

gistfile1.xml hosted with ❤ by **GitHub** **view raw**

You'll see I've changed the default InstallFolder to Inetpub. (from c:\Program Files to c:\Inetpub). This is all we have to change for the install location.

On line 26 you'll see the ComponentGroup where we'll have to define all files that have to be installed in our installation folder. Let's start with adding some files. In the example below I added 3 files in the root of out application (favicon.ico, web.config and global.asax). To add the bin folder I had to add a new ComponentGroup, a new Component, and a new Directory element before I could add the files (2 dll's).

```
    <Feature Id="ProductFeature" Title="DemoWebsite.Setup" Level="1">
      <ComponentGroupRef Id="ProductComponents" />
      <ComponentGroupRef Id="BinComponents" />
    </Feature>
  </Product>

  <Fragment>
    <!-- Will default to C:\ if that is the main disk-->
    <Directory Id="TARGETDIR" Name="SourceDir">
      <!-- Will reference to C:\inetpub-->
      <Directory Id="INETPUB" Name="Inetpub">
        <!-- Will reference to c:\Inetpub\Demowebsite-->
        <Directory Id="INSTALLFOLDER" Name="DemoWebsite">
          <Directory Id="BIN" Name="bin"/>
        </Directory>
      </Directory>
    </Directory>
  </Fragment>

  <Fragment>
    <ComponentGroup Id="ProductComponents" Directory="INSTALLFOLDER">
      <Component Id="ProductComponent" Guid="{6F44232F-1C0B-4278-AB2B-BFD34FAE863C}">
        <File Id="favicon.ico" Source="..\DemoWebsite\favicon.ico" />
        <File Id="Global.asax" Source="..\DemoWebsite\Global.asax" />
        <File Id="Web.config" Source="..\DemoWebsite\Web.config" />
      </Component>
    </ComponentGroup>
    <ComponentGroup Id="BinComponents" Directory="BIN">
```

```
        <Component Id="BinComponent" Guid="{A41811D7-49DD-462B-98D2-56DF1202008E}">
          <File Id="Antlr3.Runtime.dll" Source="..\DemoWebsite\bin\Antlr3.Runtime.dll" />
          <File Id="DotNetOpenAuth.AspNet.dll" Source="..\DemoWebsite\bin\DotNetOpenAuth.AspNet.dl
          <!-- And so on and on -->
        </Component>
      </ComponentGroup>
    </Fragment>
```

gistfile1.xml hosted with ❤ by GitHub view raw

As you can see this is a tedious job to add every file you want to be installed. Luckily there is a faster way to create this.

Use the heat component from the WiX toolset

The WiX toolset has another tool heat.exe that can help us to harvest all files that we need to install. Although heat was incorporated in Votive (the Visual Studio environment for WiX) in earlier versions, in the 3.7 – 3.8 version this is not available in Visual Studio.

MSBuild to the rescue

If we want to make use of the heat component we'll have to script it. We can create a bat file we can run every time before we build the installer or we can create an MS build script that we can run. The MS build script has the advantage that we can reuse this script for out build server (continuous integration).

Create a new text file in the Setup project and rename it to setup.build. First we'll add some properties in a 'PropertyGroup': the source of our website and the name of the WiX file we want to build. We also include the path where we should publish all files. In the 'itemgroups' we define the temporary files witch is the content of the web site and the list of WiX input files.

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="3.5" DefaultTargets="Build"
       xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <WebSiteSource>..\DemoWebsite\</WebSiteSource>
    <SetupF>..\Setup\</SetupF>
    <PublishF>publish\</PublishF>
    <Publish>$(SetupF)$(PublishF)</Publish>
    <WebSiteContentCode>WebSiteContent.wxs</WebSiteContentCode>
  </PropertyGroup>
  <!-- Defining group of temporary files which is the content of the web site. -->
  <ItemGroup>
    <WebSiteContent Include="$(WebSiteContentCode)" />
  </ItemGroup>

