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

|  |  |
| --- | --- |
| Projects Tutorial 1: How to Create a Basic Project System | |
|  | |

In Visual Studio, projects are the containers that developers use to organize source code files and other assets that appear in **Solution Explorer**. Projects let you organize, build, debug, and deploy source code and create references to Web services, databases, and other resources.

Typically, projects are specified by the contents of a project file, for example, a .csproj file for a Visual C# project. You can create your own project type that has your own project file name extension. For more information about project types, see [Project Types](http://msdn.microsoft.com/en-us/library/bb165083.aspx).

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 [Visual Studio Integration Samples](http://msdn.microsoft.com/en-us/library/bb458154.aspx). |

This tutorial teaches how to accomplish these tasks:

* Create a basic project type.
* Create a basic project template.
* Register the project template with Visual Studio.
* Create a project instance by opening the **New Project** dialog box and then using your template.
* Create a project factory for your project system.
* Create a project node for your project system.
* Add custom icons for the project system.
* Implement basic template parameter substitution.

|  |
| --- |
| **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 Basic Project Type**

Project types, just like most Visual Studio extensions, are implemented by VSPackages. For more information about VSPackages, see [Tutorials for Customizing Visual Studio By Using VSPackages](http://msdn.microsoft.com/en-us/library/cc138565.aspx). To create a project type, you must first create a VSPackage.

**To create a VSPackage**

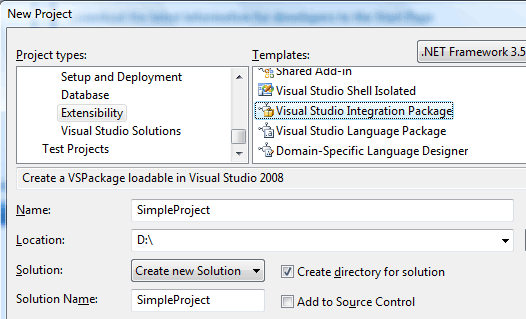
1. In Visual Studio, on the **File** menu, point to **New** and then click **Project**.
2. In the **New Project** dialog box, expand **Other Project Types** and then click **Extensibility**.

Under **Visual Studio installed templates**, click **Visual Studio Integration Package**.

Select **Create Directory for Solution**, and then type **SimpleProject** in the **Name** box.

Type a location for the solution, for example, **D:\**.

Click **OK**.



1. In the wizard, on the **Select a Programming Language** page, select **Visual C#** and **Generate a new key file to sign the assembly**.
2. On the **Select Test Options** page, clear both options and then click **Finish**.

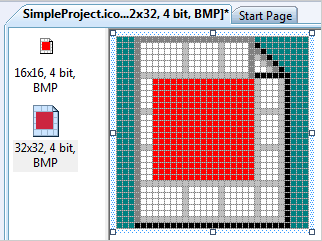
The wizard creates a VSPackage project that has the settings that you specified.

**ms-help://AstroNS/dv_vsintegration/icons/collapse_all.gifCreating a Basic Project Template**

Now, you can modify this basic VSPackage to implement the new .myproj project type. To create a project that is based on the .myproj project type, Visual Studio has to know which files, resources, and references to add to the new project. To provide this information, put project files in a project template folder. Then, when a user uses the .myproj project type in the **New Project** dialog box to create a project, the files are copied to the new project.

**To create a basic project template**

1. In **Solution Explorer**, right-click the **SimpleProject** project node, point to **Add**, and then click **New Folder**. Name the folder **Templates**.
2. In the **Templates** folder, add a folder named **Projects**.
3. In the **Projects** folder, add a folder named **SimpleProject**.
4. Right-click the **SimpleProject** folder, point to **Add**, and then click **New Item**. Add an Icon File named **SimpleProject.ico**. Click **Add** to open the icon editor.
5. Make the icon distinctive. This icon will appear in the **New Project** dialog box later in the tutorial.



