*Copyright © 2008 Microsoft Corporation. All rights reserved.*

|  |
| --- |
| Projects Tutorial 2: How to Create a Basic Project System |
|  |

|  |
| --- |
|  |

The first tutorial in this series, [Projects Tutorial 1: How to Create a Basic Project System](http://msdn.microsoft.com/en-us/library/cc512961.aspx), teaches how to create a basic project system. This tutorial builds on the basic project system by adding a Visual Studio template, a property page, and other features. You must complete the first tutorial before you start this one.

This tutorial teaches how to create a project type that has the project file name extension .myproj. To complete the tutorial, you do not have to create your own language because the tutorial borrows from the existing Visual C# project system.

|  |
| --- |
| **ms-help://AstroNS/dv_vsintegration/icons/alert_note.gifNote:** |
| For an end-to-end sample of a complete language project system, see the [IronPython sample](http://go.microsoft.com/fwlink/?LinkID=120017) . |

This tutorial teaches how to accomplish these tasks:

* Create a Visual Studio template.
* Deploy a Visual Studio template.
* Create a project type child node in the **New Project** dialog box.
* Enable parameter substitution in the Visual Studio template.
* Create a project property page.

|  |
| --- |
| **ms-help://AstroNS/dv_vsintegration/icons/alert_note.gifNote:** |
| The steps in this tutorial are based on a C# project. However, except for specifics such as file name extensions and code, you can use the same steps for a Visual Basic project. |

**ms-help://AstroNS/dv_vsintegration/icons/collapse_all.gifCreating a Visual Studio Template**

[Projects Tutorial 1: How to Create a Basic Project System](http://msdn.microsoft.com/en-us/library/cc512961.aspx) shows how to create a basic project template and add it to the project system. It also shows how to register this template with Visual Studio by using the ProvideProjectFactoryAttribute attribute, which writes the full path of the \Templates\Projects\SimpleProject\ folder in the system registry.

By using a Visual Studio template (.vstemplate file) instead of a basic project template, you can control how the template appears in the **New Project** dialog box and how template parameters are substituted. A .vstemplate file is an XML file that describes how source files are to be included when a project is created by using the project system template. The project system itself is built by collecting the .vstemplate file and the source files in a .zip file, and deployed by copying the .zip file to a location that is known to Visual Studio. This process is explained in more detail later in this tutorial.

**To create a Visual Studio template**

1. In Visual Studio 2008, open the SimpleProject solution that you created by following [Projects Tutorial 1: How to Create a Basic Project System](http://msdn.microsoft.com/en-us/library/cc512961.aspx).
2. In the SimpleProjectPackage.cs file, replace the second parameter (the project name) of the ProvideProjectFactory attribute by using null, as follows.

|  |
| --- |
| [ProvideProjectFactory(  typeof(SimpleProjectFactory),  null, |

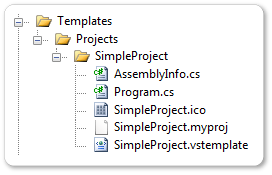
1. Replace the fourth parameter (the path to the project template folder) by using ".\\NullPath"

|  |
| --- |
| [ProvideProjectFactory(  typeof(SimpleProjectFactory),  null,  "Simple Project Files (\*.zzproj);\*.zzproj",  "zzproj", "zzproj",  ".\\NullPath",  LanguageVsTemplate = "SimpleProject")] |

1. Add an XML file named SimpleProject.vstemplate to the \Templates\Projects\SimpleProject\ folder.
2. Replace the contents of SimpleProject.vstemplate by using the following code.

|  |
| --- |
| <VSTemplate Version="2.0.0" Type="Project"  xmlns="http://schemas.microsoft.com/developer/vstemplate/2005">  <TemplateData>  <Name>SimpleProject Application</Name>  <Description>  A project for creating a SimpleProject application  </Description>  <Icon>SimpleProject.ico</Icon>  <ProjectType>SimpleProject</ProjectType>  </TemplateData>  <TemplateContent>  <Project File="SimpleProject.myproj" ReplaceParameters="true">  <ProjectItem ReplaceParameters="true" OpenInEditor="true">  Program.cs  </ProjectItem>  <ProjectItem ReplaceParameters="true" OpenInEditor="false">  AssemblyInfo.cs  </ProjectItem>  </Project>  </TemplateContent>  </VSTemplate> |

1. In the **Properties** window, select all five files in the \Templates\Projects\SimpleProject\ folder and set the **Build Action** to **ZipProject**.



