A. Leonard, *Building Custom Tasks for SQL Server Integration Services*
https://doi.org/10.1007/978-1-4842-6482-9_22

22. Building the Setup Project

Andy Leonard[1]
(1) Farmville, VA, USA

Setup projects, or installer projects, allow developers to manage distribution of code in an enterprise and to share code with other enterprises. SSIS has long promoted a healthy community of third-party task and component developers. In this chapter, we add a setup project to the ExecuteCatalogPackageTask solution and configure the setup project to install the Execute Catalog Package Task on other servers using the following steps:

- Add the ExecuteCatalogPackageTaskSetup project.
- Add References.
- Configure the Product tag.
  - Configure the Package tag.
  - Configure the Installation Path Property tags.
  - Configure the MajorUpgrade and MediaTemplate tags.
  - Initialize the UIRef and License file tags.
  - Configure the Installation feature and icon.
- Configure folders and folder structure.
  - Configure the ExecuteCatalogPackageTask GAC registration.
  - Configure the ExecuteCatalogPackageTask Tasks deployment folder.
  - Configure the ExecuteCatalogPackageTaskComplexUI GAC registration.
  - Configure the ExecuteCatalogPackageTaskComplexUI Tasks deployment folder.

Installer projects create a file named setup.exe or <*project name*>.msi. A Windows application named MsiExec.exe executes msi files, which installs the desired software. Visual Studio supports extensions that surface installer projects. One extension is Windows Installer XML, WiX. Visit the WiX Toolset page (wixtoolset.org) to download and install the Visual Studio project template we will use to build the ExecuteCatalogPackageTask.msi file.

To follow the demonstrations and samples included in this chapter, the reader *must* visit wixtoolset.org, download the latest version of the software, and follow

the WiX Toolset installation instructions found at wixtoolset.org.

This chapter merely lists the XML required to configure an msi installer file for the Execute Catalog Package Task without providing much explanation. The reader is encouraged to learn more about using the WiX Toolset at wixtoolset.org where one may find in-depth documentation and examples at wixtoolset.org/documentation.

Adding the ExecuteCatalogPackageTaskSetup Project

To add an installer project to the ExecuteCatalogPackageTask solution, right-click the solution in Solution Explorer, hover over Add, and then click New Project, as shown in Figure 22-1:

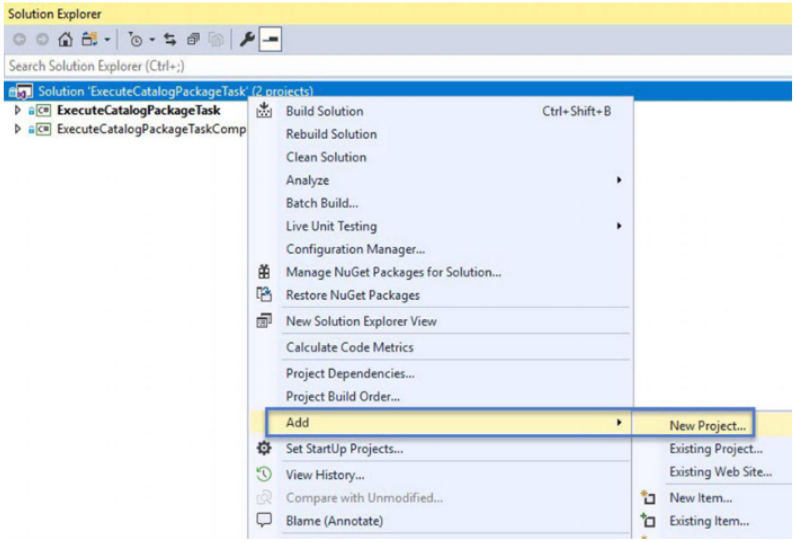
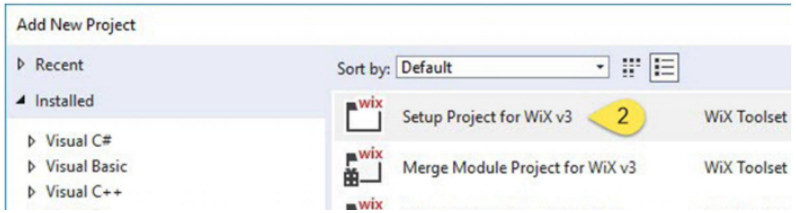


Figure 22-1 Adding a new setup project

When the Add New Project dialog displays, navigate to the WiX Toolset template category and select the latest version. Select Setup Project for WiX in the available templates list, and then name the new project "ExecuteCatalogPackageTaskSetup," as shown in Figure 22-2:



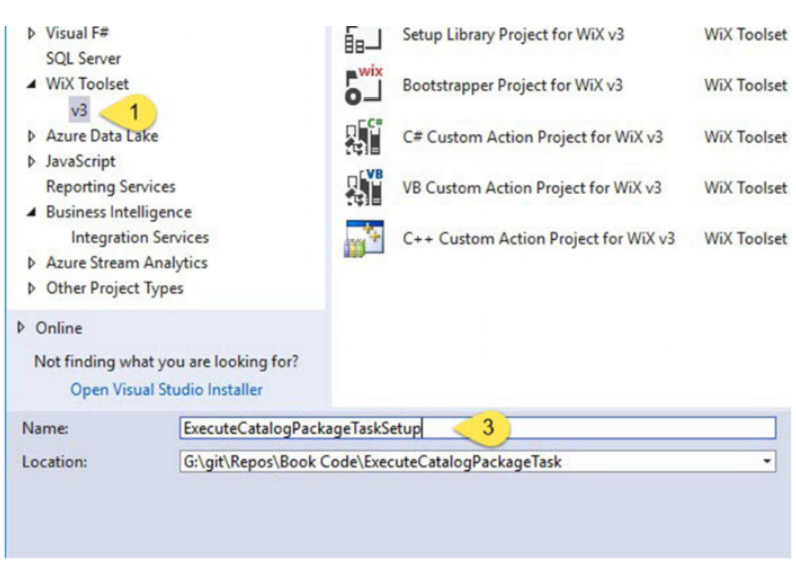


Figure 22-2 Adding the ExecuteCatalogPackageTaskSetup project

Once the new project has been added to the ExecuteCatalogPackageTask solution, Solution Explorer appears as shown in Figure 22-3:

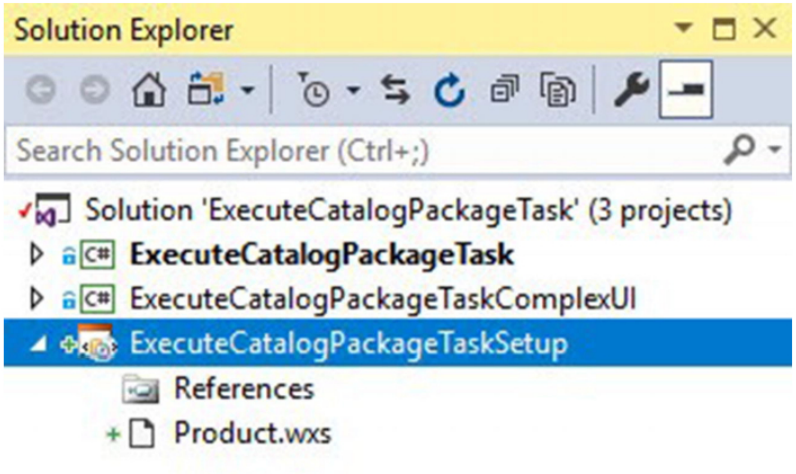
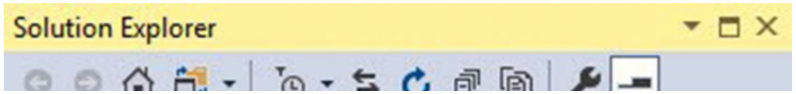


Figure 22-3 The ExecuteCatalogPackageTaskSetup project

Rename the ExecuteCatalogPackageTask.wxs file "ExecuteCatalogPackageTask.wxs," as shown in Figure 22-4:



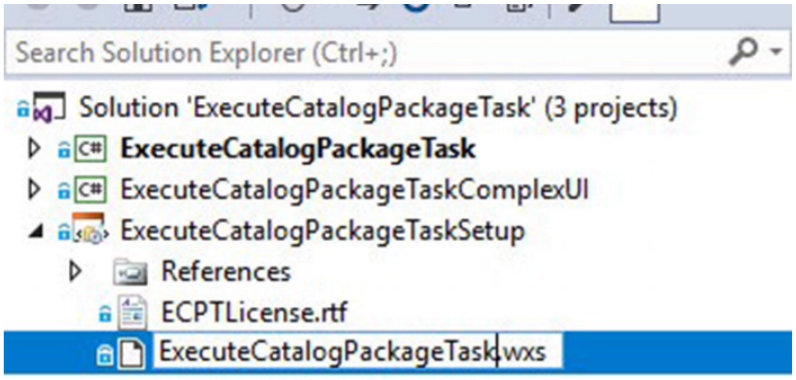


Figure 22-4 Renaming the ExecuteCatalogPackageTask.wxs file ExecuteCatalogPackageTask.wxs

The next step is to add references.

Adding References

In Solution Explorer, right-click the References virtual folder and then click Add References, as shown in Figure 22-5:

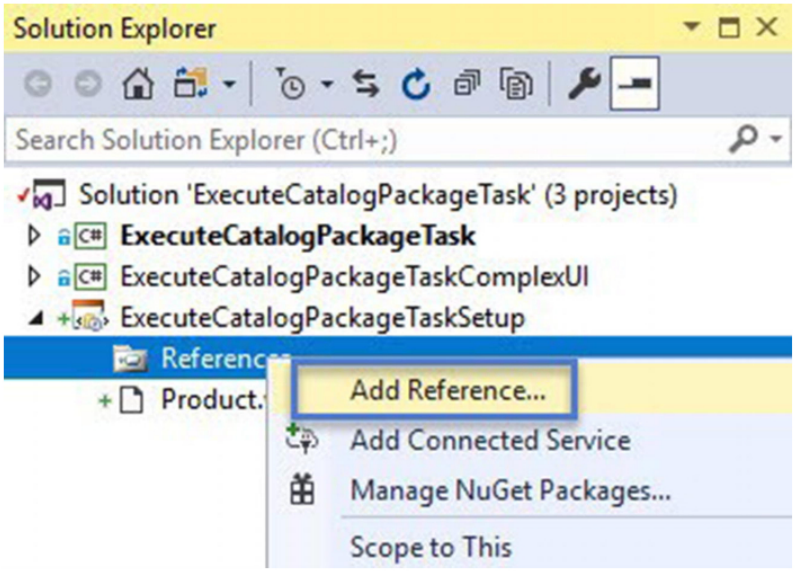


Figure 22-5 Adding References

When the WiX Add Reference dialog displays, click the Projects tab, and then select – and add – the ExecuteCatalogPackageTask and ExecuteCatalogPackageTaskComplexUI projects, as shown in Figure 22-6:

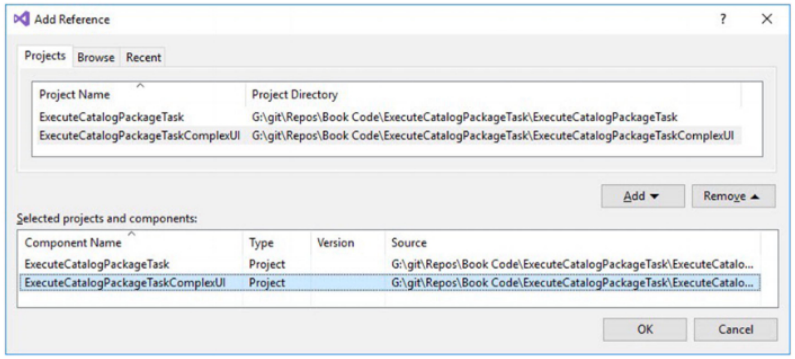


Figure 22-6 Adding the ExecuteCatalogPackageTask and ExecuteCatalogPackageTaskComplexUI projects

Click the OK button to add the projects and close the WiX Add Reference dialog.

Next, add references to the WixUIExtension and WixUtilExtension assemblies, which will be located in the WiX Toolset installation directory (on my machine, these files are located in the C:\Program Files (x86)\WiX Toolset v3.11\bin folder), as shown in Figure 22-7:

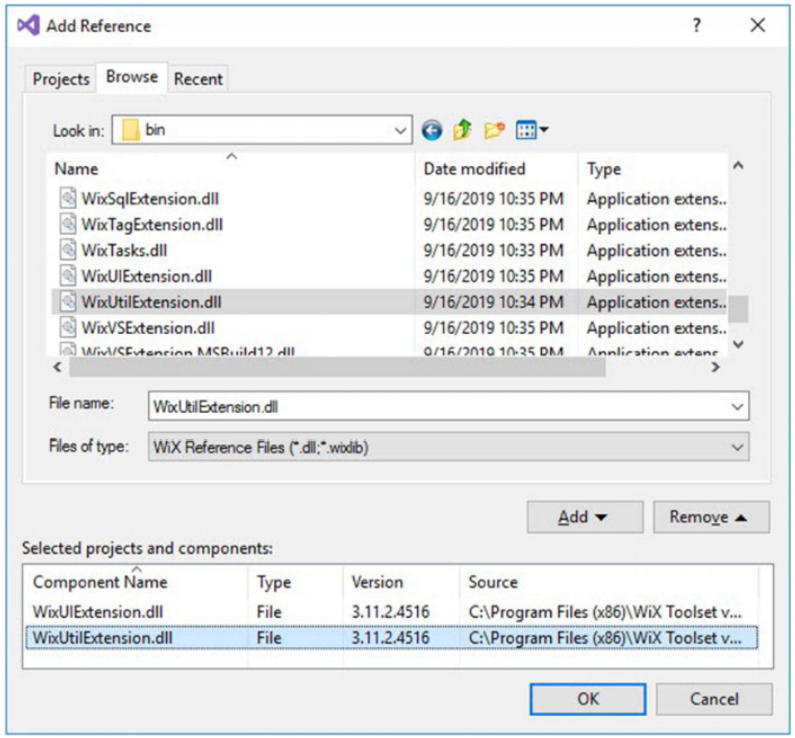


Figure 22-7 Adding the WixUIExtension and WixUtilExtension assemblies

The previous two steps are separated for the sake of clarification, but they can be easily combined into a single step where the WiX assemblies *and* projects are added together, as shown in Figure 22-8:

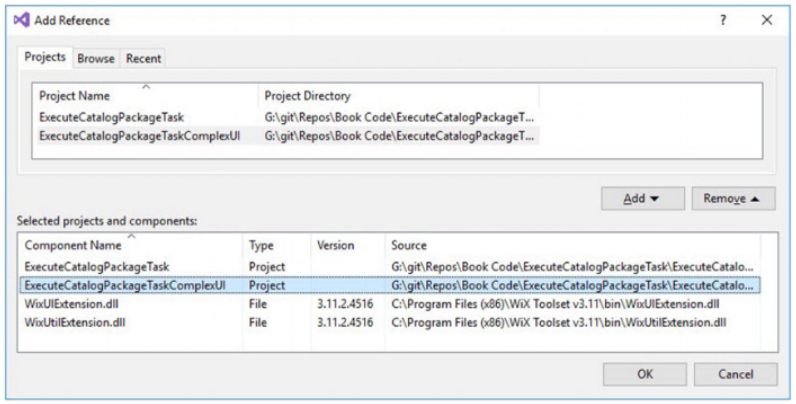


Figure 22-8 Adding WiX assemblies and Projects in one step

The next step is to configure the ExecuteCatalogPackageTask.wxs Product tag.

Configuring Tags

In this section, we configure WiX tags to build the installation project to install the Execute Catalog Package Task on Windows Server 2016 running SQL Server 2017.

