TechEd^{05}

Hands-On Lab

Lab Manual

HOL084 *Visual Studio 2005*

Templates and Starter Kits

Please do not remove this

manual from the lab.

VSTemplates – The new template architecture in VS 2005*

What is a Template?

The term ***template*** in Visual Studio refers to sample projects and items that appear in New Project/Item dialogs. These sample projects/items can be used as the base for creating new projects and items. In other words it's a pre-defined framework of certain commonly used projects or items. For example in the New Project dialog in Visual Studio you can find several project ***templates*** such as Windows Application, Console application etc.

If you have some common code or a set of resources files that are often re-used then creating your own template can be very useful.
As an example:
Let's say you have a system that relies on a plug-in architecture, and writing plug-ins involves implementing multiple methods. Rather than having each programmer rewrite the same base code from scratch, you could create a project template that pre-populates most of the necessary code, requiring the programmer to only fill in the details.

So by using a template, you can achieve uniformity across your whole team/org, as well as save coding time and reduce errors.

Objectives of this HOL

Module 1 of this Lab introduces you to the new VSTemplate architecture for templates in Visual Studio 2005 (Code name Whidbey).

The new template format (.vstemplate file) is XML based. Therefore they are easy to read and understand as well as easy to author and customize. More details about the VSTemplate file format are included in the Appendix, and you may want to refer to it during the exercises.

This module will also introduce you to VS2005 Starter Kits and guide you through the creation of your very own Starter kit! A starter kit is an enhanced project template that can be shared with other members of the community. A starter kit includes code samples that compile, user guidance documentation, and other helpful resources to enable you to learn new tools and programming techniques while building useful, real world applications. Starter kits can be sample applications for customization or learning tools that show people how to use your products.

Module 2 introduces you to the concept of IWizard() in VS 2005. For certain types of templates, it is necessary to show a custom UI by running custom code to collect user input which can then be used to customize the project that is being created. This can be achieved by implementing the IWizard() interface.

You will walk through an implementation of the IWizard () interface to dynamically customize the projects created using the templates based on user input.

What to do for more information or help

- Prasadi de Silva (prasadis@microsoft.com) and Hiren Shah (hirens@microsoft.com) are here at TechEd 2005. We should be sitting in one of two places: 1) right here in the HOL lab or 2) in the dev community lounge. We'd love to talk to you. Come find us or drop us a line and we'll come find you☺.
- Talk to the entire VS template team by sending us mail at template@microsoft.com. We'd love to hear from you. Did you like this HOL? What can we do to improve it?

Getting Started

- Choose HOL084 from the drop down to log in
- Create a directory called c:\VSTemplates<yourfirstname>\
- From the Start menu, choose All Programs → Microsoft Visual Studio 2005 Beta 2 → Microsoft Visual Studio 2005 Beta 2 (or you may use the shortcut provided for you on the desktop)

When You Are Done

Please shutdown the virtual machine. This will make sure the system is ready for the next user.

*In VS2005 only CSharp, Visual Basic and JSharp languages support the new template architecture.

Module 1 – Introductions to VS templates and Starter Kits

Exercise 1A – Creating a Template for a *project* using the Visual Studio 2005 Export Template Wizard

In this exercise you will customize a VS ***project***, use the VS 2005 **Export Template Wizard** to create a Template file for that project (including a .vstemplate) and then use that template to create a new project. Don't forget to start by following the steps under the "Getting Started" section above.

Step1 – Customizing a project

i. From within Visual Studio, choose **File -> New -> Project.** Change the Location field to **C:\VSTemplates<yourfirstname>**

ii. Then select **Visual C# -> Windows** from the left hand menu and choose "Console Application", and click **OK** in the dialog box. This will create a new C# project called **ColsoleApplication1.**

iii. Once the project is created, double click on **program.cs** to open it for editing. (You will also see the **Properties** and **References,** under the project name in the solution explorer).

iv. In the editor, add the following code to the main method of **program.cs**

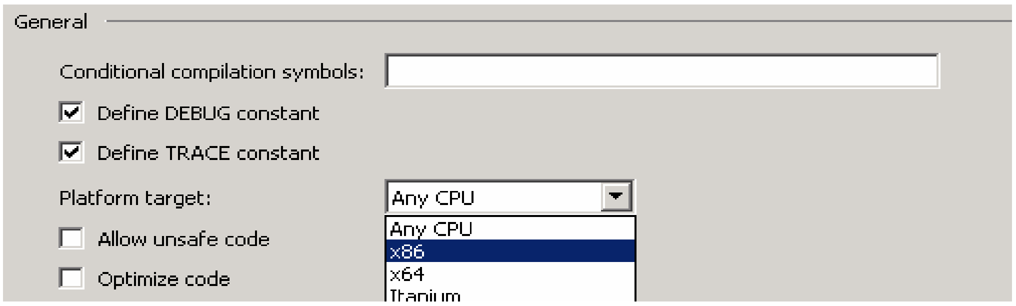
```
static void Main(string[] args)
        {
                Console.WriteLine("VS 2005 Templates are COOL!");
                Console.ReadLine();
        }
```

v. Choose **File → Save Program.cs**.

vi. Now hit F5 to build and run your application. You will see the console pop up with the message **VS 2005 Templates are COOL!** Hit **<ENTER>** to continue and close your application.

vii. Now let's change some properties for this project:

   a. Double click on the **Properties** node in the solution explorer to open the Project Properties designer.
   b. Choose the **Build** tab from the left hand menu, and choose **x86** from the drop down menu in the **Platform target** property as seen below.



General

Conditional compilation symbols:

☑ Define DEBUG constant

☑ Define TRACE constant

Platform target: Any CPU

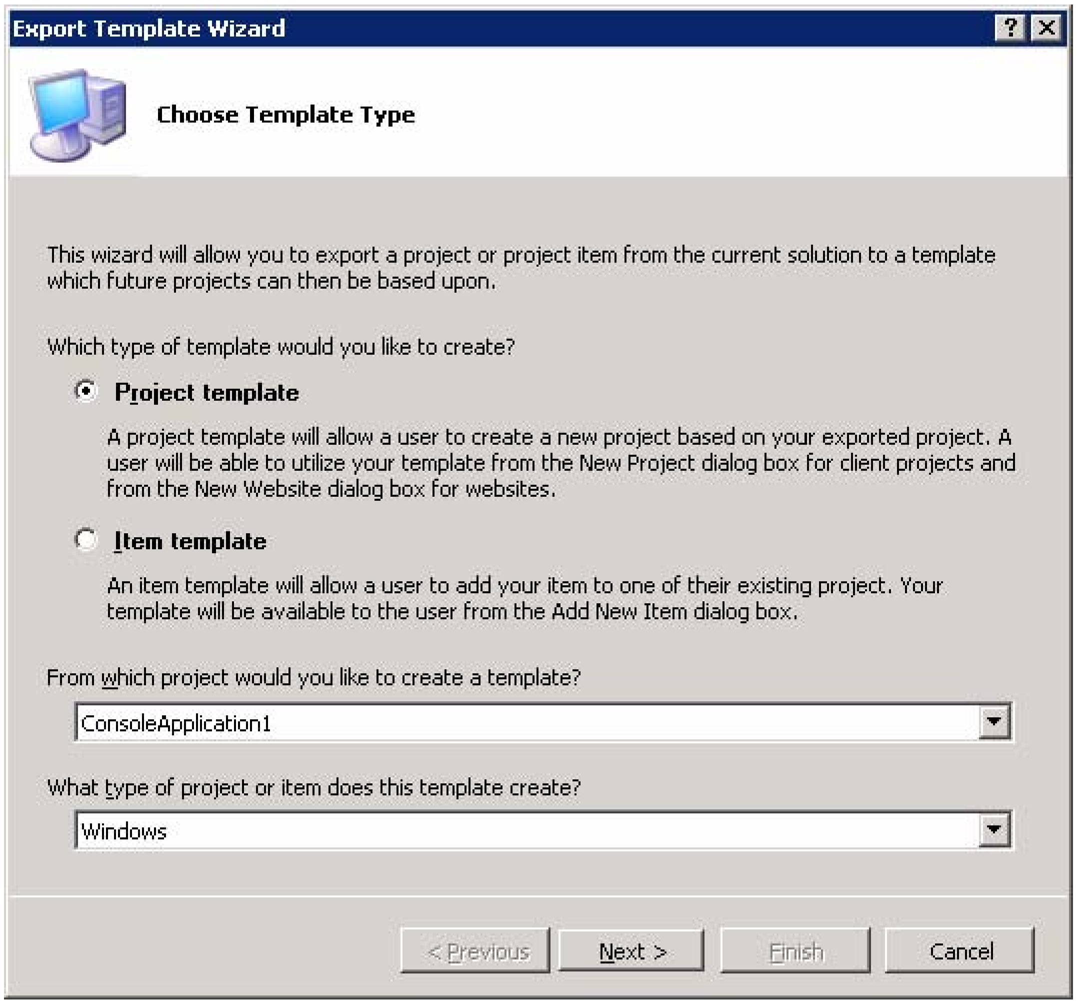
☐ Allow unsafe code

☐ Optimize code

   c. Choose **File->Save Selected Items** to save the project

Step 2 – Exporting the customized project

i. Choose **File->Export Template**
ii. Once the **Export Template Wizard** opens, choose **Project Template** as the type of template you would like to create
iii. The "From which Project…" list box will default to **ConsoleApplication1** as this is the only project you have in your solution at this time. Choose "**Windows**" from the "What type of project or item does this template create?" drop down list. Hit **Next.**



Choose Template Type

This wizard will allow you to export a project or project item from the current solution to a template which future projects can then be based upon.

Which type of template would you like to create?

[✓ Project template]

A project template will allow a user to create a new project based on your exported project. A user will be able to utilize your template from the New Project dialog box for client projects and from the New Website dialog box for websites.

Item template

An item template will allow a user to add your item to one of their existing project. Your template will be available to the user from the Add New Item dialog box.

From which project would you like to create a template?