  <!-- The list of WIX input files -->
  <ItemGroup>
    <WixCode Include="Product.wxs" />
    <WixCode Include="$(WebSiteContentCode)" />
  </ItemGroup>
</Project>
```

gistfile1.xml hosted with ❤ by **GitHub** **view raw**

Add build target

We first have to build our website so we are sure we have the latest build we are deploying. Therefor we add a target in the MS build file

```
<Target Name="Build">
  <!-- Compile whole solution in release mode -->
  <MSBuild
      Projects="..\DemoWebsite.sln"
      Targets="ReBuild"
      Properties="Configuration=Release" />
</Target>
```

gistfile1.xml hosted with ❤ by **GitHub** **view raw**

Add Publish website target

We'll use the build in publish feature of MS build to deploy the website to a new folder so we have only the files we need. (and not the .cs files etc)

```
<Target Name="PublishWebsite">
  <!-- Remove complete publish folder in order to
           be sure that evrything will be newly compiled -->
  <Message Text="Removing publish directory: $(SetupF)"/>
  <RemoveDir Directories="$(SetupF)" ContinueOnError="false" />
  <Message Text="Start to publish website" Importance="high" />
  <MSBuild
      Projects="..\\DemoWebsite\DemoWebsite.csproj"
      Targets="ResolveReferences;_CopyWebApplication"
      Properties="OutDir=$(Publish)bin\;WebProjectOutputDir=
                                   $(Publish);Configuration=Release" />
   </Target>
```

gistfile1.xml hosted with ❤ by **GitHub** **view raw**

Harvest the files in WiX

Now that we have all the files we need under a temporary folder we can use the heat.exe tool in the WiX tool belt to harvest the files and create a wxs file.