Configure the Product tag

In Solution Explorer, double-click the ExecuteCatalogPackageTask.wxs file to open the file in the Visual Studio editor, as shown in Figure 22-9:

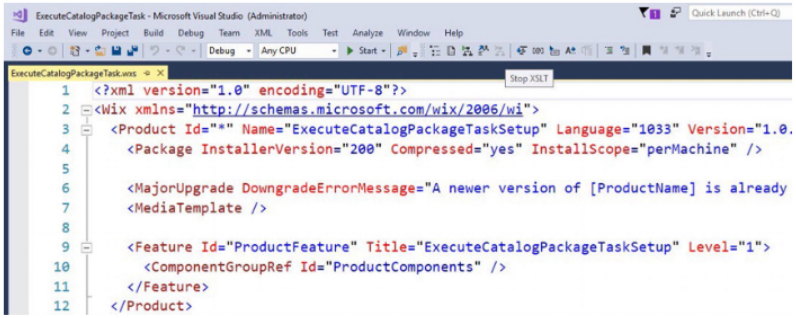


Figure 22-9 The ExecuteCatalogPackageTask.wxs file

The WiX Toolset process uses the settings in the ExecuteCatalogPackageTask.wxs file to configure the msi installer project and includes a helpful template by default. The next step is to edit the template, where needed, to install the Execute Catalog Package Task.

The ExecuteCatalogPackageTask.wxs file begins with an XML declaration shown in Figure 22-10:

```
<?xml version="1.0" encoding="UTF-8"?>
<Wix xmlns="http://schemas.microsoft.com/wix/2006/wi">
```

Figure 22-10 The xml tag

The next tag, the `Wix` tag, encapsulates the remainder of the ExecuteCatalogPackageTask.wxs file, as shown in Figure 22-11:

```
<Wix xmlns="http://schemas.microsoft.com/wix/2006/wi">
  <Product Id="*" Name="ExecuteCatalogPackageTaskSetup" Language="1033" Version="1.0.0.0" Manufacturer="">
    <Package InstallerVersion="200" Compressed="yes" InstallScope="perMachine" />

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

    <Feature Id="ProductFeature" Title="ExecuteCatalogPackageTaskSetup" Level="1">
      <ComponentGroupRef Id="ProductComponents" />
    </Feature>
  </Product>

  <Fragment>
    <Directory Id="TARGETDIR" Name="SourceDir">
      <Directory Id="ProgramFilesFolder">
        <Directory Id="INSTALLFOLDER" Name="ExecuteCatalogPackageTaskSetup" />
      </Directory>
    </Directory>
  </Fragment>

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

Figure 22-11 The Wix tag

The next step is to configure the `Product` tag. Begin by placing `Product` tag attributes on different lines. The `Product Id` value is configured to "*" which indicates the value will be automatically generated. Update the `Product Name` attribute value to "Execute Catalog Package Task," the `Product Version` attribute value, and the `Product Manufacturer` attribute value, if desired. I recommend leaving the `Product Language` and `UpgradeCode` attribute values set to their default values. Since XML ignores whitespace, feel free to add space to the ExecuteCatalogPackageTask.wxs file, as shown in Figure 22-12:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<Wix xmlns="http://schemas.microsoft.com/wix/2006/wi">

  <Product Id="*"
           Name="Execute Catalog Package Task"
           Language="1033"
           Version="1.0.0.0"
           Manufacturer="Andy Leonard Training, Inc."
           UpgradeCode="98d780e5-59db-4f86-b709-47db487643f4">
```

Figure 22-12 Updating the Product tag

Configure the Package Tag

The installation will require administrative permissions to search the registry, install to the Program Files (x86) path, and register the assemblies in the Global Assembly Cache (GAC) on the target server. Configure the `Package` tag by adding two attributes – `InstallPrivileges` and `AdminImage` – to permit installation as an administrator using the XML in Listing 22-1:

```
InstallPrivileges="elevated"
AdminImage="yes"
```

Listing 22-1 Permit Administrator installation

Once added, the XML appears as shown in Figure 22-13:

```
<Package InstallerVersion="200"
         Compressed="yes"
         InstallScope="perMachine"
         InstallPrivileges="elevated"
         AdminImage="yes" />
```

Figure 22-13 Package edited to permit administrative installation privileges

Configure the Installation Path Property Tags

SSIS task assemblies are installed in the <*drive*>:\ Program Files (x86)\Microsoft SQL Server\<*version*>\DTS\Tasks folder for use during development. SSIS task assemblies are registered in the Global Assembly Cache or GAC for use during execution.

The next step is to add a property for the SSIS Task Installation Folder using the XML in Listing 22-2:

```
<!-- Add a property for the custom task installation folder -->
<Property Id="SSISTASKINSTALLFOLDER" Value="SSISINSTALLFOLDER" />
```

Listing 22-2 Add the SSIS Task Installation Folder property

When added, the XML appears as shown in Figure 22-14:

```
<!-- Add a property for the custom task installation folder -->
<Property Id="SSISTASKINSTALLFOLDER" Value="SSISINSTALLFOLDER" />
```

Figure 22-14 Adding the SSIS Task Installation Folder property

The next step is to add the SSIS Folder property using the XML in Listing 22-3:

```
<!-- Add a property for the SQL Server folder -->
<Property Id="SSISFOLDER" Value="C:\Program Files (x86)\Microsoft SQL Se
```

Listing 22-3 Add the SSIS Folder property

When added, the XML appears as shown in Figure 22-15:

```
20 <!-- Add a property for the SQL Server folder -->
21 <Property Id="SSISFOLDER" Value="C:\Program Files (x86)\Microsoft SQL Server\140">
```

Figure 22-15 Adding the SSIS Folder property

On line 21, the property is configured by setting the Id attribute value and supplying a default Value property value.