ConsoleApplication1

What type of project or item does this template create?

Windows

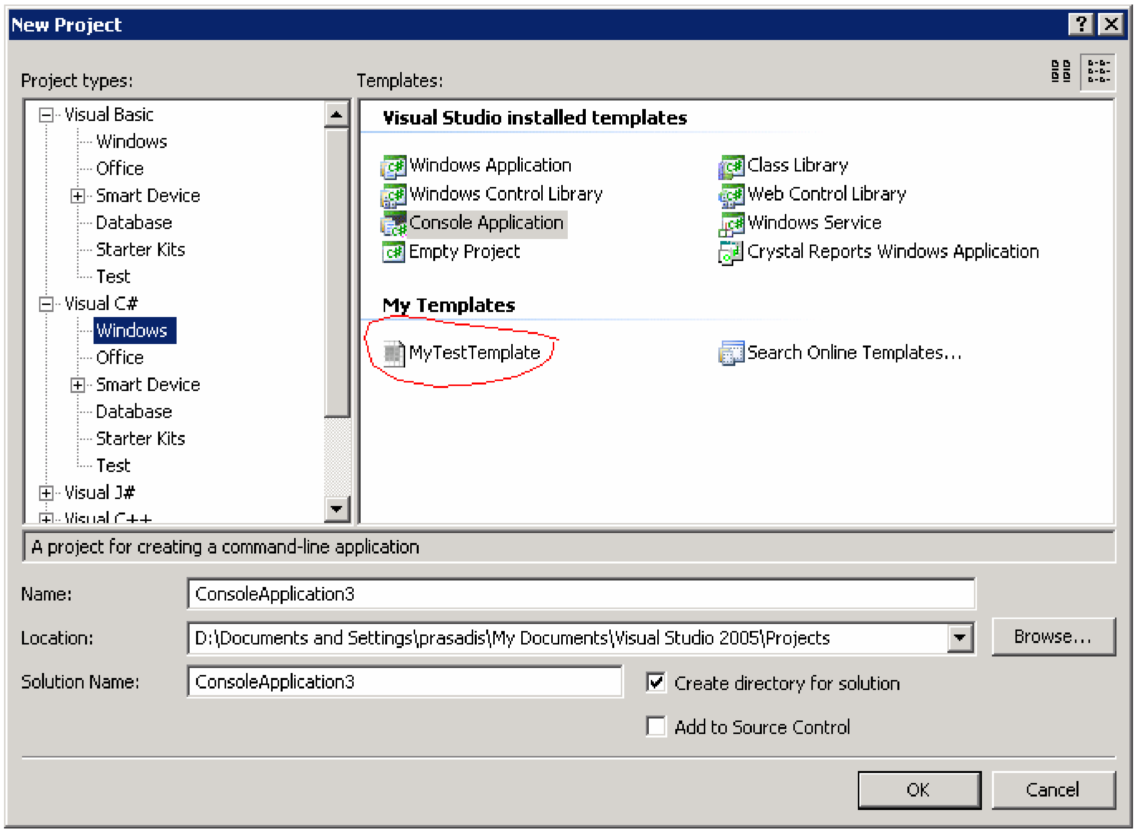
iv. On the next page type in "MyTestTemplate" and "My Test Template" in the **Template Name** and **Template Description** fields respectively.
v. Leave the other fields at their default value and hit **Finish.**
vi. Double click **MyTestTemplate.zip** in the explorer window that just opened. If the explorer window does not appear automatically you can find this Zip file under My Documents\Visual Studio 2005\My Exported Templates
vii. Open **MyTemplate.vstemplate** by double clicking. This is the vstemplate file that was created by the export template wizard.
viii. Verify that the information that you provided in the template wizard is included in the .vstemplate file. (See items in **Bold** below). The rest of the Data was inferred by the Wizard

```
  <TemplateData>
   <Name>MyTestTemplate</Name>
   <Description>My Test Template</Description>
   <Icon>__TemplateIcon.ico</Icon>
   <ProjectType>CSharp</ProjectType>
   <ProjectSubType>Windows</ProjectSubType>
   <SortOrder>1000</SortOrder>
   <CreateNewFolder>true</CreateNewFolder>
   <DefaultName>MyTestTemplate</DefaultName>
   <ProvideDefaultName>true</ProvideDefaultName>
   <LocationField>Enabled</LocationField>
   <EnableLocationBrowseButton>true</EnableLocationBrowseButton>
 </TemplateData>
```

ix. Choose **File->Close Solution** to clean up the VS shell before we begin **Step 3**. Choose **No** if you are presented with the “Save Changes to the following items?” dialog.

Step 3 – Using the customized project template to create a new project

i. Choose **File->New->Project.** You will see that the project template you just created now appears under **My Templates**!



A project for creating a command-line application

Name: ConsoleApplication3

Location: D:\Documents and Settings\prasadis\My Documents\Visual Studio 2005\Projects

Solution Name: ConsoleApplication3

☑ Create directory for solution

☐ Add to Source Control

ii. Choose **MyTestTemplate** and hit **OK**
iii. Once the project **MyTestTemplate1** is created, double click on **program.cs** to open it for editing.
iv. Verify that the code you added as part of Step1.iv is included inside program.cs file for this project.
v. Verify that the change you made to Build properties is available by default for this project (Double click on **Properties** to open the Project Properties designer, choose the **Build** tab and verify that the **Platform Target** property is set to X86)

Creating a template for your project was that easy!

Exercise 1B – Creating a Template for an *item* using the Visual Studio 2005 Export Template Wizard

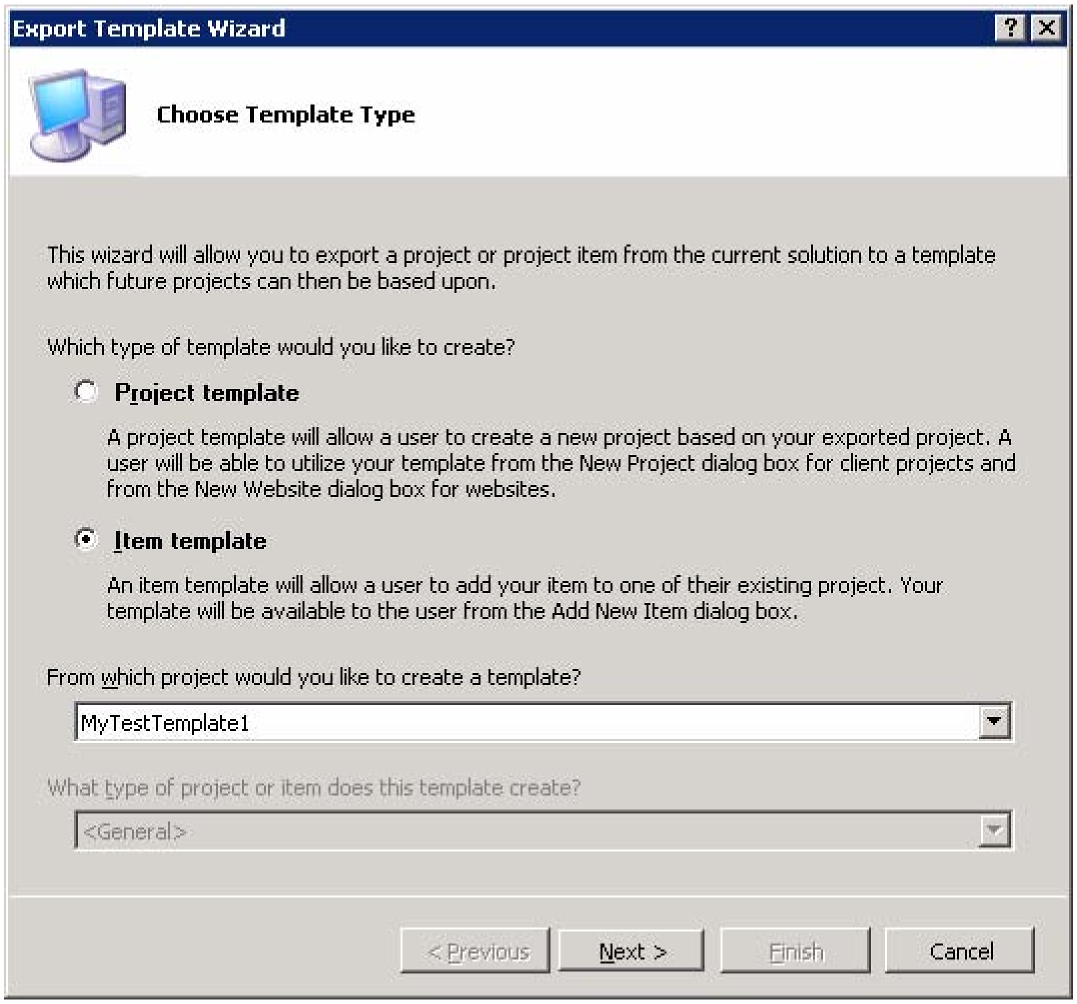
In this exercise you will customize a VS ***item*** and use the VS 2005 **Export Template** Wizard to create a template file for that item which includes a .vstemplate file.

In VS 2005, you can create templates not only for projects but for individual items as well. This is very useful in the event that you wish to share code, resources or other information across projects in the form of a class file, resource file or text file etc. The process for exporting items is similar to the method in which you created the project template in Exercise 1A.
In this example we will create a class file with the copyright information for you company built in.

i. Right click on the project (**MyTestTemplate1)** that you created in Excercise1A and choose **ADD->New Item**
ii. Choose **Class** from the “Visual Studio installed templates” and type in “MyClass1.cs” in the Name field. Click **ADD**
iii. Double click on **MyClass1.cs** and make the following modifications:

```
//File name <Insert Name>
//
//Copyright <Your Company name> ©2005
//
```

iv. Choose **File->Save MyClass.cs**
v. Choose **File->Export Template**
vi. Once the **Export Template Wizard** opens, choose **Item Template** as the type of template you would like to create. The “From which Project…” list box will default to **MyTestTemplate1** as this is the only project you have in your solution at this time. The “What type of project or item does this template create?” box will be grayed out. Hit **Next.**


Export Template Wizard

Choose Template Type

This wizard will allow you to export a project or project item from the current solution to a template which future projects can then be based upon.