The <TemplateData> section determines the location and appearance of the SimpleProject project type in the **New Project** dialog box, as follows:

* The <Name> element names the project template to be SimpleProject Application.
* The <Description> element contains the description that appears in the **New Project** dialog box when the project template is selected.
* The <Icon> element specifies the icon that appears together with the SimpleProject project type.
* The <ProjectType> element names the Project type in the **New Project** dialog box. This name replaces the project name parameter of the ProvideProjectFactory attribute.

|  |
| --- |
| **ms-help://AstroNS/dv_vsintegration/icons/alert_note.gifNote:** |
| The <ProjectType> element must match the LanguageVsTemplate argument of the ProvideProjectFactory attribute in the SimpleProjectPackage.cs file. |

The <TemplateContent> section describes these files that are generated when a new project is created:

* SimpleProject.myproj
* Program.cs
* AssemblyInfo.cs

All three files have **ReplaceParameters** set to true, which enables parameter substitution. The Program.cs file has **OpenInEditor** set to true, which causes the file to be opened in the code editor when a project is created.

For more information about the elements in the Visual Studio Template schema, see the Visual Studio Template Schema Reference.

|  |
| --- |
| **ms-help://AstroNS/dv_vsintegration/icons/alert_note.gifNote:** |
| If a project has more than one Visual Studio template, every template is in a separate folder. Every file in that folder must have the Build Action set to ZipProject. |

**ms-help://AstroNS/dv_vsintegration/icons/collapse_all.gifAdding a Minimal .vsct File**

Visual Studio must be run in setup mode to recognize a new or modified Visual Studio template. Setup mode requires a .vsct file to be present. Therefore, you must add a minimal .vsct file to the project.

**To add a minimal .vsct file**

1. In **Solution Explorer**, add an XML file named SimpleProject.vsct to the SimpleProject project.
2. In the **Properties** window, set the **Build Action** of this file to **VSCTCompile**.
3. In **Solution Explorer**, right-click the SimpleProject node and then click **Unload Project**.
4. Right-click the SimpleProject node and then click **Edit SimpleProject.csproj**.
5. Locate the SimpleProject.vsct item.

|  |
| --- |
| <None Include="SimpleProject.vsct" /> |

1. Change the build action to **VSCTCompile**.

|  |
| --- |
| <VSCTCompile Include="SimpleProject.vsct" /> |

1. Save the project file and close the editor.
2. Right-click the SimpleProject node, and then click **Reload Project**.
3. Replace the contents of the SimpleProject.vsct file by using the following code.

|  |  |
| --- | --- |
| <?xml version="1.0" encoding="utf-8" ?>  <CommandTable    xmlns="http://schemas.microsoft.com/VisualStudio/2005-10-18/CommandTable">  </CommandTable> |  |
|  | |

**ms-help://AstroNS/dv_vsintegration/icons/collapse_all.gifExamining the Visual Studio Template Build Steps**

The VSPackage project build system typically runs Visual Studio in setup mode when the .vstemplate file is changed or the project that contains the .vstemplate file is rebuilt. You can follow along by setting the verbosity level of MSBuild to Normal or higher.

**To examine the Visual Studio template build steps**

1. On the **Tools** menu, click **Options**.
2. Expand the **Projects and Solutions** node, and then select **Build and Run**.
3. Set the MSBuild project build output verbosity to **Normal**, **Detailed**, or **Diagnostic**.
4. Click OK.
5. Rebuild the SimpleProject project.