The next step is to add the SSIS Folder property. This step poses the highest challenge in configuring the installation project because it relies on the Windows Registry, and the registry for different Windows operating systems and different versions of SQL Server stores values in different locations. How and where you isolate registry values – and whether the values you find are *accurate*, even – may be a matter of trial and error.

For SQL Server 2017 running on Windows Server 2016, configure the registry-search XML using the XML in Listing 22-4:

```
<!-- Search the registry for 32-bit and 64-bit SQL Server folder locatio
 -->
  <!-- By far, this is the trickiest part of the WiX configuration,
        trying to find a registry value that is accurate and works.
        The HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Microsoft\Microsoft
        entry works on Windows Server 2016. Yes, it's different for othe
       <RegistrySearch Id="SSISFOLDER_REG_32"
```

```
                        Type="directory"
                        Key="SOFTWARE\WOW6432Node\Microsoft\Microsoft SQL
                        Name="VerSpecificRootDir"
                        Root="HKLM" Win64="no" />
</Property>
```

Listing 22-4 Add the RegistrySearch XML

When added, the XML appears as shown in Figure 22-16:

```
20 <!-- Add a property for the SQL Server folder -->
21 <Property Id="SSISFOLDER" Value="C:\Program Files (x86)\Microsoft SQL Server\140">
22   <!-- Search the registry for 32-bit and 64-bit SQL Server folder locations -->
23   <!-- By far, this is the trickiest part of the WiX configuration,
24        trying to find a registry value that is accurate and works.
25        The HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Microsoft\Microsoft SQL Server\140\VerSpecificRootDir
26        entry works on Windows Server 2016. Yes, it's different for other OS's. -->
27   <RegistrySearch Id="SSISFOLDER_REG_32"
28                   Type="directory"
29                   Key="SOFTWARE\WOW6432Node\Microsoft\Microsoft SQL Server\140"
30                   Name="VerSpecificRootDir"
31                   Root="HKLM" Win64="no" />
32 </Property>
```

Figure 22-16 Adding the RegistrySearch XML

On lines 27–31, the `HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Microsoft\Microsoft SQL Server\140\VerSpecificRootDir` registry key is read and used to set the property Value attribute with the key value, which is shown in Figure 22-17:

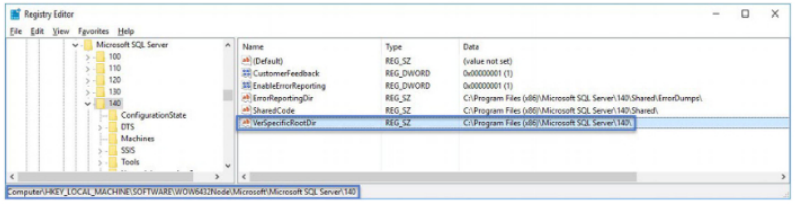


Figure 22-17 Reading the HKEY_LOCAL_MACHINE\SOFTWARE\WOW6432Node\Microsoft\Microsoft SQL Server\140\VerSpecificRootDir registry key value

Please note the Root attribute value on line 27 is "`HKLM`." `HKLM` is an abbreviation for `HKEY_LOCAL_MACHINE`.

The next step is to set the SSIS installation directory by combining the previous properties to complete the value of the SSISINSTALLFOLDER property value using the XML in Listing 22-5:

```
<!-- Set the SSISINSTALLFOLDER to the WiX UI Install Directory -->
<SetDirectory Id="SSISINSTALLFOLDER" Sequence="first" Value="[SSISFOLDER
```

Listing 22-5 Complete the SSISINSTALLFOLDER value configuration

When added, the XML appears as shown in Figure 22-18:

```
<!-- Set the SSISINSTALLFOLDER to the WiX UI Install Directory -->
<SetDirectory Id="SSISINSTALLFOLDER" Sequence="first" Value="[SSISFOLDER]\DTS\Tasks" />
```

Figure 22-18 Completing the SSISINSTALLFOLDER value configuration

The SSISINSTALLFOLDER value is now configured.

Configure the MajorUpgrade and MediaTemplate Tags

The next step is configuring the MajorUpgrade and MediaTemplate tags using the XML in Listing 22-6:

```
<!-- Configure a response to newer version detected -->
<MajorUpgrade DowngradeErrorMessage="A newer version of [ProductName] is
<MediaTemplate EmbedCab="yes" CompressionLevel="high" />
```

Listing 22-6 Configure the MajorUpgrade and MediaTemplate tags

The MajorUpgrade tag contains a message (found in the `DowngradeErrorMessage` attribute) to display when a newer version of the Execute Catalog Package Task is already installed. The `MediaTemplate` tag attributes are `EmbedCab` and `CompressionLevel`. `EmbedCab` controls whether the *cabinet*, or cab, file, which contains the Execute Catalog Package Task installation code, is embedded in the product (msi) file. `CompressionLevel` sets compression for the cab file.