1. Save the icon and close the icon editor.
2. In the **SimpleProject** folder, add a **Class** item named **Program.cs**. Click **Add** to open the code editor.
3. Replace the existing code by using the following lines.

|  |
| --- |
| using System;  using System.Collections.Generic;  using System.Text;  namespace $nameSpace$  {      public class $className$      {          static void Main(string[] args)          {              Console.WriteLine("Hello VSX!!!");              Console.ReadKey();          }      }  } |
| **ms-help://AstroNS/dv_vsintegration/icons/alert_note.gifNote:** |
| The files in the project template may contain template parameters that can be programmatically modified when the files are copied to a new project. Later in the tutorial, you can learn how to accomplish this for the template parameters, $nameSpace$ and $className$. |

This project template describes a basic Visual C# project that has both a Debug configuration and a Release configuration. The project includes two source files, AssemblyInfo.cs and Program.cs, and several assembly references. When a project is created from the template, the ProjectGuid value is automatically replaced by a new GUID.

In **Solution Explorer**, the expanded **Templates** folder should appear as follows:

Templates

    Projects

        SimpleProject

            AssemblyInfo.cs

                Program.cs

                SimpleProject.ico

                SimpleProject.myproj

**ms-help://AstroNS/dv_vsintegration/icons/collapse_all.gifCreating a Skeletal Project Factory**

You must tell Visual Studio the location of your project template folder. To do this, add an attribute to the VSPackage class that implements the project factory so that the template location is written to the system registry when the VSPackage is built. Start by creating a basic project factory that is identified by a project factory GUID. Use the ProvideProjectFactoryAttribute attribute to connect the project factory to the SimpleProjectPackage class.

**To create a skeletal project factory**

1. Open Guids.cs in the code editor.
2. On the **Tools** menu, click **Create GUID**.
3. Create a GUID for your project factory, or use the one in the following example. Add the GUID to the GuidList. The GUID must be in both Guid form and string form. The resulting code should resemble the following example.

|  |
| --- |
| static class GuidList  {  public const string guidSimpleProjectPkgString =  "96bf4c26-d94e-43bf-a56a-f8500b52bfad";  public const string guidSimpleProjectCmdSetString =  "72c23e1d-f389-410a-b5f1-c938303f1391";  public const string guidSimpleProjectFactoryString =  "471EC4BB-E47E-4229-A789-D1F5F83B52D4";  public static readonly Guid guidSimpleProjectCmdSet =  new Guid(guidSimpleProjectCmdSetString);  public static readonly Guid guidSimpleProjectFactory =  new Guid(guidSimpleProjectFactoryString);  }; |

1. Save the file and close the editor.
2. In **Solution Explorer**, right-click the **SimpleProject** project node, point to **Add**, and then click **New Item**. Add a **Class** named **SimpleProjectFactory.cs**. Click **Add** to open the code editor.
3. Add the following using statement after the other using statements.

|  |
| --- |
| using System.Runtime.InteropServices; |

1. Add a Guid attribute to the SimpleProjectFactory class. The value of the attribute is the new project factory GUID.

|  |
| --- |
| [Guid(GuidList.guidSimpleProjectFactoryString)]  class SimpleProjectFactory |

1. Rebuild the solution and verify that it builds without errors.

Now you can register your project template.

**To register the project template**

1. Open SimpleProjectPackage.cs in the code editor.
2. Add a ProvideProjectFactoryAttribute attribute to the SimpleProjectPackage class, as follows.

|  |
| --- |
| [ProvideProjectFactory(  typeof(SimpleProjectFactory),  "Simple Project",  "Simple Project Files (\*.myproj);\*.myproj",  "myproj", "myproj",  @"..\..\Templates\Projects\SimpleProject",  LanguageVsTemplate = "SimpleProject")]  [Guid(GuidList.guidSimpleProjectPkgString)]  public sealed class SimpleProjectPackage : Package |

1. Rebuild the solution and verify that it builds without errors.

Rebuilding registers the project template.

The ProvideProjectFactory attribute has the following syntax.

|  |
| --- |
| public ProvideProjectFactoryAttribute( Type factoryType, string name, string displayProjectFileExtensionsResourceID, string defaultProjectExtension, string possibleProjectExtensions, string projectTemplatesDirectory) |

The parameters *defaultProjectExtension* and *possibleProjectExtensions* are set to the project file name extension (.myproj). The *projectTemplatesDirectory* parameter is set to the relative path of the Templates folder. RegPkg.exe converts this to a full path to register it in the system registry.