Which type of template would you like to create?

( ) Project template

A project template will allow a user to create a new project based on your exported project. A user will be able to utilize your template from the New Project dialog box for client projects and from the New Website dialog box for websites.

(●) Item template

An item template will allow a user to add your item to one of their existing project. Your template will be available to the user from the Add New Item dialog box.

From which project would you like to create a template?

MyTestTemplate1

What type of project or item does this template create?

<General>

vii. On the next page type check **MyClass1.cs** as the item that you wish to export. Hit **Next.**

viii. In the **Select Item References** page you can choose to add additional references. These references will be added to the project when you use this template to add an item to your project. For project templates the references are already included in the project file, therefore no additional references need to be added at export time.
Hit **Next.**

ix. On the next page type in “MyClassTemplate” and “My Test Class Template” in the **Template Name** and **Template Description** fields respectively.

x. Leave the other fields at their default value and hit **Finish.**

xi. An explorer window will open, containing **MyClassTemplate.zip** and **MyTestTemplate.zip**. If the explorer window does not appear automatically you can find the Zip files under My Documents\Visual Studio 2005\My Exported Templates. Double click **MyClassTemplate.zip** to view the contents.

xii. Open **MyTemplate.vstemplate** by double clicking. This is the .vstemplate file that was created for the Class1.cs item by the export template wizard.

xiii. Verify that the information that you provided in the template wizard is included in the .vstemplate file. (See items in **Bold** below). The rest of the Data was inferred by the Wizard

```
<TemplateData>
      <Icon>__TemplateIcon.ico</Icon>
      <DefaultName>MyClassTemplate</DefaultName>
      <Name>MyClassTemplate</Name>
      <Description>My Test Class template</Description>
      <ProjectType>CSharp</ProjectType>
      <SortOrder>10</SortOrder>
</TemplateData>
```

If you added any references in step viii, you will see these in the `<TemplateData>` section of the .vstemplate.

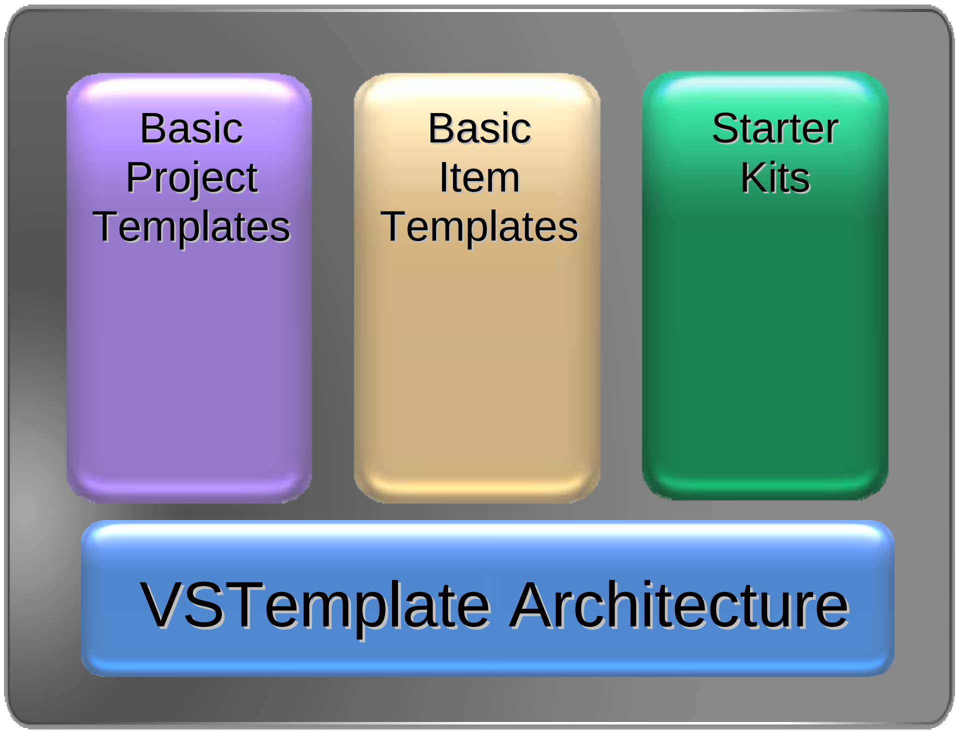
xiv. Switch back to VS 2005 IDE and right click on the project name and choose **Add->New->Item**. When the item templates pop up, verify that **MyClassTemplate** shows up under **My Templates.** You can now add this item to any project!

xv. Choose **File->Close Solution** to clean up the VS shell before we begin Step 3. Choose **No** if you are presented with the "Save Changes to the following items?" dialog.

The above exercises give some very simple examples of how you can use the new VS Template architecture to create your own custom files. The VS Template architecture can be used for much more complicated and complex scenarios. For example you can create multi project templates which contain a collection of projects that can be used to instantiate solutions. You can find more information about multi project templates in the Appendix here.

Exercise 3 – Creating a Starter Kit

A starter kit is essentially an enhanced project template that can be shared with other members of the community. It's built on the same VS Template architecture.

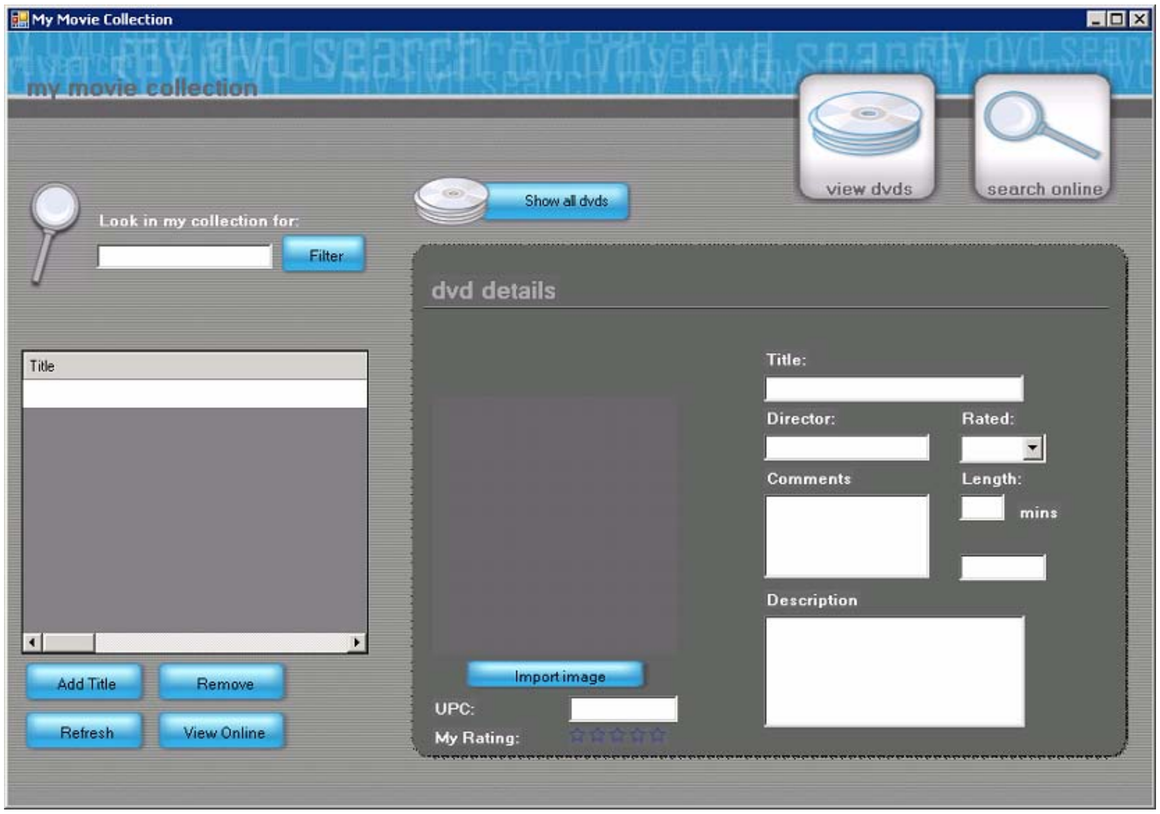


A starter kit includes code samples that compile, user guidance documentation, and other helpful resources to enable you to learn new tools and programming techniques while building useful, real world applications.

In this exercise you will use an existing solution to create a template, make some modification to the .vstemplate file and add user guidance documentation to complete the starter kit.

Step 1 – Opening and using the base project for the Starter Kit

i. From within Visual Studio, choose **File -> Open -> Project/Solution.** Browse to C:\HOL084\Module1\DVDProject
ii. Then select **DVDProject.sln** and click **OPEN** in the dialog box. This will open up the existing project as **DVDProject**
iii. Hit **F5** to start the application. You will see that it's an application that can store information about your DVD collection.



Feel free to play with the application further if you wish. When you finish click on the little **X** in the top right hand corner to close the application.

Step 2 – Adding user guidance documentation

Clearly if you wished other users to use this application some documentation would be very useful. Well let's add it!

i. Right Click on the **Documentation** node in the solution explorer and choose **Add->Existing Item**
ii. When the dialog box opens browse to **C:\HOL084\Module1\DVDProject\DVDProject\Documentation**. Make sure you select "All Files (*.*) in the "**Files of Type**" drop down menu.
iii. Select **Getting Started Tutorial.htm** and click **Add**. This document contains all the documentation necessary to get the users started with this application. To view the document right click on **Getting Started Tutorial.htm** in the solution explorer and select **View in Browser**. The contents of the htm file will open in an Internet Explorer window. Close the IE window when you have finished reading the document.