```
  <Target Name="Harvest">
    <!-- Harvest all content of published result -->
    <Exec
        Command='$(WixPath)heat dir $(Publish) -dr INSTALLFOLDER -ke -srd -cg MyWebWebComponents
        ContinueOnError="false"
        WorkingDirectory="." />
  </Target>
```

gistfile1.xml hosted with ❤ by **GitHub** **view raw**

The parameters used in this command:

- dir $(Publish) tells to harvest a directory (our published website)
- -dr: The directory where the files have to be installed to
- -ke: Keep the Empty directories
- -srd: Suppress harvesting the root directory as an element.
- -cg: The ComponentGroup name that have to be used
- -var var.publishDir: Will substitute the source directory with a wix variable so we can use $(var.publishDir)\myfile.txt in the wxs files
- -out $(WebsiteContentCode) the file we want to be created (see PropertyGroup settings)

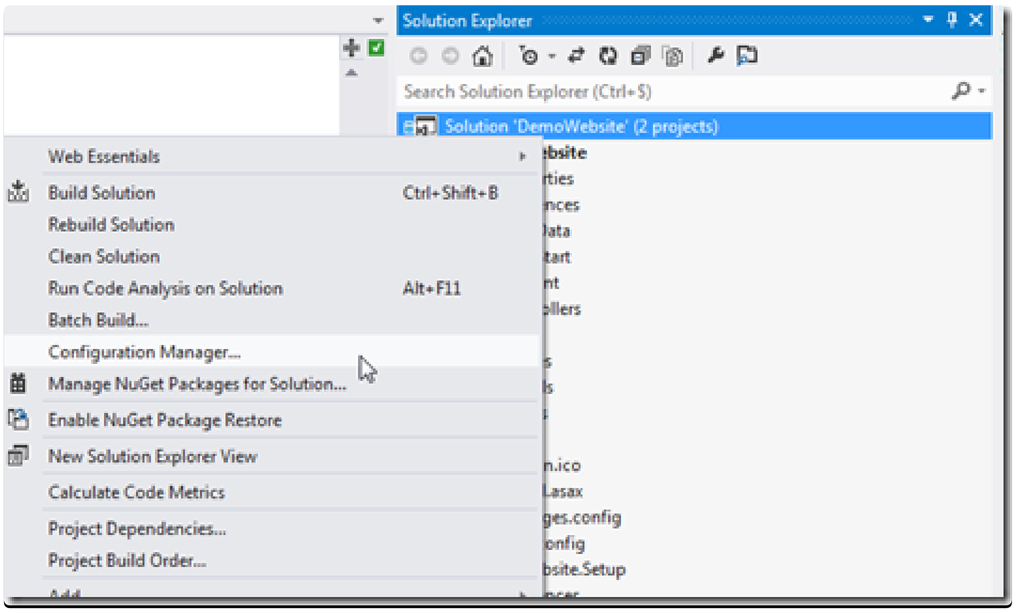
Test the script

With the heat command inserted we can test our script. Open up a Developer Command Prompt for VS2012. Change the prompt to the DemoWebsite.Setup project folder and type the following command:

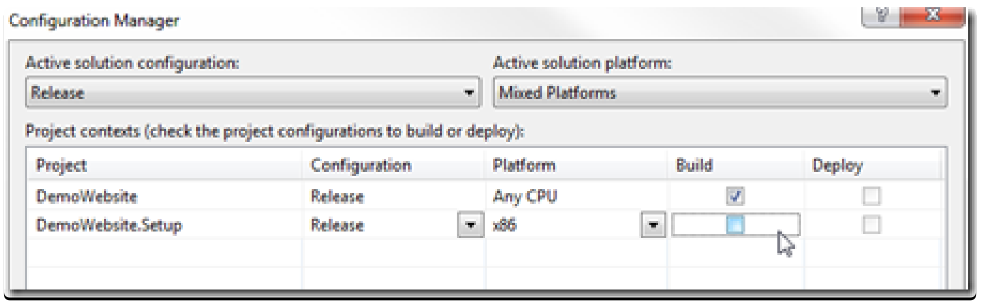
msbuild /t: Build;PublishWebsite;Harvest setup.build and hit enter. If everything goes well you'll see a lot of command coming by. The script will create a folder Setup\publish under the root and publish the website. At last a WebsiteContent.wxs file will be created in the setup project folder.

If you open up the WebsiteContent.wxs file you'll see all files and folders are added with their own Id under a ComponentGroup MyWebComponents.

If you looked closely you'll have seen a few WiX commands passing by when executing the build file. Because we are going to handle the WiX build process in our build file we can exclude the setup project from the build configuration. Right click on the solution in the Solution Explorer in Visual Studio and choose for Configuration manager.


Solution Explorer
Search Solution Explorer (Ctrl+S)
Solution 'DemoWebsite' (2 projects)
Web Essentials
Build Solution Ctrl+Shift+B
Rebuild Solution
Clean Solution
Run Code Analysis on Solution Alt+F11
Batch Build...
Configuration Manager...
Manage NuGet Packages for Solution...
Enable NuGet Package Restore
New Solution Explorer View
Calculate Code Metrics
Project Dependencies...
Project Build Order...

Change the active solution configuration to 'Release' and uncheck the build flag next to the setup project.


Configuration Manager

Active solution configuration: Release

Active solution platform: Mixed Platforms

Project contexts (check the project configurations to build or deploy):

Project	Configuration	Platform	Build	Deploy
DemoWebsite	Release	Any CPU	✓	
DemoWebsite.Setup	Release	x86		

Update the Product.wxs file

Now we have all our files we have to install we have to reference to the created MyWebComponents

CompenentGroup and delete the entries we made before to add the files.

```
<?xml version="1.0" encoding="UTF-8"?>
<Wix xmlns="http://schemas.microsoft.com/wix/2006/wi">
  <Product Id="*" Name="Demo website setup" Language="1033" Version="1.0.0.0" Manufacturer="Me"
    <Package InstallerVersion="200" Compressed="yes" InstallScope="perMachine" />

    <MajorUpgrade DowngradeErrorMessage="A newer version of [ProductName] is already installed.'
    <MediaTemplate />

    <Feature Id="ProductFeature" Title="DemoWebsite.Setup" Level="1">
      <ComponentGroupRef Id="MyWebWebComponents" />
    </Feature>
  </Product>