**ms-help://AstroNS/dv_vsintegration/icons/collapse_all.gifExamining the Template Registration**

Run regedit.exe and examine the following system registry key.

HKEY\_CURRENT\_USER\Software\Microsoft\VisualStudio\9.0Exp\Configuration\Projects\{471EC4BB-E47E-4229-A789-D1F5F83B52D4}

|  |  |  |
| --- | --- | --- |
| **ms-help://AstroNS/dv_vsintegration/icons/alert_note.gifNote:** | | |
| The GUID in the \Projects\ key should match the value of GuidList.guidSimpleProjectFactoryString in the earlier section. The GUID in the \Package\ value name should match the value of GuidList.guidSimpleProjectPkgString. | | |
| **Value Name** | | **Type** | **Value Data** | |
| (Default) | | REG\_SZ | SimpleProjectFactory | |
| DefaultProjectExtension | | REG\_SZ | myproj | |
| DisplayName | | REG\_SZ | Simple Project | |
| DisplayProjectFileExtensions | | REG\_SZ | Simple Project Files (\*.myproj);\*.myproj | |
| Language(VsTemplate) | | REG\_SZ | SimpleProject | |
| Package | | REG\_SZ | {96bf4c26-d94e-43bf-a56a-f8500b52bfad} | |
| PossibleProjectExtensions | | REG\_SZ | myproj | |
| ProjectTemplatesDir | | REG\_SZ | D:\SimpleProject\...\Templates\Projects\SimpleProject | |

The value data is obtained by reflecting over the VSPackage assembly, especially the ProvideProjectFactory attribute. The \Package\ value data is obtained by reflecting over the SimpleProjectPackage class Guid attribute, the value of which is given by *GuidList.guidSimpleProjectPkgString*. The value of your package may differ.

The ProjectTemplatesDir value is the full path of the \Templates\Projects\SimpleProject\ folder that you created for the project. This folder holds the project template files that are copied and expanded when a project of type .myproj is created.

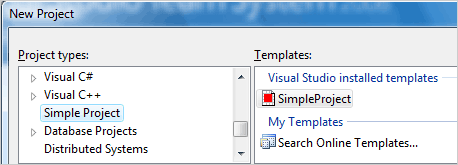
**ms-help://AstroNS/dv_vsintegration/icons/collapse_all.gifTesting the Template Registration**

Template registration tells Visual Studio the location of your project template folder so that Visual Studio can display the template name and icon in the **New Project** dialog box.

**To test the template registration**

1. In Visual Studio, press F5 to start an experimental instance of Visual Studio.
2. On the **File** menu, point to **New**, and then click **Project**.In the **New Project** dialog box, select the SimpleProject project type.

The SimpleProject icon appears under **Visual Studio installed templates**.



1. Close the experimental instance of Visual Studio.

Now you have a project factory that can demonstrate registration. However, it cannot yet create a project. The project package and project factory work together to create and initialize a project.

**ms-help://AstroNS/dv_vsintegration/icons/collapse_all.gifInitializing the Project Factory**

To implement the connection between the project package and the project factory, complete these tasks:

* To the solution, add links to the source-code files for the **Microsoft.VisualStudio.Package** Project framework. This framework is an extension of the Managed Package Framework (MPF). The Project framework is provided only as source code.
* Derive SimpleProjectPackage from **Microsoft.VisualStudio.Package.ProjectPackage**.
* Create a SimpleProjectFactory and register it with Visual Studio by using the **Microsoft.VisualStudio.Package.RegisterProjectFactory** method.
* Derive SimpleProjectPackage from **Microsoft.VisualStudio.Package.ProjectPackage**.
* Pass to the SimpleProjectFactory constructor a reference to the SimpleProjectPackage. This reference is cached for setting a service provider site later. For more information about services, see [Services](http://msdn.microsoft.com/en-us/library/bb166389.aspx).