When added, the XML appears as shown in Figure 22-19:

```
<!-- Configure a response to newer version detected -->
<MajorUpgrade DowngradeErrorMessage="A newer version of [ProductName] is already installed." />
<MediaTemplate EmbedCab="yes" CompressionLevel="high" />
```

Figure 22-19 Configuring the MajorUpgrade and MediaTemplate tags

The next step is: Initialize the `UIRef` and `License` file tags.

Initialize the UIRef and License File Tags

Initialize the `UIRef` and `License` file tags using the XML in Listing 22-7:

```
<!-- Initialize UI -->
<UIRef Id="WixUI_Minimal"/>
```

```
<WixVariable Id="WixUILicenseRtf" Value="ECPTLicense.rtf" />
```

Listing 22-7 Initializing the UIRef and License tags

When added, the XML appears as shown in Figure 22-20:

```
<!-- Initialize UI -->
<UIRef Id="WixUI_Minimal"/>
<WixVariable Id="WixUILicenseRtf" Value="ECPTLicense.rtf" />
```

Figure 22-20 Adding and initializing UIRef and WixUILicenseRtf WixVariable

Software licensing is above my pay grade, so I used the tools available at creativecommons.org, as shown in Figure 22-21:

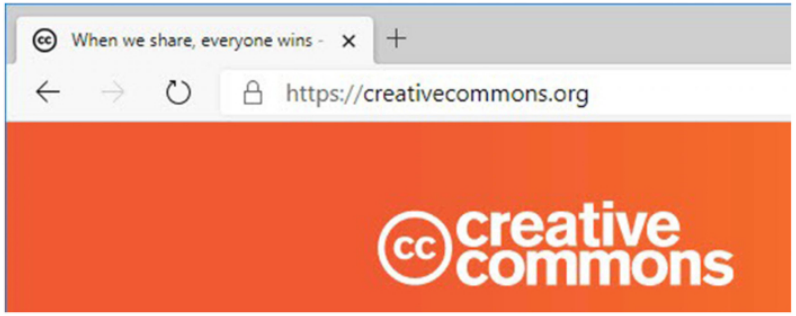


Figure 22-21 Creativecommons.org

At the time of this writing, the Creative Commons website presents a step-based page named Chooser (currently in beta at chooser-beta.creativecommons.org) where users answer questions and the Chooser suggests a license, as shown in Figure 22-22:

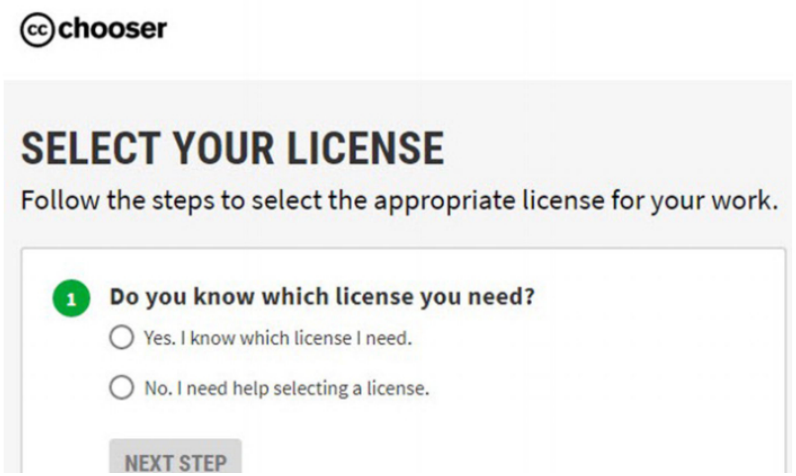


Figure 22-22 Chooser

Once the questions are complete, copy the rich text format documentation from the website, as shown in Figure 22-23:

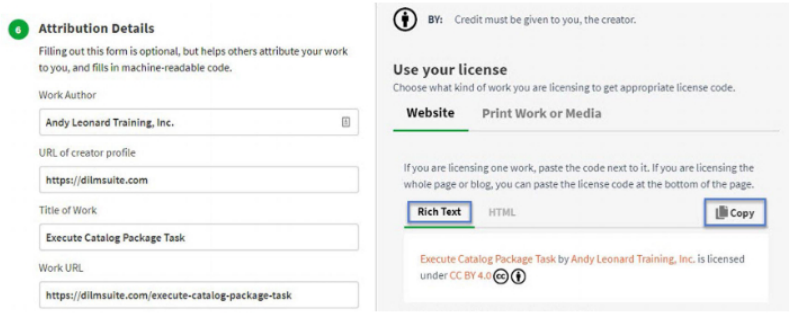


Figure 22-23 Copying the license rich text format

Once copied, open WordPad (or another rich text format editor) and paste the contents of the clipboard, as shown in Figure 22-24:

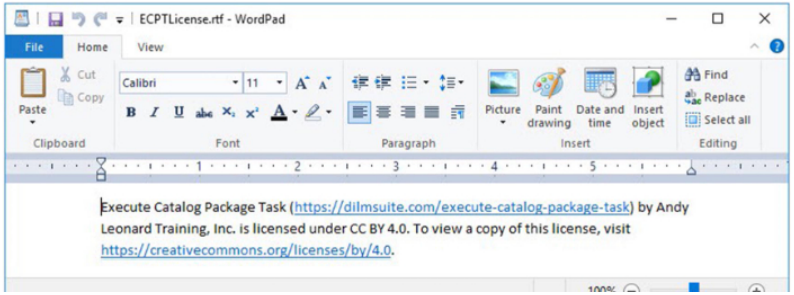


Figure 22-24 License RTF

Save the license RTF file and then add it to the setup project by right-clicking the ExecuteCatalogPackageTaskSetup project, hovering over Add, and then clicking Existing Item, as shown in Figure 22-25:

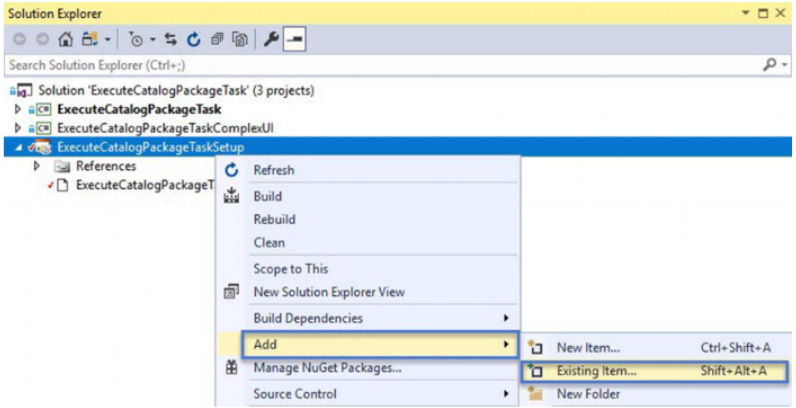


Figure 22-25 Adding the license file to the setup project

Navigate to and select the license RTF file, as shown in Figure 22-26:

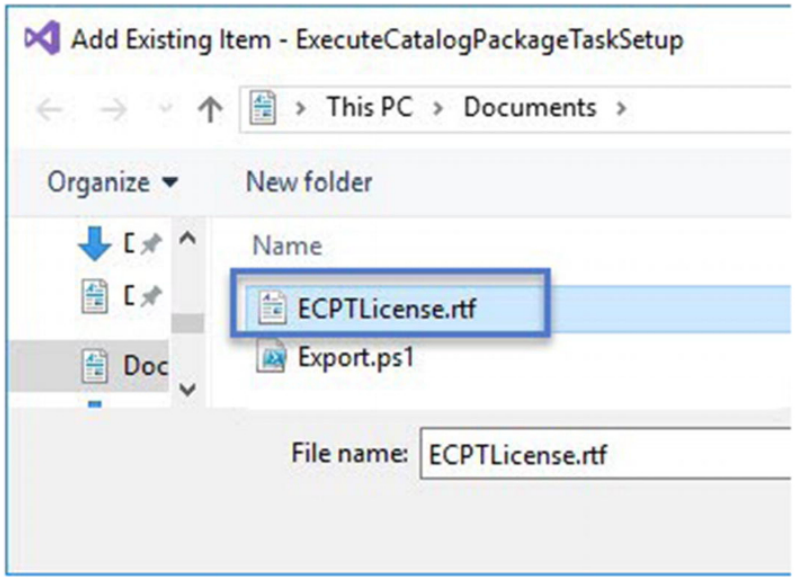


Figure 22-26 Selecting the license RTF file

Click the Add button to add the license RTF file to the ExecuteCatalogPackageTaskSetup

Click the Add button to add the license RTF file to the ExecuteCatalogPackageTaskSetup project, as shown in Figure 22-27:

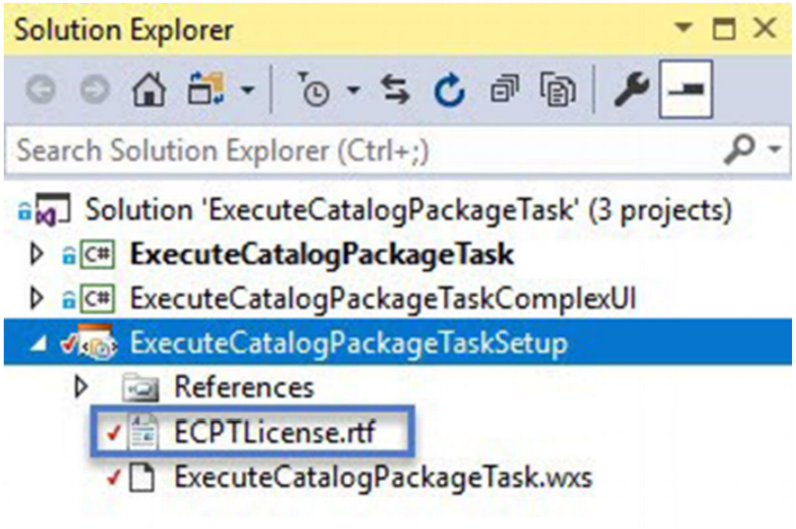


Figure 22-27 Adding the license RTF file to the ExecuteCatalogPackageTaskSetup project

The next step is configuring the installation feature and icon.

Configure the Installation Feature and Icon

The `Feature` tag identifies the ExecuteCatalogPackageTask as the "unit of installation" in the `Title` attribute.

The icon need not be imported into the WiX project because the `SourceFile` tag identifies the path to the icon file. The `ARPPRODUCTICON` property is used to set the icon to the installation file. If you want to use the ALStrike.ico, you may; it's part of the ExecuteCatalogPackageTask Visual Studio project.

