

WCM401 Lab Manual

Developing Publishing Sites with SharePoint Server 2007 Web Content Management

Developer Lab Exercises

- 1) Windows SharePoint Services Primer
- 2) MOSS 2007 & Web Content Management Overview
- 3) Authentication & Authorization
- 4) Creating Master Pages & Configuring Navigation
- 5) Creating Custom Page Layouts
- 6) Extending the Out-Of-The-Box Authoring Experience
- 7) Custom Field Types & Controls
- 8) Leveraging Publishing & Custom Web Parts
- 10) Understanding Workflow Foundation & Creating Custom Workflows
- 12) Implementing Multilingual Sites Using Variations

Lab 01: Windows SharePoint Services 3.0 Development Primer

Lab Time: 45 minutes

Lab Overview: This lab should serve as an overview to working with the SharePoint core object model within Windows SharePoint Services (WSS) 3.0 and the Feature framework. If you are new to WSS, you should first take about 10 minutes to click around a WSS Team Site and explore what you get out-of-the-box. However, the primary goal of this lab is for you to create a WSS Feature, work with the object model and deploy your code using the WSS solution package framework. In this lab, after creating a new site collection, you will create a Feature in Visual Studio that provisions a new page into an existing WSS Team Site based off a template. The Feature will also create a new menu item in the Quick Launch navigation to access the new page. The new page contains a text box and a button for you to create new task items. Upon clicking the button on the page, a new list item is created in the site's Tasks list.

Exercise 1: Provisioning a new Site Collection

In this exercise you will create a new WSS 3.0 Site Collection within an existing Web Application.

1. If you haven't already, make sure you are logged in as **LITWAREINC\Administrator** (with a password of **pass@word1**).
2. Launch Central Administration by selecting **Start » All Programs » Microsoft Office Server » SharePoint 3.0 Central Administration**.
3. From the **Central Administration** site, select the **Application Management** tab and then select **Create site collection** under the **SharePoint Site Management** section.
4. On the **Create Site Collection** page, complete the required information as follows and click OK to create a new site collection:
5. **Web Application:** <http://litwareinc.com>
 - **Title:** Lab 1
 - **Description:** Lab 1 description
 - **Web Site Address:** <http://litwareinc.com/sites/lab1>
 - **Template Selection:** Team Site
 - **Primary Site Collection Administrator:** LITWAREINC\administrator

*Make sure to click the icon or press **[CTRL]+[K]** to validate the account... this may take a second or two to resolve. It should resolve to **Litware Admin Guy**.*

6. When the site collection process is complete, you will be redirected to a page with a link to the new site collection. Click the link to open the site in a new window.

At this point, if you've worked with WSS 3.0 sites before, skip this step... otherwise if this is the first time you have created a WSS 3.0 Team Site, feel free to take a few minutes to poke around. Specifically, go into the site settings page (Site Actions » Site Settings) and change the title and description of the site since they aren't important for this lab.

Exercise 2: Creating a new WSS 3.0 Feature that provisions a new page

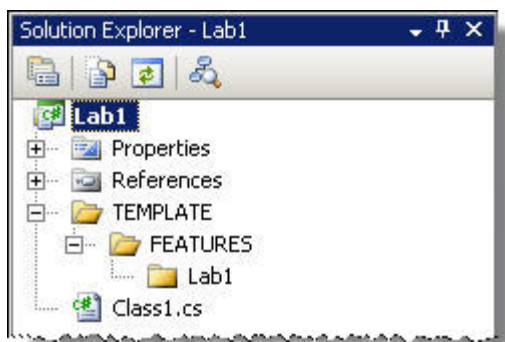
In this exercise you will create a new WSS 3.0 Feature that will be deployed using the WSS solution package framework. Once the Feature has been installed, upon activation, it will create a new page within the site collection created in Exercise 1 based off a template that will be included with the Feature.

1. Open **Visual Studio** and create a new **C# Class Library** project named **Lab1**. You should create the project within the following path so the project files will reside within the same directory structure as all the other lab exercises:

If you prefer, feel free to create a VB.NET project for this lab, or all labs in this course. All the code samples are in C# but nothing requires C#.

c:\Student\Labs\01_WssPrimer\Lab

2. Now you need to create the folder structure that will contain the Feature. The Feature you will create is going to live within the Features folder nested within SharePoint's 12 folder. To make your life easier, create a mirror of the SharePoint 12 folder structure from the **TEMPLATE** folder on down. The following image shows what your project should look like after creating the necessary folder structure:



3. Now create an ASPX page that will be used as the template for the page instance that the **Lab1** Feature will create. Because the Visual Studio Class Library project template is not configured to allow for the creation of ASPX pages, you will have to create the page manually. Create a new text file inside the **Lab1** directory named **PageTemplate.aspx**.