**To initialize the project factory**

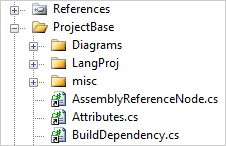
1. In **Solution Explorer**, right-click the **SimpleProject** node and then click **Unload Project**.
2. Right-click the **SimpleProject** node and then click **Edit SimpleProject.csproj**.
3. Locate the managed project source files by following [this link](http://go.microsoft.com/fwlink/?LinkId=122771). Copy these files to a local folder named, for example, C:\MyLocalFolder\Project
4. In the XML editor, add the following **ProjectBasePath** element after the **RegisterWithCodebase** element. Do not break the line that contains the **ProjectBasePath** element.

|  |
| --- |
| <RegisterWithCodebase>true</RegisterWithCodebase>  **<ProjectBasePath>C:\MyLocalFolder\Project</ProjectBasePath>** |

1. Add the following **Import** element after the existing **Import** elements.

|  |
| --- |
| <Import Project="$(MSBuildBinPath)\Microsoft.CSharp.targets" />  <Import Project="$(MSBuildExtensionsPath)\Microsoft\VisualStudio\v9.0\VSSDK\Microsoft.VsSDK.targets" />  <Import Project="$(ProjectBasePath)\ProjectBase.Files" /> |

1. Save the project file and close the editor.
2. Right-click the **SimpleProject** node, and then click **Reload Project**.**Solution Explorer** should now display a **ProjectBase** folder.



1. Right-click the **References** node, and add the following .NET references.
   * EnvDTE (*Visual Studio installation path*\Common7\IDE\PublicAssemblies\EnvDTE.dll)
   * **Microsoft.VisualStudio.Designer.Interfaces**
2. In the SimpleProjectPackage.cs file, add the following **using** statement after the existing **using** statements.

|  |
| --- |
| **using Microsoft.VisualStudio.Package;** |

1. Derive the SimpleProjectPackage class from **Microsoft.VisualStudio.Package.ProjectPackage**.

|  |
| --- |
| **public sealed class SimpleProjectPackage : ProjectPackage** |

1. Add the following line to the SimpleProjectPackage.Initialize method, just after base.Initialize.

|  |
| --- |
| **base.Initialize();**  **this.RegisterProjectFactory(new SimpleProjectFactory(this));** |

1. In SimpleProjectFactory.cs, add the following **using** statement after the existing **using** statements.

|  |
| --- |
| **using Microsoft.VisualStudio.Package;** |

1. Derive the SimpleProjectFactory class from **ProjectFactory**.

|  |
| --- |
| **class SimpleProjectFactory : ProjectFactory** |

1. Add the following dummy method to the SimpleProjectFactory class. You will implement this method in a later section.

|  |
| --- |
| **protected override ProjectNode CreateProject()**  **{**  **return null;**  **}** |

1. Add the following field and constructor to the SimpleProjectFactory class. This SimpleProjectPackage reference is cached in a private field so that it can be used in setting a service provider site.

|  |
| --- |
| **private SimpleProjectPackage package;**  **public SimpleProjectFactory(SimpleProjectPackage package)**  **: base(package)**  **{**  **this.package = package;**  **}** |

1. Rebuild the solution and verify that it builds without errors.

**ms-help://AstroNS/dv_vsintegration/icons/collapse_all.gifTesting the Project Factory Implementation**

Test whether the constructor for your project factory implementation is called.

**To test the project factory implementation**

1. In the SimpleProjectFactory.cs file, set a breakpoint on the following line in the SimpleProjectFactory constructor.

|  |
| --- |
| **this.package = package;** |

1. Press F5 to start an experimental instance of Visual Studio.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **New Project** dialog box, select the SimpleProject project type and then click **OK**.Execution stops at the breakpoint.
4. Clear the breakpoint.
5. Press SHIFT+F5 to stop debugging.