The build step to create the .zip project file should resemble the following example.

|  |
| --- |
| Target ZipProjects:  Build\_Status Zipping ProjectTemplates  Zipping D:\SimpleProject\SimpleProject\obj\Debug\SimpleProject.zip...  Copying file from "D:\SimpleProject\SimpleProject\obj\Debug\SimpleProject.zip" to  "\\Users\<YourAccount>\AppData\Roaming\Microsoft\VisualStudio\9.0Exp\ProjectTemplates\SimpleProject.zip". |

The build step to run Visual Studio in setup mode should resemble the following example.

|  |
| --- |
| Target SettingUpDevenv:  \\Program Files\Microsoft Visual Studio 9.0\Common7\IDE\devenv.com  /rt "Software\Microsoft\VisualStudio\9.0Exp" /RANU /setup |

**ms-help://AstroNS/dv_vsintegration/icons/collapse_all.gifDeploying a Visual Studio Template**

Visual Studio templates do not contain path information. Therefore, the template .zip file must be deployed to a location that is known to Visual Studio. During development, Visual Studio is typically run under RANU (run as normal user). Under RANU, when you build the SimpleProject factory, the build process copies any templates to the ProjectTemplates folder of your user account. The location of the ProjectTemplates folder is typically \\Users\*<YourAccount>*\AppData\Roaming\Microsoft\VisualStudio\9.0Exp\ProjectTemplates\.

To deploy your project factory, its installation program must have administrator privileges. It must deploy any templates to a known location on the computer, typically \Program Files\Microsoft Visual Studio 9.0\Common7\IDE\ProjectTemplates\.

If you are familiar with Wix deployment, you can obtain more information about how to install a project factory by examining the Product.wxs and Integration.wxs files in the IronPython sample Setup folder, which is typically located at \Program Files\Microsoft Visual Studio 2008 SDK\VisualStudioIntegration\Samples\IronPythonIntegration\Setup\.

**ms-help://AstroNS/dv_vsintegration/icons/collapse_all.gifTesting a Visual Studio Template**

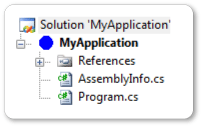
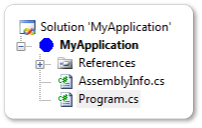
Test your project factory to see whether it creates a project hierarchy by using the Visual Studio template.

**To test the Visual Studio Template**

1. Close Visual Studio.
2. On the taskbar, click **Start**, and then in sequence, click **All Programs**, **Microsoft Visual Studio 2008 SDK**, **Tools**, **Reset the Microsoft Visual Studio 2008 Experimental hive**.
3. After the Command Prompt window closes, open Visual Studio.
4. Rebuild the SimpleProject project.
5. Press F5 to start an experimental instance of Visual Studio.
6. Choose a default environment setting.
7. On the **File** menu, point to **New**, and then click **Project**.
8. In the **New Project** dialog box, in the Project types pane, select SimpleProject.
9. Under **Visual Studio installed templates**, select **SimpleProjectApplication**.
10. Select **Create Directory for Solution**, and then type **MyApplication** in the **Name** box.
11. Type a location for the solution, for example, D:\.
12. Click **OK**.

Visual Studio calls your project factory to create a project by using the Visual Studio template.

1. Close the experimental instance of Visual Studio.



**ms-help://AstroNS/dv_vsintegration/icons/collapse_all.gifCreating a Project Type Child Node**

You can add a child node to a project type node in the **New Project** dialog box. For example, for the SimpleProject project type, you could have child nodes for console applications, window applications, web applications, and so on.

Child nodes are created by altering the project file and adding <OutputSubPath> children to the <ZipProject> elements. When a template is copied during build or deployment, every child node becomes a subfolder of the project templates folder.

This section shows how to create a Console child node for the SimpleProject project type.

**To create a project type child node**

1. Rename the \Templates\Projects\SimpleProject\ folder to \Templates\Projects\ConsoleApp\.
2. In the **Properties** window, select all five files in the \Templates\Projects\ConsoleApp\ folder and set the **Build Action** to **ZipProject**.
3. In the SimpleProject.vstemplate file, add the following line at the end of the <TemplateData> section, just before the closing tag.

|  |
| --- |
| <NumberOfParentCategoriesToRollUp>1</NumberOfParentCategoriesToRollUp> |

1. This causes the Console Application template to appear both in the Console child node and in the SimpleProject parent node, which is one level above the child node.
2. Save the SimpleProject.vstemplate file.
3. In **Solution Explorer**, right-click the **SimpleProject** node and then click **Unload Project**.
4. Right-click the **SimpleProject** node and then click **Edit SimpleProject.csproj**.
5. Locate the <ZipProject> elements. In the following example, they are organized in <ItemGroups> for clarity.