Add the following code to the **PageTemplate.aspx** page:

```
<%@ Page Language="C#"
    MasterPageFile="~/masterurl/default.master"
    meta:progid="SharePoint.WebPartPage.Document" %>

<asp:Content runat="server" ContentPlaceHolderID="PlaceHolderMain">
    <h1>Task Creator Page</h1>
```

```
<b>Task Title:</b><br />
<asp:TextBox ID="txbTaskTitle" runat="server" /><br />
<asp:Button ID="btnCreateTask" runat="server" Text="Create Task" />
</asp:Content>
```

At this point the page will simply display an ASP.NET 2.0 `TextBox` and `Button`, but nothing happens when it's clicked. You'll wire code up to the page in Exercise 4.

4. Create an XML file inside the **Lab1** directory named **feature.xml**. This file contains the definition of the Feature itself. Add the following XML to the **feature.xml** file:

```
<?xml version="1.0" encoding="utf-8" ?>
<Feature xmlns="http://schemas.microsoft.com/sharepoint/"
  Id="872BA9EF-B5B2-4562-9861-B4D14415FCCB"
  Title="Lab 1 - Working with Provisioned Pages"
  Scope="Web"
  Hidden="False"
  Version="1.0.0.0">

  <ElementManifests>
    <ElementManifest Location="elements.xml" />
    <ElementFile Location="PageTemplate.aspx" />
  </ElementManifests>

</Feature>
```

Note the two elements created within the `<ElementManifest>` XML element. The first one (`<ElementManifest>`) will contain the XML to create a new page instance based off a template. The second one (`<ElementFile>`) is important only to WSS solution package deployment. By listing all files that are not element manifests as `<ElementFiles>`, you won't have to manually deploy each and every file within the solution when you create the solution manifest file later in this exercise.

5. Next, create another XML file in the same **Lab1** directory named **elements.xml**. This file contains the XML that will create a page instance based off a page template. Add the following XML to the **elements.xml** file:

```
<?xml version="1.0" encoding="utf-8" ?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Module Url="CustomPages">
    <File Url="PageTemplate.aspx"
      Name="TaskCreator.aspx"
      Type="Ghostable" />
  </Module>
</Elements>
```

This creates a new page instance **TaskCreator.aspx** that is based off the **PageTemplate.aspx** template within your Feature **Lab1** within the subdirectory **CustomPages** off the root of your site.

At this point the Feature is finished. Now you need to package it up as a WSS solution package for deployment.

Exercise 3: Creating a WSS solution package

In this exercise you will package your feature into a WSS solution package (*.wsp). This is done using the `MakeCab.exe` utility included in the [Microsoft Cabinet SDK](http://support.microsoft.com/default.aspx/kb/310618)¹ (KB 310618). This file has already been added to the virtual machine in the following location:

```
c:\windows\system32
```

1. `MakeCab.exe` is a command line tool that requires a few parameters. One is a Diamond Directive File (*.ddf) that contains a list of all the files to include in the package as well as any folders that should be included in the package. Since `MakeCab.exe` is a command line tool, that means it can be scripted, or the execution can be automated. This exercise will also walk you through the process of changing your Visual Studio project to always build a new WSS solution package using MSBuild, the .NET Framework's build utility (which is also used by Visual Studio for build / compile projects).

First, you need to add a few files to your project. Create a new folder called **DeploymentFiles** in the root of your project and add the following two files to this directory (you'll find these files in the **Resources** folder within first lab in the `c:\Student` folder):

- **BuildSharePointPackage.ddf** - This is the file that you will modify to include all the necessary files within the package; its used by `MakeCab.exe` to determine what folders and files should be added to the package.
 - **BuildSharePointPackage.targets** - This is a custom MSBuild targets file that contains the commands to execute `MakeCab.exe` with the necessary command line arguments.
2. Now you need to configure Visual Studio to call the custom MSBuild target (defined within the `BuildSharePointPackage.targets` file) every time the project is built. To do this, you need to unload the project by right-clicking it in the **Solution Explorer** tool window and selecting **Unload Project**. *If you don't see this option then Visual Studio is not likely configured to show solutions. To change this, from Visual Studio select **Tools » Options**, select the **Projects and Solutions » General** page. Check the option **Always show solution** and click **OK**.*
 3. With the project unloaded, right-click the project again in the **Solution Explorer** tool window and select **Edit Lab1.csproj**.
 4. Scroll to the bottom of the XML file. You should notice how it ends with an XML comment. Immediately before the comment you will also see an `<Import>` element. This XML element imports MSBuild targets files and is where you need to import your custom target file as well. Add the following XML just below the existing XML `<Import>` element:

```
<Import Project="DeploymentFiles\BuildSharePointPackage.targets" />
```