Step 3 – Exporting the project as a template to be used for the Starter Kit

In order to create the starter kit we need to first create a template out of the existing project. We will use the same process used in Step2 of Exercise 1A.

i. Choose **File->Export Template**
ii. Once the **Export Template Wizard** opens, choose **Project Template** as the type of template you would like to create
iii. The “From which Project…” dialog will default to **DVDProject** as this is the only project you have in your solution at this time. Leave the default **<General>** in the “What type of project or item does this template create?” drop down list. Hit **Next.**
iv. On the next page type in “MyDVDTemplate” and “My DVD Collection Template” in the **Template Name** and **Template Description** fields respectively.
v. Leave the other fields at their default value and hit **Finish.**
vi. Verify that the export succeeded and **MyDVDTemplate.zip** has been created inside explorer window that just opened. If the explorer window does not appear automatically you can find this Zip file under My Documents\Visual Studio 2005\Templates\ProjectTemplates\Visual C#

Step 4 – Modifying the .vstemplate file to enhance the user experience with the starter kit

Let’s make sure that the user guidance documentation is readily discoverable by the user

i. To modify the .vstemplate file for the **MyDVDTemplate** template we need to first extract the contents of the .zip file. For your convenience the files have already been extracted to **C:\HOL084\Module1\DVDTemplateExtracted**
ii. Open **MyTemplate.vstemplate** in this folder by double clicking. This is the vstemplate file that was created by the export template wizard.
iii. Find the line containing the name **Getting Started Tutorial.htm** of the documentation we added in **Step 2.iii**. The entry will look like this:

```
<Folder Name="Documentation">
        <ProjectItem ReplaceParameters="true" TargetFileName="Getting Started
    Tutorial.htm">GETTIN~1.HTM</ProjectItem>
```

iv. Modify this line to include **OpenInWebBrowser="true"** after the <ProjectItem tag as follows:

```
<Folder Name="Documentation">
        <ProjectItem OpenInWebBrowser="true" ReplaceParameters="true"
    TargetFileName="Getting Started Tutorial.htm">GETTIN~1.HTM</ProjectItem>
```

This tag will ensure that the htm file will open by default when the user creates a new project from your template.

v. Choose **File->Save MyTemplate.vstemplate**

vi. Choose File->Close Solution to clean up the VS shell before we begin Step 5. Choose Yes if you are presented with the “Save Changes to the following items?” dialog

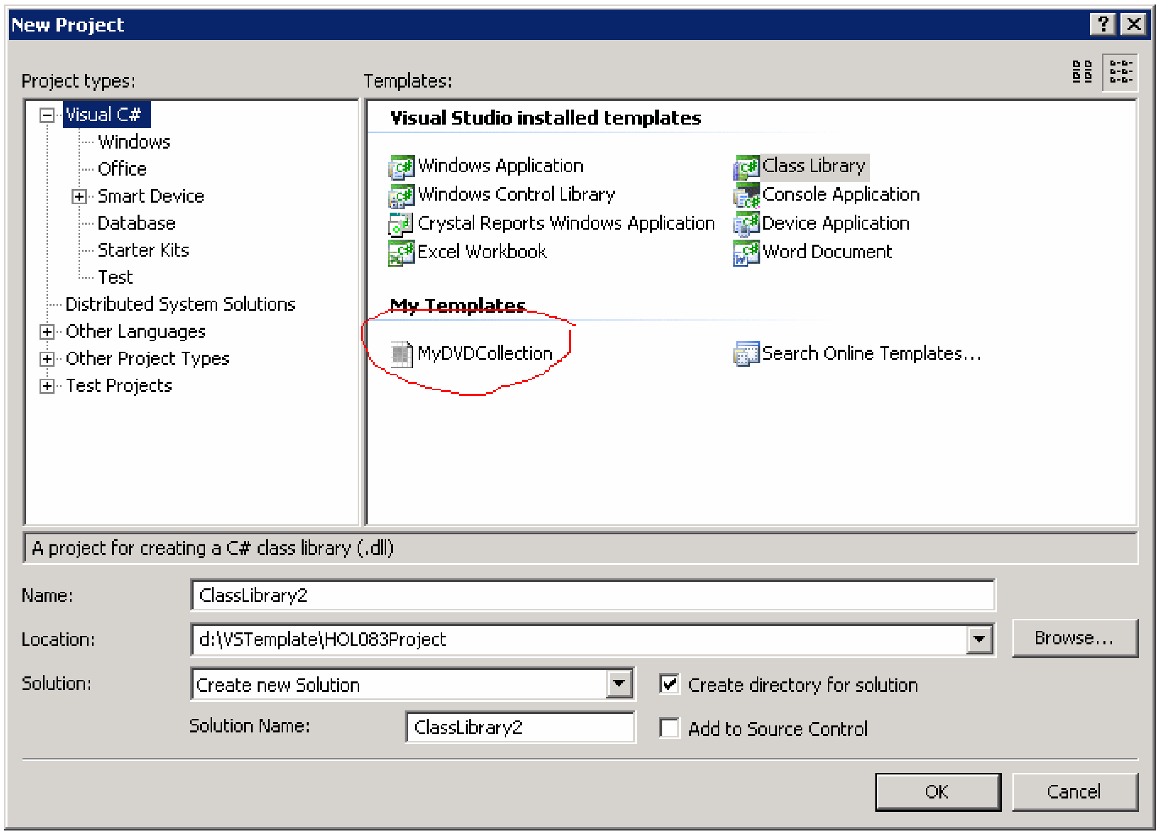
Step 5 – Final packaging and placement of the new starter kit

i. Now we need to zip up the contents once more with the modified .vstemplate file. To do this select all the files within **C:\HOL084\Module1\DVDTemplateExtracted** by selecting **CTRL+A**. Then right click and select **Send to->Compressed (zipped) Folder**. This will create a .zip file within the same folder.
ii. Depending on the file that you selected when compressing the folder the name of your ZIP folder maybe different so rename the .zip folder to **MyDVDCollection.zip**. This step is not required but useful for consistency.
iii. Move the **MyDVDCollection.zip** file to **My Documents\Visual Studio 2005\Templates\ProjectTemplates\Visual C#.** This step is required in order for your template to be available to the **File->New Project** dialog.

Congratulations! You have just completed creating your first starter kit!

Step 6 – Using the new starter kit

i. From within VS 2005 Choose **File->New->Project.** You will see that the project template you just created now appears under **My Templates**!



New Project

Project types:
- Visual C#
  - Windows
  - Office
  - Smart Device
  - Database
  - Starter Kits
  - Test
- Distributed System Solutions
- Other Languages
- Other Project Types
- Test Projects

Templates:

Visual Studio installed templates

Windows Application	Class Library
Windows Control Library	Console Application
Crystal Reports Windows Application	Device Application
Excel Workbook	Word Document

My Templates

MyDVDCollection	Search Online Templates...

A project for creating a C# class library (.dll)

Name: ClassLibrary2

Location: d:\VSTemplate\HOL083Project

Solution: Create new Solution

[✓] Create directory for solution

Solution Name: ClassLibrary2

[ ] Add to Source Control

ii. Choose **MyDVDCollection** and hit **OK**. You will notice that the **Getting Started Tutorial.htm** opens in the shell by default. So now it's very easy for users to start using and customizing this starter kit!

iii. Verify that the application still works by hitting **F5** and running it.

Step 7 – Creating .vsi file for the starter kit

VS 2005 makes it very easy for you to share starter kits that you create with the community.

The community installer that ships with VS 2005 allows you to easily install a variety of packages in the .vsi format such as starter kits, code snippet and add-ins. The .vsi file format allows the content of these items to be packaged up in a format that is recognized by the community installer. For starter kits the .vsi file is essentially a zip files that also contain a metadata file called the VSContent file.

i. Locate the **MyDVDCollection.zip** file that was created in **Step 5.ii** for your starter kit in **My Documents\Visual Studio 2005\Templates\ProjectTemplates\Visual C#.**
ii. Open another instance of VS 2005 by using the short cut on your desktop and choose **File->New->File->XML File** to open a blank xml file. (For the purpose of this exercise it is important that you use the Visual Studio XML editor due encoding issues)
iii. Replace the contents of that file with the following:

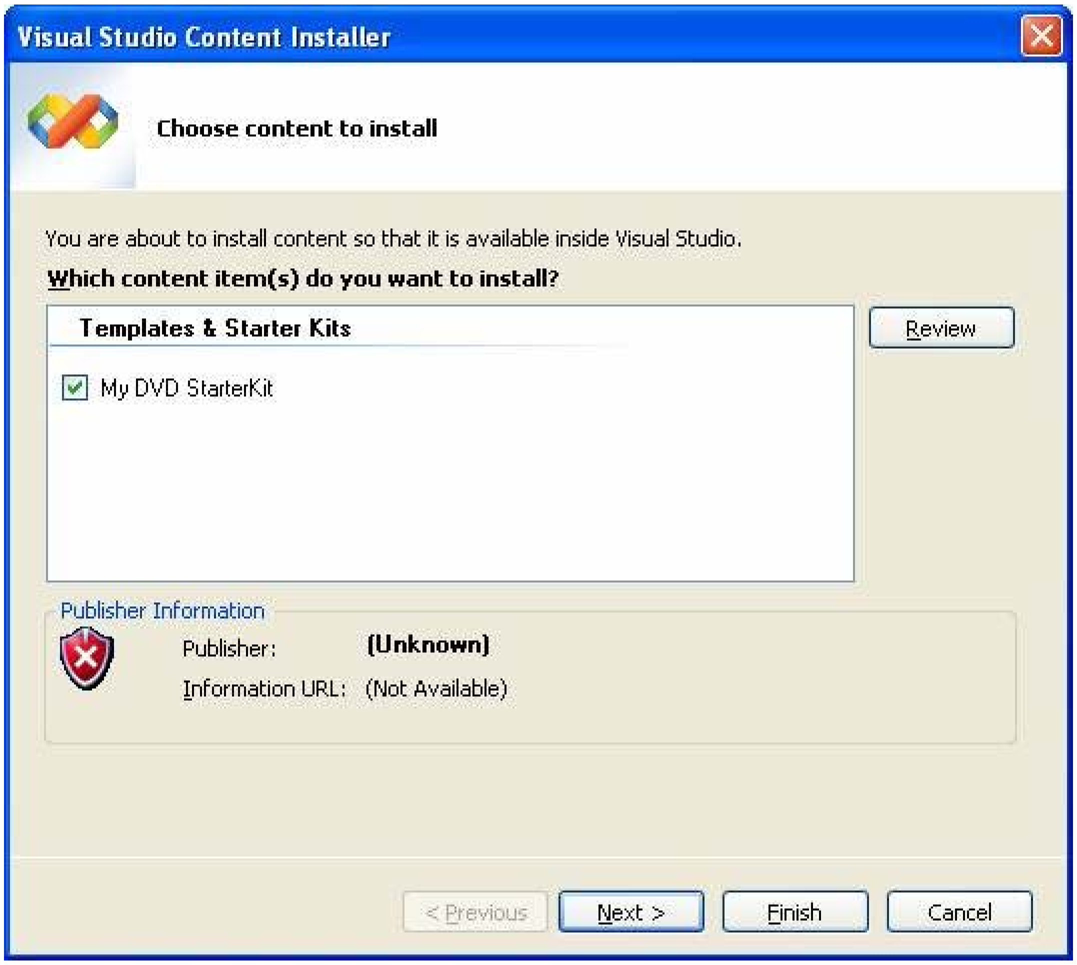
```
<VSContent xmlns="http://schemas.microsoft.com/developer/vscontent/2005">
     <Content>
        <FileName>MyDVDCollection.zip</FileName>
        <DisplayName>My DVD Starter Kit</DisplayName>
        <Description>DVD collection starter kit</Description>
        <FileContentType>VSTemplate</FileContentType>
        <ContentVersion>1.0</ContentVersion>
        <Attributes>
           <Attribute name="TemplateType" value="Project"></Attribute>
            <Attribute name="ProjectType" value="Visual C#"></Attribute>
            <Attribute name="ProjectSubType" value=""></Attribute>
        </Attributes>
    </Content>
</VSContent>
```

iv. Choose **File->Save Xml1.xml As** and browse **C:\Documents and Settings\Administrator\My Documents\Visual Studio 2005\Templates\ProjectTemplates\Visual C#**. Type in **DVD.vscontent** in the **File name** text box. Make sure you select "All Files (*.*) in the **Save as Type** drop down list.
v. **CTRL+ Click DVD.vscontent** and **MyDVDCollection.zip** to select them
vi. **Right click** and select **Send To->Compressed (zipped) Folder**. This will create a DVD.zip file in your folder. (If you are getting errors while executing this step, make sure your Mouse pointer is over the DVD.vscontent file before you right click)
vii. Rename this .zip file to **DVD.vsi**

Step 8 – Using the content installer to install your starter kit

You can now share the.vsi file for the starter kit that you created in **Step 6** with anyone who's got Visual Studio 2005 installed. They just need to double click on the .vsi file to install it.

i. The content installer will install the starter kit in **\My Documents\Visual Studio 2005\Templates\ProjectTemplates\ folder.** Therefore delete the **MyDVDCollection.zip** file from the **\My Documents\Visual Studio 2005\Templates\ProjectTemplates\Visual C#** folder so that you can observe the behaviour of the content installer.
ii. Double click on the **DVD.vsi**. This will bring up the Visual Studio Content Installer wizard.


Visual Studio Content Installer

Choose content to install

You are about to install content so that it is available inside Visual Studio.

Which content item(s) do you want to install?

Templates & Starter Kits

☑ My DVD StarterKit

Publisher Information

Publisher: (Unknown)

Information URL: (Not Available)

iii. **My DVD Starter Kit** will be selected by default in the window and hit **Next.**
iv. For the purpose of this exercise click **Yes** on the **No Signature Found** message box. (This is shown because the .vsi file that you created is not signed. For deploying your .vsi files in a real world scenario you sign your package)

v. On the next page hit **Finish** to complete the installation. (If you see a **Rename Item** dialog box, it may be because you did not delete the **MyDVDCollection.zip** file in Step7.i)

vi. Verify that the installer indicates that the **Installation complete successfully**. Click **Close** to dismiss.

vii. Browse back to **My Documents\Visual Studio 2005\Templates\ProjectTemplates\Visual C#** and verify that **MyDVDCollection.zip** is now present. This template is once again available in the **File->New->Project** dialog under My templates.

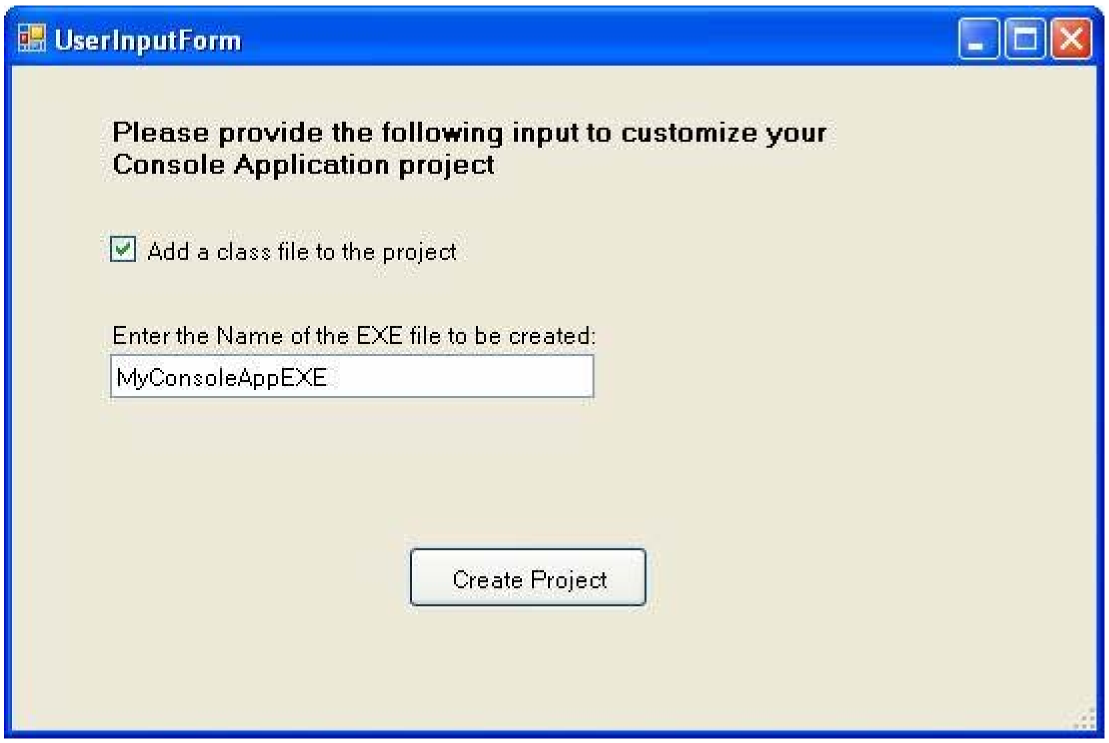
Module 2 Customizing Templates by Creating an IWizard () Implementation

Introduction to IWizard () interface

In VS 2005, a new template architecture called **"VSTemplate"** is introduced. This new architecture and template file format make creating new templates and modifying existing ones very easy. For more information on VSTemplate architecture and how to create you own templates see **"Module 1"** of this lab.

For certain types of templates, you may wish to show a custom UI by running custom code to collect user input which can then be used to customize the project that is being created. This can be achieved by implementing the IWizard () interface currently defined in **Microsoft.VisualStudio.CommonIDE.dll.*****

As an example, the IWizard() implementation you will work with in this module pops up a User Input Form as shown below when a user creates a new project using a user "Console Application" template which calls the IWizard() implementation. This Form asks users to provide two inputs and the resulting "Console Application" project will be customized based on these inputs when the user hits "Create Project" on the following Form.



Please provide the following input to customize your Console Application project

[✓] Add a class file to the project

Enter the Name of the EXE file to be created: MyConsoleAppEXE

***Note:

The dll in which the IWizard() interface is defined is scheduled to be changed for final release of Visual Studio 2005. For the final release, it may not be defined in Microsoft.VisualStudio.CommonIDE.dll

Objectives

This module uses **Visual Studio Automation object model (DTE)** to programmatically interact with various components of VS. If you are not familiar with DTE object model you can find the documentation for it on MSDN but is not necessary to do so to accomplish the tasks in this module. The high level summary of steps you will perform is as follows.

1) Create a project using the IWizard() starter kit included in this lab. At this point you will have a working IWizard() implementation. The IWizard() starter kit is designed to perform two different customizations on a "Console Application" project based on user input. One of the customizations is already implemented and you will implement the second customization. The details of the customizations are provided in **STEP 1.V**.
2) Create a "Console Application" template that calls this IWizard() implementation.
3) Create a project using the "Console Application" template to see the IWizard() implementation getting called and perform the customization.
4) Understand the code for one of the customizations.
5) Implement the second customization and see it in action.

STEP 1 – Create a project using the IWizard() Starter Kit.

i. First let's perform some steps to ensure that we start in a clean state and are not affected by the templates created by previous users of this lab. For this:

    a. If you already don't have an instance of VS 2005 IDE running, launch VS 2005 (There is a shortcut for this on the desktop.)
    b. Once the IDE loads, using My Computer or Windows Explorer browse to the location:
    My Documents\Visual Studio 2005\Templates\Project Templates\Visual C#.