  <Fragment>
    <!-- Will default to C:\ if that is the main disk-->
    <Directory Id="TARGETDIR" Name="SourceDir">
      <!-- Will reference to C:\inetpub-->
      <Directory Id="INETPUB" Name="Inetpub">
        <!-- Will reference to c:\Inetpub\Demowebsite-->
        <Directory Id="INSTALLFOLDER" Name="DemoWebsite">
        </Directory>
      </Directory>
    </Directory>
  </Fragment>
</Wix>
```

gistfile1.xml hosted with ❤ by **GitHub** **view raw**

Build the installer

Now that we have all the components for the first fase (installing the files) we can use the candle.exe and light.exe tools from the WiX tool belt to built our installer.

Add properties in the build file

First we need some more properties in our build file. Add the WebSiteContentObject parameter that will hold the compiled WiX code (WebSiteContent.wixobj). And also add the MsiOut parameter that will hold the path and name of the installer (.msi) file.

```
<PropertyGroup>
```

```
    <WebSiteSource>..\DemoWebsite\</WebSiteSource>
    <SetupF>..\Setup\</SetupF>
    <PublishF>publish\</PublishF>
    <Publish>$(SetupF)$(PublishF)</Publish>
    <WebSiteContentCode>WebSiteContent.wxs</WebSiteContentCode>
    <WebSiteContentObject>WebSiteContent.wixobj</WebSiteContentObject>
    <MsiOut>bin\Release\DemoWebsite_Setup.msi</MsiOut>
  </PropertyGroup>
```

gistfile1.xml hosted with ❤ by **GitHub** **view raw**

Add candle.exe in the build file

Add a new target tag in the build file and add the candle.exe tool with the parameters where to find the publish directory and witch files he has to compile.

```
  <Target Name="WIX">
    <Exec
        Command='"$(WixPath)candle" -dpublishDir=$(Publish) -dMyWebResourceDir=. @(WixCode, &apos;
        ContinueOnError="false"
        WorkingDirectory="." />
  </Target>
```

gistfile1.xml hosted with ❤ by **GitHub** **view raw**

Add light.exe in the build file

In the same target (WIX) add the light.exe command with parameters where to put the generated msi and witch source files to include.

```
  <Target Name="WIX">
    <!--     At last create an installer -->
    <Message Text="TEST: @(WixCode)"/>
    <Exec
        Command='"$(WixPath)candle" -dpublishDir=$(Publish) -dMyWebResourceDir=. @(WixCode, &apos;
        ContinueOnError="false"
        WorkingDirectory="." />
    <Exec
        Command='"$(WixPath)light" -out $(MsiOut) @(WixObject, &apos; &apos;)'
        ContinueOnError="false"
        WorkingDirectory="." />

    <!-- A message at the end -->
    <Message Text="Install package has been created." />
  </Target>
```

gistfile1.xml hosted with ❤ by GitHub view raw

Final run

Open up your Developer Command Prompt and type the next command where we added the WIX target:

msbuild /t: Build;PublishWebsite;Harvest;WIX setup.build

Hit enter and keep your fingers crossed. If you had no error messages you should find a msi file in the bin/release folder of the setup project. Run that installer and you'll see that under the C:\inetpub folder a DemoWebsite folder is added with all the published files from our webapplication.

If you had any errors you can find the complete files here:

- The build file
- The Product.wxs file

Next parts

Enough for one blog post I would say. the next posts in this series will handle the next actions:

- Install the .NET 4.5 framework if that isn't installed already
- Install the MVC 4 framework if that isn't installed already.
- Create a folder and copy all needed files to run the application (done)
- Create a new database on an existing SQL server and prefill the database with the correct tables and values. (the connection details and database name should be entered by the end user running the installer)
- Create a new website in IIS 7.5 (create website and application pool running under .NET 4.5)
- Alter the config file so the correct connection settings are used (entered by the end user)

Complete source code

You can find the complete source code for this project on GitHub. Keep in mind that this project will be altered when the next parts are implemented. I will try to keep the commits together with the series.

Other posts in this series

- Create an installer for website with WIX, part 1 (install files, create MS Build script)
- Create an installer for website with WIX, part 2 (Custom UI)
- Create an installer for website with WIX, part 3 (Install website in IIS)
- Create an installer for website with WIX, part 4 (Create database and run scripts)