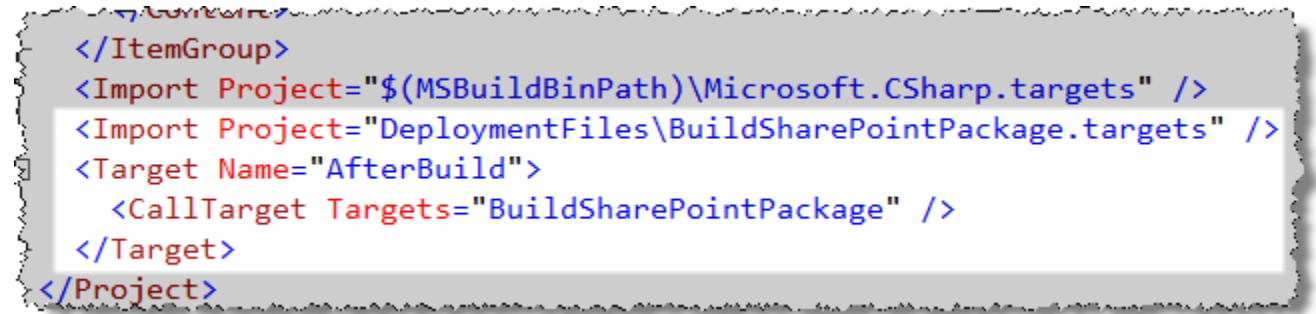
Once the custom MSBuild targets file has been imported, you need to tell MSBuild that after the project has been built, it needs to call your custom target (again,

¹ <http://support.microsoft.com/default.aspx/kb/310618>

defined in the *.targets file added to our project). Add the following XML, replacing the XML comment:

```
<Target Name="AfterBuild">
  <CallTarget Targets="BuildSharePointPackage" />
</Target>
```

The end of your project file should now look like the following image:



```
</ItemGroup>
<Import Project="$(MSBuildBinPath)\Microsoft.CSharp.targets" />
<Import Project="DeploymentFiles\BuildSharePointPackage.targets" />
<Target Name="AfterBuild">
  <CallTarget Targets="BuildSharePointPackage" />
</Target>
</Project>
```

Finally, save your changes and reload the project: Right-click the project in the **Solution Explorer** tool window and select **Reload project**. You may receive prompts to close the project file (select **OK**) and a security warning (select **Load project normally**, uncheck **Ask me for every project in this solution** and click **OK**). The security warning is Visual Studio telling you that it doesn't recognize this type of project file, which is expected because you just customized it!

5. SharePoint expects to find a `manifest.xml` file within the root of every WSS solution package. Add a new XML file named **manifest.xml** to the **DeploymentFiles** folder in the project and add the following XML code to the file:

```
<?xml version="1.0" encoding="utf-8" ?>
<Solution xmlns="http://schemas.microsoft.com/sharepoint/"
  SolutionId="942DF5EA-5BAE-4E2B-8854-8CD8401D6425"
  DeploymentServerType="WebFrontEnd"
  ResetWebServer="TRUE">

  <FeatureManifests>
    <FeatureManifest Location="Lab1\feature.xml" />
  </FeatureManifests>

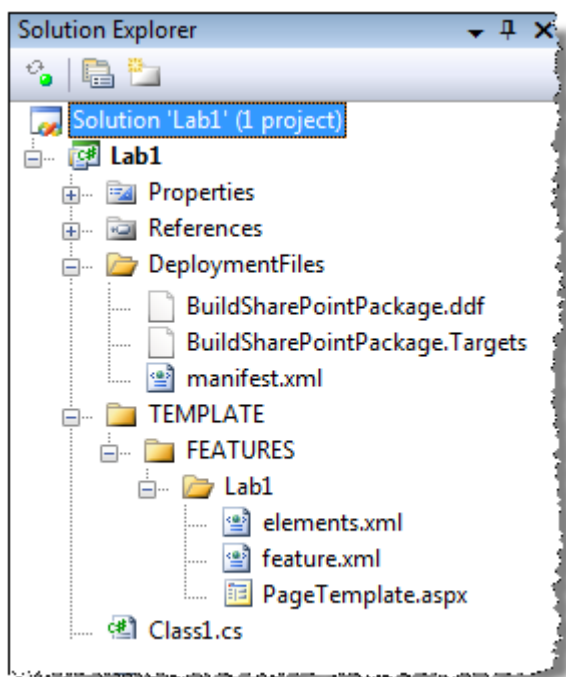
</Solution>
```