       This is the location where user templates for C# are stored.

    c. Delete any template zip files you find in this folder.

ii. Now, copy the IWizard() starter kit template zip file called **IWizardStarterKit.zip**, from **C:\HOL084\Module2** to user template location: **My Documents\Visual Studio 2005\Templates\Project Templates\Visual C#.**

iii. From within the IDE, choose **File -> New -> Project**. Select the **"Visual C#"** node in the New Project dialog. You will see a template called **"IWizardStarterKit"** under **"My Templates"** section in new project dialog. Select this template.

iv. Change the project location to **C:\VSTemplates<yourfirstname>\**. Change the name of your project to **MyIWizard<yourname>**. Create the project.

v. Let's understand the project contents a little bit. The project is essentially a class library project and it contains a class in the file called **IWizardImplementation.cs** which implements the IWizard() interface. The code in **IWizardImplementation.cs** file shows up a Form as shown before called **UserInputForm** which asks the user for the following two inputs for customization.

   a. The user can choose whether or not a class file should be added as an item to the project or not. This serves as an example of adding items to the project dynamically based on user input using the DTE.
   b. The user can provide the name for the EXE file that gets created when the project is built. With this customization the user can choose to name the EXE file different than the project name. This serves as an example of accessing and changing project properties through DTE based on user input.

   Once the user provides the above input, we use DTE to perform these customizations on the Console Application project that is getting created.

vi. Build the project. To do this, choose the **MyIWizard<yourname>** node in **Solution Explorer** in the IDE, **right click** and select **Build option** in the **context menu.**

VS requires that the dll containing the IWizard() implementation be in one of four designated location. One such location is the VS installation folder that contains devenv.exe. The project contains a post-build event which copies the output dll for the project in the folder containing devenv.exe in addition to creating the output dll in the project bin folder. Verify that you have a dll called **MyIWizard<yourname>.dll** in the VS installation folder: **C:\Program Files\Microsoft Visual Studio 8\Common7\IDE.**

At this point you have a working IWizard() implementation dll and it is in the right location so that it can be called by any template.

STEP 2- Creating a Console Application Template that calls the IWizard() implementation

i. Once you create an IWizard() implementation you need to create a template, the .vstemplate file which contains the following <WizardExtension> section.*** This section in the .vstemplate file points VS to the appropriate IWizard() dll you created in **STEP 1.**

```
<WizardExtension>

<Assembly>MyIWizard<yourname>.dll</Assembly>
<FullClassName>MyIWizard<yourname>.WizardImplementation</FullClassName>

</WizardExtension>
```

ii. For your convenience, the IWizard() starter kit you used to create the implementation does contain a template **"MyConsoleApplicationTemplate"** in an un-zipped form. Look for a folder **"MyConsoleApplicationTemplate"** in **Solution Explorer** under the node for your project**.** Within this folder look for a file **MyTemplate.vstemplate**. **Double-click** on this file in **Solution Explorer** to open it in VS XML Editor. This file should have the <WizardExtension> section as shown above.

***Note:

For security reasons, the final release of VS 2005 will require you to include a fully qualified strong name for your IWizard() assembly in the <WizardExtension> section as follows. Unqualified names as shown above will not be permitted.

<Assembly>MyIWizard<yourname>.dll, Version=8.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a</Assembly>

iii. Now, you need to zip this template up and put it in the user template location. To do this,

a. Browse to **C:\VSTemplates<yourfirstname>\MyIWizard<yourname>\MyIWizard<yourname>\MyConsoleApplicationTemplate.**

b. Once inside **MyConsoleApplicationTemplate**, hit **Ctrl-A** to select all the files, right click and choose **Send to -> Compressed (zipped) Folder.** Re-name the zip file created to **MyConsoleApplicationTemplate.zip.**

c. Copy this zip file to the user template location: **My Documents\Visual Studio 2005\Templates\Project Templates\Visual C#.**

STEP 3 –Create a project using the Console Application Template and see the IWizard() implementation in action.

i. Launch another instance of VS IDE in addition to the instance that contains your IWizard() implementation project **MyIWizard<yourname>.**

ii. Select **File -> New -> Project.**

iii. Select the **C# node** and you should see the **"MyConsoleApplication**" template available in the **"My Templates"** section along with the IWizardStarterKit template. Select the **MyConsoleApplication** template.

iv. Change the location field to **C:\VSTemplates<yourfirstname>** location and hit **OK** to create the project.

   You should see the UserInput form shown in the introduction section of the lab. The IWizard() implementation that is referenced in the Console Application template launches this Form.

v. With the default values selected in the Form, hit **"Create Project".** You will see that the Console Application project that is created will contain a file Class.cs in the project because you selected to include a class file.

   Remember the second customization which renames the EXE is not coded up yet and hence when you build this project instead of creating an EXE called **"MyConsoleAppEXE"** it will create an EXE with the project name. We will implement the code for this as part of STEP 5.

vi. You can create another project using the same Console Application template and this time choose not to include a Class file and verify that the class file does not get included in the project.

STEP 4 – Understanding the Code

The WizardImplementation class in the IWizardImplementation.cs file implements the IWizard() interface. Two of the methods defined in IWizard() interface are implemented in the class: The RunStarted() method and ProjectFinishedGenerating()method.

The RunStarted() method gets called when the user instantites the template before the project is created from the template. Hence collection of user input should be done here. The RunStarted() method shows the UserInput Form to the user and collects the input for the two customizations outlined above. The rest of the code in the implementation is about performing the customizations which are performed after the project is created in the ProjectFinishedGenerating() method.

The first customization performed in the ProjectFinishedGenerating() method is to add a class file based on user's input. The ProjectFinishedGenerating() method is called after the template engine creates a project from the template. And the method is then called with the newly created DTE project object as an argument. Hence, from this object you can use project.ProjectItems.AddFromTemplate() method to add additional items to the project based on user input.

STEP 5 – Implement the other Customization

The customization that we want to do is to rename the EXE that gets created for the Console Application project based on the user's input. This is an example of changing Project properties on the fly through the IWizard() implementation.

i. The code for this is very simple. Switch to the IDE instance which contains your IWizard() project, **MyIWizard<yourname>.** In the file **IWizardImplementation.cs**, inside method ProjectFinishedGenerating you will see comments for placing the code for #Customization 2. Insert the following lines of code there.

```
// Customization #2. Changing a project property based on user's
input.
// Please insert the code for customization #2 here.
EnvDTE.Property p = project.Properties.Item("AssemblyName");
p.Value = exeName;
```

ii. Now, rebuild your IWizard()project, so that the dll in the **devenv.exe folder** gets updated. To do this, choose the project node in Solution Explorer in the IDE, **right click** and select **Rebuild*****

iii. To verify the above customization works, create another project in the other instance of the IDE using the "**MyConsoleApplication"** template as described in STEP 3. This time when you build the Console Application project created, the EXE that is generated should be named **"MyConsoleAppEXE"** rather than the name of the project itself.

iv. Launch the Project Properties Designer by **right clicking** on the node for your project in Solution Explorer and selecting **properties**. You can verify in the **Application** page of the Project **Properties Designer** that the **"AssemblyName"** property is set to **"MyConsoleAppEXE"** instead of the name of the project.

***Note:

If you see a build error during this step, make sure that you close all other instances of the VS 2005 IDE and try the build action again. This error may appear due to a known Beta 2 issue.

Appendix – VSTemplate File Format

VSTEMPLATE File Format for Project Templates

.VSTEMPLATE project templates consist of three fundamental elements:

- Identity Descriptor
- TemplateData
- TemplateContent

Example:

```
<VSTemplate Type=”Item” Version="2.0.0">
    <TemplateData>
        …
    </TemplateData>
    <TemplateContent>
        …
    </TemplateCntent>
    <WizardExtension>
        …
    </WizardExtension>
</VSTemplate>
```

Identity Descriptor

This element identifies the template and specifies its version.

Example:

```
<VSTemplate Type=”Project” Version="2.0.0">
    …
</VSTemplate>
```

Description:

Element	Required	Default	Attributes	Description
VSTemplate	Yes	n/a	Version. Format= “n.n.n” TemplateType=”Project” or “ProjectGroup” Any version of format “2.n.n” will work with VS2005.	Identifies the file as a project template

TemplateData

This element contains meta-information about the template which is used to categorize it and define its display characteristics in the New Project dialog. Localizable/embedded information such as the template name and icon can optionally be specified in the file (as a string) or can be contained in a VS Package which the template refers to. Other options control how the dialog appears, whether a base name is provided, what the sort order is in the dialog and other display options.

Simple Example:

Defaults will be used for all elements not specified, so the .vstemplate file can be very simple.

```
<TemplateData>
        <Name>WindowsApplication</Name>
        <Description>A windows user interface application</Description>
        <Icon>WindowsApplication.ico</Icon>
        <ProjectType>CSharp</ProjectType>
        <SortOrder>10</SortOrder>
        <DefaultName>WindowsApplication</DefaultName>
</TemplateData>
```

Advanced Example#1:

Alternatively the user can specify elements to get the dialog to appear as desired.

```
<TemplateData>
        <Name>WindowsApplication</Name>
        <Description>A windows user interface application</Description>
        <Icon>WindowsApplication.ico</Icon>
        <ProjectType>CSharp</ProjectType>
        <ProjectSubType>Windows</ProjectType>
        <SortOrder>10</SortOrder>
        <CreateNewFolder>false</CreateNewFolder>
        <ProvideDefaultName>false</ProvideDefaultName>
        <PromptForSaveOnCreation>false</PromptForSaveOnCreation>
        <EnableEditOfLocationField>true</EnableEditOfLocationField>
        <EnableLocationBrowseButton>false</EnableLocationBrowseButton>
</TemplateData>
```

TemplateData Description

Element	Required	Default	Attributes	Description
TemplateData	Yes			Element containing the metadata for the template
Name	Yes		Package – the VS Package GUID ID – the ID of the string resource	Specifies the project name to appear in the NPD[1] as a string.
Description	Yes		Package – the VS Package GUID ID – the ID of the string resource	Specifies the description to appear in the NPD as a string.
Icon	Yes		Package – the VS Package GUID ID – the ID of the icon resource	Specifies the icon to appear in the NPD as a string.
ProjectType	Yes			The type of the template. Possible values include: • CSharp • VisualBasic • JSharp • VisualC
ProjectSubType	No			The type of the template. Possible values include: • Windows • Office • Database • Smart Devices • Web

[1] NPD – abbrev. for New Project Dialog

SortOrder	No	100		Template order in NPD. For MS and VSIP authored templates only. User templates have mandatory alphabetical sort order.
CreateNewFolder	No	True		Whether a containing folder is created on instantiation
DefaultName	No			The DefaultName in the Name file in the NPD, ex: ClassLibrary
ProvideDefaultName	No	False		Whether to provide a default DefaultName for the project in the name field
AllowCreationWithoutSave	No	True		Whether the project supports being a 'temporary' in-memory project
EnableLocationBrowseButton	No	True		Whether the user can browse to a different directory to create the solution
Hidden	No	False		Specifies that the template should not appear in the New Project Dialog. If specified, no other elements inside <TemplateData> are required
NumberOfParentCategoriesToRollUp	No	0		Display the template in parent categories (roll up display). Not respected for user templates.