|  |
| --- |
| <ItemGroup>  <ZipProject Include="Templates\Projects\ConsoleApp\AssemblyInfo.cs" />  <ZipProject Include="Templates\Projects\ConsoleApp\Program.cs" />  <ZipProject  Include="Templates\Projects\ConsoleApp\SimpleProject.ico" />  <ZipProject  Include="Templates\Projects\ConsoleApp\SimpleProject.vstemplate" />  <ZipProject  Include="Templates\Projects\ConsoleApp\SimpleProject.zzproj" />  </ItemGroup> |

1. To each <ZipProject> element, add an <OutputSubPath> element and give it the value Console.

|  |
| --- |
| <ItemGroup>  <ZipProject Include="Templates\Projects\ConsoleApp\AssemblyInfo.cs">  <OutputSubPath>Console</OutputSubPath>  </ZipProject>  <ZipProject Include="Templates\Projects\ConsoleApp\Program.cs">  <OutputSubPath>Console</OutputSubPath>  </ZipProject>  <ZipProject Include="Templates\Projects\ConsoleApp\SimpleProject.ico">  <OutputSubPath>Console</OutputSubPath>  </ZipProject>  <ZipProject  Include="Templates\Projects\ConsoleApp\SimpleProject.vstemplate">  <OutputSubPath>Console</OutputSubPath>  </ZipProject>  <ZipProject  Include="Templates\Projects\ConsoleApp\SimpleProject.zzproj">  <OutputSubPath>Console</OutputSubPath>  </ZipProject>  </ItemGroup> |

1. Add this <PropertyGroup> to the project file:

|  |
| --- |
| <PropertyGroup>  <VsTemplateLanguage>SimpleProject</VsTemplateLanguage>  </PropertyGroup> |

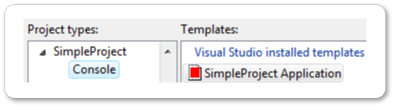
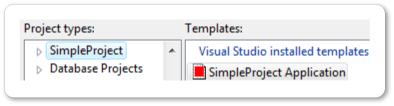
1. Save the project file and close the editor.
2. Right-click the **SimpleProject** node, and then click **Reload Project**.

**ms-help://AstroNS/dv_vsintegration/icons/collapse_all.gifTesting the Project Type Child Node**

Test the modified project file to see whether the **Console** child node appears in the **New Project** dialog box.

**To test the project type child node**

1. Close Visual Studio.
2. Run the **Reset the Microsoft Visual Studio 2008 Experimental hive** tool.
3. When the Command Prompt window closes, open Visual Studio.
4. Rebuild the SimpleProject project.
5. Press F5 to start an experimental instance of Visual Studio.
6. Choose a default environment setting.
7. On the **File** menu, point to **New**, and then click **Project**.
8. In the **Project types** pane, click the **SimpleProject** node.The **Console Application** template should appear in the **Templates** pane.
9. Expand the **SimpleProject** node.The **Console** child node should appear. The **Console Application** template continues to appear in the **Templates** pane.
10. . Click **Cancel**, and then close the experimental instance of Visual Studio.



**ms-help://AstroNS/dv_vsintegration/icons/collapse_all.gifSubstituting Project Template Parameters**

[Projects Tutorial 1: How to Create a Basic Project System](http://msdn.microsoft.com/en-us/library/cc512961.aspx) showed how to overwrite the **ProjectNode.AddFileFromTemplate** method to achieve basic template parameter substitution. This section teaches how to use the more sophisticated Visual Studio template parameters.

When you create a project by using a Visual Studio template in the **New Project** dialog box, template parameters are replaced with strings to customize the project. A template parameter is a special token that begins and ends with a dollar sign, for example, $time$. The following two parameters are especially useful for enabling customization in projects that are based on the template:

* $GUID[1-10]$ is replaced by a new Guid. You can specify up to 10 unique GUIDs, for example, $guid1$.
* $safeprojectname$ is the name provided by a user in the **New Project** dialog box, modified to remove all unsafe characters and spaces.

For a complete list of template parameters, see Template Parameters. If you want to create your own custom template parameter, see How to: Pass Custom Parameters to Templates.