Configure the installation `Feature` and `Icon` using the XML in Listing 22-8:

```
  <!-- Configure the unit of installation (feature) -->
  <Feature Id="CustomTask" Title="ExecuteCatalogPackageTask" Level="1">
    <ComponentGroupRef Id="CustomSSISTask" />
  </Feature>
  <!-- Configure WixUI icon -->
  <Icon Id="ALCstrike.ico" SourceFile="G:\git\Repos\Book Code\ExecuteCat
  <Property Id="ARPPRODUCTICON" Value="ALCstrike.ico" />
</Product>
```

Listing 22-8 Configure the installation feature and icon

When added, the XML appears as shown in Figure 22-28:

Figure 22-28 Configuring the installation feature and icon

Configure Folders and Folder Structure

In this section of the ExecuteCatalogPackageTask.wxs installation file configuration, we configure two paths for each assembly:

- The Tasks folder
- The Global Assembly Cache, or GAC

As stated earlier, Visual Studio uses assemblies deployed to the Tasks folder to populate the SSIS Toolbox for development. The SSIS execution engine uses the assemblies registered with the GAC for execution.

Configure the folders and folder structure fragment using the XML in Listing 22-9:

```
<Fragment>
  <Directory Id="TARGETDIR" Name="SourceDir">
    <Directory Id="GAC" Name="GAC" />
    <Directory Id="SSISINSTALLFOLDER" />
  </Directory>
</Fragment>
```

Listing 22-9 Configure the folders and folder structure fragment

When added, the XML appears as shown in Figure 22-29:

Figure 22-29 Configuring the folders and folder structure fragment

In Figure 22-29, the `Fragment` tag spans lines 51–57. The outermost `Directory` tag opens on line 53 and closes on line 56. The outermost `Directory` tag's `Id` attribute value is `TARGETDIR` and its `Name` attribute value is `SourceDir`. The `Directory` tag on line 54 contains Id and Name attributes set to GAC is a self-closing `Directory` tag because the installer recognizes the values for the GAC. On line 55, the `SSISINSTALLFOLDER` property Id is specified in a `Directory` tag that defines the `SSISINSTALLFOLDER` registry-informed path – configured earlier – to the SSIS Tasks folder. On my virtual machine running the Windows Server 2016 operation system, SQL Server 2017 is installed on the E: drive, and my `SSISINSTALLFOLDER` path resolves to `E:\Program Files (x86)\Microsoft SQL Server\140\DTS\Tasks`. Once installed on my virtual machine, the ExecuteCatalogPackageTask and ExecuteCatalogPackageTaskComplexUI assemblies will be deployed to `E:\Program Files (x86)\Microsoft SQL Server\140\DTS\Tasks`, as shown in Figure 22-30:

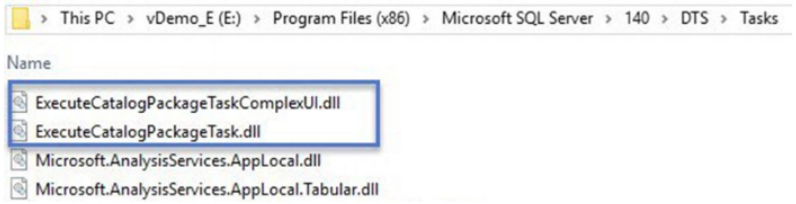


Figure 22-30 Task assemblies used in SSIS 2017 development

The ComponentGroup, Folders, and Folder Structure Fragment

The next `Fragment` tag covers the remainder of the WiX configuration.

Configure opening tags for the `ComponentGroup` and folders and folder structure `Fragment` using the XML in Listing 22-10:

```
<Fragment>
  <ComponentGroup Id="CustomSSISTask">
```

Listing 22-10 Open the ComponentGroup fragment tag

When added, the XML appears as shown in Figure 22-31:

```
<Fragment>
  <ComponentGroup Id="CustomSSISTask">
```

Figure 22-31 Opening the ComponentGroup fragment tag

Configure the ExecuteCatalogPackageTask GAC Registration

This last `Fragment` tag in the ExecuteCatalogPackageTask.wxs file configures the deployment of each assembly – ExecuteCatalogPackageTask and ExecuteCatalogPackageTaskComplexUI – in the development folder and the GAC. The `Fragment` details each deployment target in `Component` tags that include `File` and `RemoveFile` sub-tags. All `Component` tags are sub-tags of the `ComponentGroup` tag, which was defined earlier.

Configure the ExecuteCatalogPackageTask GAC registration `Component` tag using the XML in Listing 22-11:

```
<!-- ExecuteCatalogPackageTask [GAC] -->
<Component Id="Tasks_GAC"
           Directory="GAC"
           Guid="F4B72F57-AE0E-4EEE-8B04-10198ABDC523">
  <File Id="Task_GAC"
        Name="$(var.ExecuteCatalogPackageTask.TargetFileName)"
        Source="$(var.ExecuteCatalogPackageTask.TargetPath)"
        Assembly=".net"
        KeyPath="yes"
        Checksum="yes" />
  <RemoveFile Id="Task_GAC"
              On="uninstall"
              Name="$(var.ExecuteCatalogPackageTask.TargetFileName)" />
</Component>
```

Listing 22-11 Configure the ExecuteCatalogPackageTask GAC registration

When added, the XML appears as shown in Figure 22-32:

```
<!-- ExecuteCatalogPackageTask [GAC] -->
<Component Id="Tasks_GAC"
           Directory="GAC"
```