Web Projects

Web projects are unusual in a few ways. First, web projects appear in a separate dialog, the Add New Website dialog. Likewise, Web Project Items appear in the Add New Web Item dialog. In these dialogs, items are them subcategorized by language. To get items to appear in this dialog, use the following logic. Set <ProjectType> to Web and set <ProjectSubType> to Language. Note this is the reverse of normal configuration, where <ProjectType> is set to language and <ProjectSubType> is set to the technology type (e.g. Windows, Office).

Example:

```
<VSTemplate Version="1.1.1" Type="Project">
  <TemplateData>
    <Name Package="{39c9c826-8ef8-4079-8c95-428f5b1c323f}" ID="3326" />
    <Description Package="{39c9c826-8ef8-4079-8c95-428f5b1c323f}" ID="3327"/>
    <Icon Package="{39c9c826-8ef8-4079-8c95-428f5b1c323f}" ID="4701"/>
    <ProjectType>Web</ProjectType>
    <ProjectSubType>CSharp</ProjectSubType>
    <SortOrder>30</SortOrder>
    <CreateNewFolder>true</CreateNewFolder>
    <DefaultName>WebSite</DefaultName>
    <ProvideDefaultName>true</ProvideDefaultName>
    <PromptForSaveOnCreation>true</PromptForSaveOnCreation>
    <EnableEditOfLocationField>true</EnableEditOfLocationField>
    <EnableLocationBrowseButton>true</EnableLocationBrowseButton>
    <LocationField>hidden</LocationField>
  </TemplateData>
```

Second, in the new web project system, web projects no longer have a project file. Therefore web project templates only need to contain a dummy project file. This allows them to be handled by the standard template engine and still create a website. Therefore a website template would look like this.

Example:

```
      <TemplateContent>
        <Project File="PersonalWebSite.webproj" ReplaceParameters="true">
          <ProjectItem ReplaceParameters="true" OpenInEditor="true"
OpenOrder="10">Albums.aspx</ProjectItem>
          <ProjectItem ReplaceParameters="true">Albums.aspx.cs</ProjectItem>
          <ProjectItem ReplaceParameters="false">Default.aspx</ProjectItem>
          <ProjectItem ReplaceParameters="true">Default.aspx.cs</ProjectItem>
          <ProjectItem ReplaceParameters="true">Default.master</ProjectItem>
          <ProjectItem ReplaceParameters="true">Default.master.cs</ProjectItem>
          ...
        </Project>
      </TemplateContent>
    </VSTemplate>
```

Finally, in some cases a user may want to create an item. If you wish to create an item which has code-behind and you want to place the code in the /code directory, you will need to do one special step. You can specify that a file should be dropped in a root directory by specifying placing a "\" in front of the path, e.g. "\code\login.aspx". Otherwise if you leave the slash off, e.g. "code\login.aspx", then the file will be placed in the /Code directory.

```
      <TemplateContent>
          <ProjectItem ReplaceParameters="true">Login.aspx</ProjectItem>
          <ProjectItem ReplaceParameters="true">\code\Login.aspx.cs</ProjectItem>
          ...
      </TemplateContent>
    </VSTemplate>
```

Using VS Packages for resources

The <Name>, <Description> and <Icon> identifiers can either be provided with the template, like this:

```
        <TemplateData>
                <Name>WindowsApplication</Name>
                <Description>A windows user interface application</Description>
                <Icon>WindowsApplication.ico</Icon>
        ...
        </TemplateData>
```

Or can point off to a VS package which contains the localized strings and the icon file, like this:

```
        <TemplateData>
                <Name Package="{164B10B9-B200-11D0-8C61-00A0C91E29D5}" ID="3000" />
                <Description Package ="{164B10B9-B200-11D0-8C61-00A0C91E29D5}" ID="3001" />
                <Icon Package="{164B10B9-B200-11D0-8C61-00A0C91E29D5}" ID="4500" />
        ...
        </TemplateData>
```

Hidden Templates

Some template creators have the requirement that they be able to create templates which exist on disk but which do not appear in the New Project or Add New Item dialogs. The <Hidden> element can be used to hide these templates from view. When used, no other elements are required. This element can be specified for project templates or item templates. It is most commonly used for item templates which are hidden from view but accessed programmatically by different functions in the IDE to add project items when required, e.g. to add a .RESX file.

```
<TemplateData>
        <Hidden>True<Hidden>
</TemplateData>
```

Sort Order

The SortOrder tag, which determines the Sort Order in the New Project dialog (i.e. which template comes 1st, 2nd, 3rd, etc.). This tag is only used for internally (VS or VSIP) produced templates. For any templates which are identified as user-authored (i.e. not installed under Program Files) this tag will be ignored and the template will be sorted alphabetically.

```
<TemplateData>
        ...
        <SortOrder>10</SortOrder>
        ...
</TemplateData>
```

Getting templates to appear in a root category

Some template creators for Visual Studio-shipped templates will want these templates to not only appear in a category but also to appear under a root category- for instance, C# would like the Class Library template to appear under Windows templates, but also under general C# templates. To enable this, add the <DisplayInParentCategories> tag to your template. This tag is only used for internally (VS or VSIP) produced templates. For any templates which are identified as user-authored (i.e. not installed under Program Files) this tag will be ignored and the template will only appear in per its <ProjectType> and <ProjectSubType>.

```
<TemplateData>
        <DisplayInParentCategories>True</DisplayInParentCategories>
</TemplateData>
```

TemplateContent

This element defines the content for the template. It consists of a project which contains project items

Simple Example:

```
    <VSTemplate Type=”ProjectGroup”  Version=”2.0.0” />
        <TemplateContent>
                <Project File=”Application.csproj” ReplaceParameters=”true”>
                        <ProjectItem ReplaceParameters="true">Form.cs</ProjectItem>
                        <ProjectItem>File.cs</ProjectItem>

                </Project>
        </TemplateContent>
```

TemplateContent Description

Element	Required	Default	Attributes	Description
TemplateContent	Yes			Element containing the content for the template
ProjectCollection	No			A collection of links to other templates. Used for multi-project templates.
ProjectTemplateLink	No		ProjectName	A link to another project template (by .vstemplate file) to use in the collection.
Folder	No		Name	Containing solution folder. If specified a folder will also be created on disk.
Project			File TargetFileName ReplaceParameters	The project file for the template. May be multiple.
ProjectItem	Yes		TargetFileName ReplaceParameters OpenInEditor OpenOrder OpenInWebBrowser OpenInHelpBrowser	An item in the project. NOTES: OpenInHelpBrowser exists to open HTML files and text files which are local to the project. We should always open these files in the help browser. None of the following are supported for OpenInWebBrowser or OpenInHelpBrowser: - opening any non-HTML or non-text file - supporting OpenOrder - opening external (http:// address) websites If OpenInHelpBrowser is present the OpenInEditor and OpenOrder tags should be ignored.
CustomParameters	No		Name	Any custom parameters which should be passed to the wizard when it is run to do parameter replacement.
CustomParameter	No		Name Value	A custom parameter (name/value pair) to pass to the wizard

Folders

The <Folder> element is no longer required for specifying folders inside of projects. Users can ZIP up projects containing folders. The following is an example of how items for a project with folders should be specified in a .vstemplate file. In this example, the user would create a project containing folders. On save, the project will contain physical folders on disk. The user can compress the project as-is. On project create from a template, the folder structure will be recreated based on the template project.

Example:

```
        <TemplateContent>
                <Project File=”WindowsApplication.csproj” ReplaceParameters=”true”>
                        <ProjectItem ReplaceParameters="true">File1.cs</ProjectItem>
                        <ProjectItem>Icon.ico</ProjectItem>
                        <ProjectItem>Web
References\net.xmethods.www\Reference.cs</ProjectItem>
                        <ProjectItem>Web
References\net.xmethods.www\Reference.map</ProjectItem>
                        <ProjectItem>Web
References\net.xmethods.www\TemperatureService.wsdl</ProjectItem>
                </Project>
        </TemplateContent>
```

However, if desired a user can use the <Folder> tag to specify either Solution Folders (folders which contain projects) or project folders (folders which contain items). More information on the latter is below.

Multi-Project Templates

In Beta2 Project Templates will support multi-project templates. This is done by specifying all the Projects. Note that this template can be used either to create new projects or to add projects to an existing solution. A template can only be a container template or a single project template. It can contain links to several other projects or a project. It cannot contain both. The user can specify a solution folder structure in the XML. If specified the project wizard will create a folder structure on disk mimicking the solution folder structure. The projects created will be named according to whatever the project file is named in the template. This means that what the user enters in the Name field on the New Project Dialog will be ignored.

Note that in the ZIP file, the ZIP file should be structured to place all sub-templates in folders, with the root vstemplate at the too

Example:

```
<TemplateData>
                ...
</TemplateData>
<TemplateContent>
   <ProjectCollection>
        <ProjectTemplateLink
        ProjectName="Project1"> Project1\MyTemplate.vstemplate</ProjectTemplateLin>
        <ProjectTemplateLink
        ProjectName="Project2"> Project2\MyTemplate.vstemplate</ProjectTemplateLin>
   </ProjectCollection>
</TemplateContent>
```

Folder structure:

```
ProjectCollection.vstemplate
|_Project1Folder
|   |_Project1.vstempalte
|   |_other files
|_Project2Folder
|   |_Project2.vstempalte
|   |_other files
|_Project3Folder
    |_Project3.vstempalte
    |_other files
```

Note that the <SolutionFolder> element can be used to specify solution folders which contain projects.