**To substitute project template parameters**

1. In the SimpleProjectNode.cs file, comment out or remove the **AddFileFromTemplate** method.
2. In the \Templates\Projects\ConsoleApp\SimpleProject.myproj file, locate the <RootNamespace> property and change its value to $safeprojectname$.

|  |
| --- |
| <RootNamespace>$safeprojectname$</RootNamespace> |

1. In the \Templates\Projects\SimpleProject\Program.cs file, replace the contents of the file by using the following code:

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Text;  using System.Runtime.InteropServices; // Guid  namespace $safeprojectname$  {  [Guid("$guid1$")]  public class $safeprojectname$  {  static void Main(string[] args)  {  Console.WriteLine("Hello VSX!!!");  Console.ReadKey();  }  }  } |

1. Rebuild the SimpleProject project.
2. Press F5 to start an experimental instance of Visual Studio.
3. On the **File** menu, point to **New**, and then click **Project**.
4. In the **Project types** pane, select **SimpleProject**.
5. Under **Visual Studio installed templates**, select **Console Application**.
6. Select **Create Directory for Solution**, and accept the default name and location.
7. Click **OK**.

Visual Studio calls your project factory to create a project based on the Visual Studio template. The new program.cs file is opened in the code editor.

1. Click **Cancel**, and then close the experimental instance of Visual Studio.

The new Program.cs file shows the template parameter substitutions. (GUID values in your file will differ.)

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Text;  using System.Runtime.InteropServices; // Guid  namespace Console\_Application1  {  [Guid("9dd573f0-199d-45bd-9652-608ed22db574)"]  public class Console\_Application1  {  static void Main(string[] args)  {  Console.WriteLine("Hello VSX!!!");  Console.ReadKey();  }  }  } |

**ms-help://AstroNS/dv_vsintegration/icons/collapse_all.gifCreating a Project Property Page**

You can create a property page for your project type so that users can view and change properties in projects that are based on your template. This section teaches how to create a configuration-independent property page. This basic property page uses a properties grid to display the public properties that you expose in your property page class.

Derive your property page class from the **SettingsPage** base class. The properties grid provided by the **SettingsPage** class is aware of most primitive data types and knows how to display them. In addition, the **SettingsPage** class knows how to persist property values to the project file.

The property page you create in this section lets you alter and save these project properties:

* AssemblyName
* OutputType
* RootNamespace.

**To create a project property page**

1. In the SimpleProjectPackage.cs file, add this **ProvideObject** attribute to the **SimpleProjectPackage** class:

|  |
| --- |
| [ProvideObject(typeof(GeneralPropertyPage))]  public sealed class SimpleProjectPackage : ProjectPackage |

1. This registers the property page class **GeneralPropertyPage** with COM.
2. In the SimpleProjectNode.cs file, add these two overridden methods to the **SimpleProjectNode** class:

|  |
| --- |
| protected override Guid[] GetConfigurationIndependentPropertyPages()  {  Guid[] result = new Guid[1];  result[0] = typeof(GeneralPropertyPage).GUID;  return result;  }  protected override Guid[] GetPriorityProjectDesignerPages()  {  Guid[] result = new Guid[1];  result[0] = typeof(GeneralPropertyPage).GUID;  return result;  } |

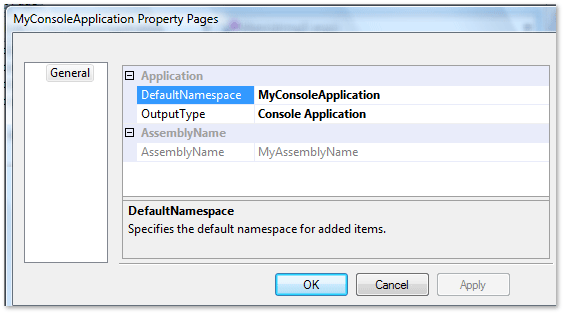
1. Both of these methods return an array of property page GUIDs. The GeneralPropertyPage GUID is the only element in the array, so the **Property Pages** dialog box will show only one page.
2. Add a class file named GeneralPropertyPage.cs to the SimpleProject project.
3. Replace the contents of this file by using the following code:

|  |
| --- |
| using System;  using System.Runtime.InteropServices;  using Microsoft.VisualStudio;  using Microsoft.VisualStudio.Package;  using System.ComponentModel;  namespace Company.SimpleProject  {  [ComVisible(true)]  [Guid("6BC7046B-B110-40d8-9F23-34263D8D2936")]  public class GeneralPropertyPage : SettingsPage  {  private string assemblyName;  private OutputType outputType;  private string defaultNamespace;  public GeneralPropertyPage()  {  this.Name = "General";  }  [Category("AssemblyName")]  [DisplayName("AssemblyName")]  [Description("The output file holding assembly metadata.")]  public string AssemblyName  {  get { return this.assemblyName; }  }  [Category("Application")]  [DisplayName("OutputType")]  [Description("The type of application to build.")]  public OutputType OutputType  {  get { return this.outputType; }  set { this.outputType = value; this.IsDirty = true; }  }  [Category("Application")]  [DisplayName("DefaultNamespace")]  [Description("Specifies the default namespace for added items.")]  public string DefaultNamespace  {  get { return this.defaultNamespace; }  set { this.defaultNamespace = value; this.IsDirty = true; }  }  protected override void BindProperties()  {  this.assemblyName = this.ProjectMgr.GetProjectProperty(  "AssemblyName", true);  this.defaultNamespace = this.ProjectMgr.GetProjectProperty(  "RootNamespace", false);  string outputType = this.ProjectMgr.GetProjectProperty(  "OutputType", false);  this.outputType =  (OutputType)Enum.Parse(typeof(OutputType), outputType);  }  protected override int ApplyChanges()  {  this.ProjectMgr.SetProjectProperty(  "AssemblyName", this.assemblyName);  this.ProjectMgr.SetProjectProperty(  "OutputType", this.outputType.ToString());  this.ProjectMgr.SetProjectProperty(  "RootNamespace", this.defaultNamespace);  this.IsDirty = false;  return VSConstants.S\_OK;  }  }  } |

1. The **GeneralPropertyPage** class exposes the three public properties AssemblyName, OutputType, and RootNamespace. Because AssemblyName has no set method, it is displayed as a read-only property. OutputType is an enumerated constant, so it appears as dropdown list.
2. The **SettingsPage** base class provides **ProjectMgr** to persist the properties. The **BindProperties** method uses **ProjectMgr** to retrieve the persisted property values and set the corresponding properties. The **ApplyChanges** method uses **ProjectMgr** to get the values of the properties and persist them to the project file. The property set method sets **IsDirty** to true to indicate that the properties have to be persisted. Persistence occurs when you save the project or solution.
3. Rebuild the SimpleProject solution.
4. Press F5 to start an experimental instance of Visual Studio.
5. On the **File** menu, point to **New**, and then click **Project**.
6. In the **Project types** pane, select **SimpleProject**.
7. Under **Visual Studio installed templates**, select **Console Application**.
8. Select **Create Directory for Solution**, and then in the **Name** box, type **MyConsoleApplication** and accept the default location.
9. Click **OK**.

Visual Studio calls your project factory to create a project by using the Visual Studio template. The new program.cs file is opened in the code editor.

1. Right-click the project node in **Solution Explorer**, and then click **Properties**.The **Property Pages** dialog box is displayed.

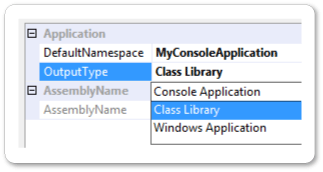


**ms-help://AstroNS/dv_vsintegration/icons/collapse_all.gifTesting the project property page**

Now you can test whether you can alter and persist property values.

**To test the project property page**

1. In the **MyConsoleApplication Property Pages** dialog box, change the **DefaultNamespace** to **MyApplication**.
2. Select the **OutputType** property, and then click the drop-down arrow. Select **Class Library**.
3. Click **Apply**, and then click **OK**.
4. Reopen the **Property Pages** dialog box and verify that your changes have been persisted.
5. Close the experimental instance of Visual Studio.
6. Reopen the experimental instance.
7. Reopen the **Property Pages** dialog box and verify that your changes have been persisted.
8. Close the experimental instance of Visual Studio.



**ms-help://AstroNS/dv_vsintegration/icons/collapse_all.gifSee Also**

**Concepts**

[Visual Studio Integration SDK Roadmap](http://msdn.microsoft.com/en-us/library/cc138569.aspx)

**Other Resources**

[Tutorials for Customizing Visual Studio By Using VSPackages](http://msdn.microsoft.com/en-us/library/cc138565.aspx)

[VSPackage Branding](http://msdn.microsoft.com/en-us/library/bb165358.aspx)