**ms-help://AstroNS/dv_vsintegration/icons/collapse_all.gifExtending the Project Node Class**

Now you can implement the SimpleProjectNode class, which derives from the **ProjectNode** class. The **ProjectNode** base class handles the following tasks of project creation:

* Copies the project template file, SimpleProject.myproj, to the new project folder. The copy is renamed according to the name that is entered in the **New Project** dialog box. The *ProjectGuid* property value is replaced by a new GUID.
* Traverses the MSBuild elements of the project template file, SimpleProject.myproj, and looks for **Compile** elements. For each **Compile** target file, copies the file to the new project folder.

The derived SimpleProjectNode class handles these tasks:

* Enables icons for project and file nodes in **Solution Explorer** to be created or selected.
* Enables additional project template parameter substitutions to be specified.

**To extend the project node class**

1. Right-click the **SimpleProject** project node, point to **Add**, and then click **New Item**. Add a **Class** named **SimpleProjectNode.cs**. Click **Add** to open the code editor.
2. Replace the existing code with the following code.

|  |
| --- |
| **using System;**  **using System.Collections.Generic;**  **using Microsoft.VisualStudio.Package;**  **namespace Company.SimpleProject**  **{**  **public class SimpleProjectNode : ProjectNode**  **{**  **private SimpleProjectPackage package;**  **public SimpleProjectNode(SimpleProjectPackage package)**  **{**  **this.package = package;**  **}**  **public override Guid ProjectGuid**  **{**  **get { return GuidList.guidSimpleProjectFactory; }**  **}**  **public override string ProjectType**  **{**  **get { return "SimpleProjectType"; }**  **}**  **public override void AddFileFromTemplate(**  **string source, string target)**  **{**  **this.FileTemplateProcessor.UntokenFile(source, target);**  **this.FileTemplateProcessor.Reset();**  **}**  **}**  **}** |

This SimpleProjectNode class implementation has these overridden methods:

* ProjectGuid, which returns the project factory GUID.
* ProjectType, which returns the localized name of the project type.
* AddFileFromTemplate, which copies selected files from the template folder to the destination project. This method is further implemented in a later section.

The SimpleProjectNode constructor, like the SimpleProjectFactory constructor, caches a SimpleProjectPackage reference in a private field for later use.

To connect the SimpleProjectFactory class to the SimpleProjectNode class, you must instantiate a new SimpleProjectNode in the SimpleProjectFactory.CreateProject method and cache it in a private field for later use.

**To connect the project factory class and the node class**

1. In the SimpleProjectFactory.cs file, add the following **using** statement after the existing **using** statements.

|  |
| --- |
| **using IOleServiceProvider = Microsoft.VisualStudio.OLE.Interop.IServiceProvider;** |

1. Replace the SimpleProjectFactory.CreateProject method by using the following code.

|  |
| --- |
| **protected override ProjectNode CreateProject()**  **{**  **SimpleProjectNode project = new SimpleProjectNode(this.package);**  **project.SetSite((IOleServiceProvider) ((IServiceProvider)this.package).GetService( typeof(IOleServiceProvider)));**  **return project;**  **}** |

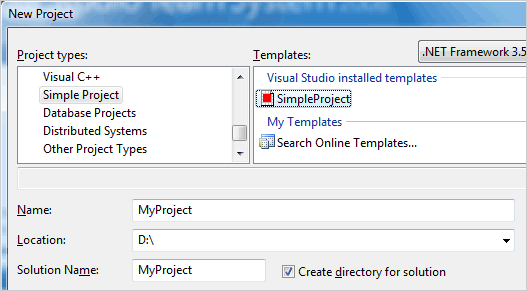
1. Rebuild the solution and verify that it builds without errors.

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

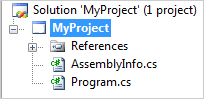
Test your project factory to see whether it creates a project hierarchy.

**To test the project node class**

1. Press F5 to start an experimental instance of Visual Studio.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **New Project** dialog box, under **Visual Studio installed templates**, select **SimpleProject**.
4. Select **Create Directory for Solution**, and type **MyProject** in the **Name** box.
5. Type a location for the solution, for example, **D:\**.
6. Click **OK**.



Visual Studio should call your project factory to create a project.