Example:

```
<TemplateData>
                ...
</TemplateData>
<TemplateContent>
  <ProjectCollection>
    <SolutionFolder Name="ClientUI" >
        <ProjectTemplateLink
          ProjectName="Project1">Project1\MyTemplate.vstemplate< /ProjectTemplateLin>
        <ProjectTemplateLink
          ProjectName="Project2"> Project2\MyTemplate.vstemplate</ProjectTemplateLin>
    </SolutionFolder>
  </ProjectCollection>
</TemplateContent>
```

Setting the name of a file using a parameter

In some cases a user may want to set the name of a file based on an inputted parameter. This is supported using the <TargetFileName> element. The following is an example of how to rename a file contained in a project.

Example:

```
                <Project>
                        <ProjectItem TargetFileName= “$projectname$.exe”>File1.exe</ProjectItem>
                        ...
                </Project>
```

Opening a file, including opening in a HTML or Help Viewer

In some cases a user may want to open a specific file on project create such as a code file or HTML documentation file. The following are examples of how to do so:

Open Code File Example:

```
   <TemplateContent>
      <Project File=”ClassLibrary.vbproj” ReplaceParameters=”true”>
         <ProjectItem ReplaceParameters=”true” OpenInEditor=”true”
OpenOrder=”10”>Class1.vb</ProjectItem>
          ...
          </Project>
   </TemplateContent>
```

Open HTML File in Help Browser Example:

```
   <TemplateContent>
      <Project File=”ClassLibrary.vbproj” ReplaceParameters=”true”>
         <ProjectItem ReplaceParameters=”true” OpenInEditor=”true” OpenOrder=”10”
                   OpenInHelpBrowser=”true”>Class1.vb</ProjectItem>
          ...
          </Project>
   </TemplateContent>
```

VSTEMPLATE File Format for Item Templates

.VSTEMPLATE item templates consist of three fundamental elements:

- Identity descriptor
- TemplateData
- TemplateContent

Example:

```
<VSTemplate Type=”Item” Version="2.0.0">
      <TemplateData>
            …
      </TemplateData>
      <TemplateContent>
            …
      </TemplateContent>
</VSTemplate>
```

Identity Descriptor

This element identifies the template and specifies its version.

Example:

```
<VSTemplate TemplateType=”Item” Version="2.0.0">
      <TemplateData>
            …
      </TemplateData>
      <TemplateContent>
            …
      </TemplateContent>
</VSTemplate>
```

Description:

Element	Required	Default	Attributes	Description
VSTemplate	Yes	n/a	Version. Format= “n.n.n” TemplateType=”Item” Any version of format “2.n.n” will work with VS2005.	Identifies the file as a project item template

TemplateData

This element contains meta-information about the template which is used to categorize it and define its display characteristics in the Add New Item dialog.

Example:

```
            <TemplateData>
                        <Name>Class</Name>
                        <Description>An empty class file</Description>
                        <Icon>Class.cs</Icon>
                        <ProjectType>CSharp</ProjectType>
                        <SortOrder>30</SortOrder>
                        <DefaultName>Class.cs</DefaultName>
                        <AppendDefaultExtension>true</AppendDefaultExtension>
```

```
        </TemplateData>
```

TemplateData Description

Element	Required	Default	Attributes	Description
TemplateData	Yes			Element containing the metadata for the template
Name	Yes		Package – the VS Package GUID ID – the ID of the string resource	Specifies the project name to appear in the NPD[2] as a string.
Description	Yes		Package – the VS Package GUID ID – the ID of the string resource	Specifies the description to appear in the NPD as a string.
Icon	Yes		Package – the VS Package GUID ID – the ID of the icon resource	Specifies the icon to appear in the NPD as a string.
ProjectType	Yes			The type of the template. Possible values: • CSharp • VisualBasic • JSharp • VisualC • General
ProjectSubType	No			The type of the template. Possible values include: • Windows • Office • Database • Smart Devices • Web
SortOrder	Yes			Template order in ANID. For MS and VSIP authored templates only. User templates have mandatory alphabetical sort order.
DefaultName	Yes			The DefaultName in the Name file in the ANID, ex: Class
AppendDefaultExtension	Yes	True		If set to true, will create the target new file(s) with the same extension as the source files, ignoring what the user may have typed in the Add New Item dialog. (I.e. If user is creating C# class file and types "Foo.vb", will be created as "Foo.cs"
LocationField	No	Enabled		Whether the location field is enabled, disabled or hidden. Values (restricted): • Enabled • Disabled • Hidden
SupportsMasterPage	No			Whether the template supports having a master page (Web option)

[2] NPD – abbrev. for New Project Dialog

SupportsCodeSeparation	No			Whether the template supports code separation (Web option)
SupportsLanguageDropdown	No			Whether the template is identical for multiple languages and a language dropdown should appear.
Hidden	No	False		Specifies that the template should not appear in the New Project Dialog. If specified, no other elements inside <TemplateData> are required
DisplayInParentCategories	No	False		Display the template in parent categories (roll up display). Not respected for user templates.

Hidden Templates

See above for notes on how to use the <Hidden> element.

TemplateContent

This element defines the content for the template. It consists of 1 subelement, the ProjectItem.

Simple Example:

```
        <TemplateContent>
                <ProjectItem> Class1.cs</ProjectItem>
        </TemplateContent>
```

Advanced Example:

This example shows a template where references are being added, a Form file is being set to have a Form subtype so that it will open properly in Form view in the Editor, and parameter replacement (replacing the class name) is taking place.

```
        <TemplateContent>
                <References>
                        <Reference>

<Assembly>System,Version=1.0.0.1,Culture=neutral,PublicKeyToken=9b35aa32c18d
                                4fb1</Assembly>
                        </Reference>
                        <Reference>
                                <Assembly>System.Windows.Forms,Version=1.0.0.1,Culture=neutral,
PublicKeyToken
                                  =8c35aa32c18d4fb2</Assembly>
                        </Reference>
                </References>
                <ProjectItem ReplaceParameters="true" SubType="Form"> Form1.cs</ProjectItem>
        </TemplateContent>
```

TemplateContent Description

Element	Required	Default	Attributes	Description
TemplateContent	Yes			Contains all the items in the project.
References	No			Any references which should be added to a project when the item is added

Reference	No			The reference to be added. Includes: • Assembly
Assembly	No			Required if reference element is specified. Assembly element supports both simple text and strong name assembly references: <assembly>System</assembly> <assembly>System,Version=1.0.0.1, Culture=neutral, PublicKeyToken=9b35aa32c18d4fb1</assembly>
ProjectItem	Yes		ReplaceParameters SubType TargetFileName	A project item. SubType: Used for multi-file item templates when an item has a SubType which sets how it should be opened in the Editor. Specific to managed project system-based templates. Sets the SubType for the item in the Project file. <Compile Include="Form1.cs"> <SubType>Form</SubType> </Compile>
CustomParameters	No		Name	Any custom parameters which should be passed to the wizard when it is run to do parameter replacement.
CustomParameter	No		Name Value	A custom parameter (name/value pair) to pass to the wizard

Custom Parameters

Note that the customparameters section can be used to specify special custom parameters which you wish to pass to the wizard. This can be used to define new parameter substitutions which are not pre-defined below.

Example:

```
<TemplateContent>
   ...
   <CustomParameters>
       <CustomParameter Name="$username$" Value="CraigSkibo"/>
   </CustomParameters>
</TemplateContent>
```

Using User Input (Parameter Substitution)

All templates support parameter substitution to enable replacement of key parameters such as class names, namespaces, etc on template instantiation. If the <ReplaceParameters> tag is true on any artifact in a template, the template wizard will perform parameter substitution. The format for parameters is $[parameter]$. Examples:

$safeprojectname$
$safeclassname$
$guid1$
$guid5$

The library of available parameter replacement options is the following. This library will be user-extensible in future releases. Replacement parameters are case-sensitive.

Parameter	Description
itemname	The user-provided name of an item for an item template being added to a project. Used to replace classnames in Class file templates Form file templates, etc, i.e. Public Class Class1
safeitemname	The user-provided name of the item with all unsafe characters and spaces removed.
saferootitemname	The name of the root item. Use for multi-file items when the root item name (the user-entered name) should be used for all parameter replacements, e.g. for Form templates which contain Form.cs and Form.desginer.cs.
guid [1-10]	A guid. Used to replace the project GUID in a project file. User can specify up to 10 unique GUIDs. Example: guid1
projectname	The user-provided name of the project for a project template.
safeprojectname	The user-provided name of the project for a project template with all unsafe characters and spaces removed. Used to replace the namespace and the name of the main class in some templates, i.e. NameSpace WindowsApplication1.
rootnamespace	The root namespace of a project for an item template being added to a project. Used to replace the namespace in the item file, i.e. Namespace WindowsApplication1
time	The current time (Format: DD/MM/YYYY 00:00:00).

Parameter	Description
year	The current year (Format: YYYY).
username	The current username (Ex: CraigS).
userdomain	The current user domain (Ex: REDMOND).
machinename	The current machinename (Ex: VISUALSTUDIO1).
clrversion	Current version of the CLR
registeredorganization	The registry key value from: HKLM\Software\Microsoft\Windows NT\CurrentVersion\RegisteredOrganization. Used for the company name field in AssemblyInfo.vb.
wizarddata	The XML payload, if any, of the <WizardData> element. This will be passed as a single string.

Note that the customparameters section for the VSTEMPLATE file can be used to specify special custom parameters which you wish to pass to the wizard. This can be used to define new parameter substitutions which are not pre-defined.

Example:

```
<TemplateContent>
   …
   <CustomParameters>
       <CustomParameter Name=”$username$” Value=”CraigSkibo”/>
   </CustomParameters>
</TemplateContent>
```