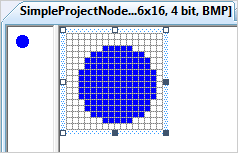
1. Close the experimental instance of Visual Studio.

**ms-help://AstroNS/dv_vsintegration/icons/collapse_all.gifAdding a Custom Project Node Icon**

The project node icon in the earlier section is a default icon. You can change it to a custom icon.

**To add a custom project node icon**

1. Right-click the **Resources** folder, point to **Add**, and then click **New Item**. Add a **Bitmap File** named SimpleProjectNode.bmp. Click **Add** to open the bitmap editor.
2. In the **Properties** windows, reduce the bitmap to 16 by 16 pixels. Make the bitmap distinctive.



1. In the **Properties** window, change the **Build action** of the bitmap to **Embedded Resource**.
2. In SimpleProjectNode.cs, add the following **using** statements after the existing **using** statements.

|  |
| --- |
| **using System.Drawing;**  **using System.Windows.Forms;** |

1. Add the following static field and constructor to the SimpleProjectNode class.

|  |
| --- |
| **private static ImageList imageList;**  **static SimpleProjectNode()**  **{**  **imageList = Utilities.GetImageList( typeof(SimpleProjectNode).Assembly.GetManifestResourceStream( "Company.SimpleProject.Resources.SimpleProjectNode.bmp"));**  **}** |

1. Add the following property to the beginning of the SimpleProjectNode class.

|  |
| --- |
| **internal static int imageIndex;**  **public override int ImageIndex**  **{**  **get { return imageIndex; }**  **}** |

1. Replace the instance constructor by using the following code.

|  |
| --- |
| **public SimpleProjectNode(SimpleProjectPackage package)**  **{**  **this.package = package;**  **imageIndex = this.ImageHandler.ImageList.Images.Count;**  **foreach (Image img in imageList.Images)**  **{**  **this.ImageHandler.AddImage(img);**  **}**  **}** |

During static construction, SimpleProjectNode retrieves the project node bitmap from the assembly manifest resources and caches it in a private field for later use. Notice the syntax of the [Assembly.GetManifestResourceStreamAssembly.GetManifestResourceNames](http://msdn2.microsoft.com/en-us/library/xc4235zt.aspx) image path. To see the names of the manifest resources embedded in an assembly, use the [Assembly.GetManifestResourceNames](http://msdn2.microsoft.com/en-us/library/system.reflection.assembly.getmanifestresourcenames.aspx) method. When this method is applied to the SimpleProject assembly, the results should be as follows:

* Company.SimpleProject.Resources.resources
* Microsoft.VisualStudio.Package.Project.resources
* Company.SimpleProject.VSPackage.resources
* Resources.imagelis.bmp
* Microsoft.VisualStudio.Package.DontShowAgainDialog.resources
* Microsoft.VisualStudio.Package.SecurityWarningDialog.resources
* Company.SimpleProject.Resources.SimpleProjectNode.bmp

During instance construction, the **ProjectNode** base class loads Resources.imagelis.bmp, in which are embedded commonly used 16 x 16 bitmaps from Resources\imagelis.bmp. This bitmap list is made available to SimpleProjectNode as ImageHandler.ImageList. SimpleProjectNode appends the project node bitmap to the list. The offset of the project node bitmap in the image list is cached for later use as the value of the public **ImageIndex** property. Visual Studio uses this property to determine which bitmap to display as the project node icon.

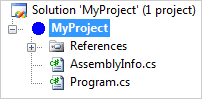
**ms-help://AstroNS/dv_vsintegration/icons/collapse_all.gifTesting the Custom Project Node Icon**

Test your project factory to see whether it creates a project hierarchy that has your custom project node icon.

**To test the custom project node icon**

1. Delete the MyProject solution.
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 **New Project** dialog box, under **Visual Studio installed templates**, select **SimpleProject**.
5. Select **Create Directory for Solution**, and type **MyProject** in the **Name** box.
6. Type a location for the solution, for example, **D:\**.
7. Click **OK**.

Visual Studio should create a project. Notice that SimpleProjectNode.bmp is used as the project node icon.



1. Open Program.cs in the code editor. You should see source code that resembles the following code.

|  |
| --- |
| **using System;**  **using System.Collections.Generic;**  **using System.Text;**  **namespace $nameSpace$**  **{**  **public class $className$**  **{**  **static void Main(string[] args)**  **{**  **Console.WriteLine("Hello VSX!!!");**  **Console.ReadKey();**  **}**  **}**  **}** |

1. Notice that the template parameters $nameSpace$ and $className$ do not have new values. You can implement template parameter substitution in the next section.
2. Press SHIFT+F5 to stop debugging.

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

In an earlier section, you registered the project template with Visual Studio by using the **ProvideProjectFactory** attribute. Registering the path of a template folder in this manner lets you enable basic template parameter substitution by overriding and expanding the **ProjectNode.AddFileFromTemplate** class. For more information, see [New Project Generation: Under the Hood, Part Two](http://msdn.microsoft.com/en-us/library/bb286986.aspx).

To replace the values of template parameters $nameSpace$ and $className$ in the Program.cs file template, you must add code to the **AddFileFromTemplate** class.

**To substitute template parameters**

1. In the SimpleProjectPackage.cs file, add the following **using** statement after the existing **using** statements.

|  |
| --- |
| **using System.IO;** |

1. Replace the **AddFileFromTemplate** method by using the following code.

|  |
| --- |
| **public override void AddFileFromTemplate(**  **string source, string target)**  **{**  **string nameSpace =**  **this.FileTemplateProcessor.GetFileNamespace(target, this);**  **string className = Path.GetFileNameWithoutExtension(target);**  **this.FileTemplateProcessor.AddReplace("$nameSpace$", nameSpace);**  **this.FileTemplateProcessor.AddReplace("$className$", className);**  **this.FileTemplateProcessor.UntokenFile(source, target);**  **this.FileTemplateProcessor.Reset();**  **}** |

1. Set a breakpoint in the method, just after the **className** assignment statement.

The assignment statements determine reasonable values for a namespace and a new class name. The two **ProjectNode.FileTemplateProcessor.AddReplace** method calls replace the corresponding template parameter values by using these new values.

**ms-help://AstroNS/dv_vsintegration/icons/collapse_all.gifTesting the Template Parameter Substitution**

Test your project factory to see if it creates a project hierarchy with your custom project node icon.

**To test the template parameter substitution**

1. Delete the MyProject solution.
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 **New Project** dialog box, under **Visual Studio installed templates**, select **SimpleProject**.
5. Select **Create Directory for Solution**, and type **MyProject** in the **Name** box.
6. Type a location for the solution, for example, **D:\**.
7. Click **OK**.

Execution stops at the breakpoint in the **AddFileFromTemplate** method.

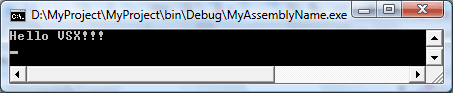
1. Examine the values for the *nameSpace* and *className* parameters.
   * *nameSpace* is given the value of the <RootNamespace> element in the \Templates\Projects\SimpleProject\SimpleProject.myproj project template file. In this case, the value is “MyRootNamespace”.
   * *className* is given the value of the class source file name, without the file name extension. In this case, the first file to be copied to the destination folder is AssemblyInfo.cs; therefore, the value of className is “AssemblyInfo”.
2. Remove the breakpoint and press F5 to continue execution.

Visual Studio should finish creating a project.

1. Open Program.cs in the code editor. You should see source code that resembles the following code.

|  |
| --- |
| **using System;**  **using System.Collections.Generic;**  **using System.Linq;**  **using System.Text;**  **namespace MyRootNamespace**  **{**  **public class Program**  **{**  **static void Main(string[] args)**  **{**  **Console.WriteLine("Hello VSX!!!");**  **Console.ReadKey();**  **}**  **}**  **}** |

1. Notice that the namespace is now “MyRootNamespace” and the class name is now “Program”.
2. In the experimental instance of Visual Studio, press F5. The new project should compile, run, and display “Hello VSX!!!” in the console window.



Congratulations! You have implemented a basic managed project system.

**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)