6. At this point all the files necessary for deployment have been created, filled with the necessary information (including our Feature) and your Visual Studio project has been configured to automatically build WSS solution packages. Now you need to edit the *.ddf file to tell `MakeCab.exe` to include the necessary files. Add the following code between the comments (the two lines with semicolons) in the **DeploymentFiles\BuildSharePointPackage.ddf** file:

```
DeploymentFiles\manifest.xml
bin\debug\Lab1.dll

.Set DestinationDir=Lab1
TEMPLATE\FEATURES\Lab1\feature.xml
TEMPLATE\FEATURES\Lab1\elements.xml
TEMPLATE\FEATURES\Lab1\PageTemplate.aspx
```

The Visual Studio project should now look like the following image:



7. Build the project and look at what was generated. Using Windows Explorer, navigate to the folder containing the Visual Studio solution. You will find two new files (**Lab1.cab** & **Lab1.wsp**) with a new folder: **wsp\Debug**. The MSBuild targets file called `MakeCab.exe` two times; both times it created a cabinet file but the first time it saved it with a *.cab extension and the second time with a *.wsp extension. Because Windows does not know what a *.wsp file is, creating a *.cab is easier so you can double-click and examine its contents. Take a moment to see what was included in the package.

At this point you could deploy the Feature by adding the WSS solution package to the SharePoint farm's solution store, deploy it and activate the Feature. However, it doesn't do anything worthwhile just yet. Before deployment, you will add some logic to the page template.

Exercise 4: Adding a code behind file to the provisioned page

In this exercise you will add some business logic in a code behind file to the page template within the Feature created in exercise 1.

1. Working with SharePoint pages is very similar to working with ASP.NET 2.0 pages with respect to code behind files. The difference is that you have to link the two files (the ASPX file and the code behind) together manually. The first step is to create the code behind file for the page template. Create a new C# class file within the **TEMPLATE\FEATURES\Lab1** folder named **PageTemplate.aspx.cs**. By default, Visual Studio's new class template creates private classes. Make sure you change it to a public class. Visual Studio also automatically created a namespace for you based on the folder structure. Change the namespace to be simply **Lab1**.

2. Because this Feature will contain code that will be deployed to the Global Assembly Cache (GAC), which is where assemblies containing Feature receivers must be deployed to, the assembly must be strongly named. To do this, right-click the project in the **Solution Explorer** tool window and select **Properties**. Select the **Signing** tab, check **Sign the assembly**, and select **<Browse...>** from the **Choose a strong name key file:**. Within the Resources directory for this lab you will find the **Litware.snk** key file. Select **Litware.snk** and close the Lab1 properties window.
3. In the overview of this lab, you read that you were going to build a page that created pages in a WSS 3.0 Team Site task list. In order to accept the input, you will need to access an ASP.NET 2.0 `TextBox`. Likewise, in order to write to a SharePoint list, you will need to work with SharePoint objects. This means you need to add two assembly references to your project. Add the following assemblies to the project: **System.Web** (`System.Web.dll`) and **Windows SharePoint Services** (`Microsoft.SharePoint.dll`).
4. The code behind will contain a single method, an event handler, that is triggered when a button is clicked. First, modify the class declaration so it inherits from **System.Web.UI.Page** since this will be a code behind file for your page template. Add the following method skeleton to the **PageTemplate.aspx.cs** file within the class:

```
protected void CreateTask_OnClick (object sender, EventArgs e)
{
}
```

5. Now you need to add code to the method to get the value from the ASP.NET `TextBox` on the page and create a new item in the current SharePoint site's **Tasks** list. If you get stuck, feel free to look at the lab solution located in the Solution folder for this lab.

Tip: Break this into two parts. First, get a reference to the `TextBox` (you will need to get a reference to the ASP.NET `ContentPlaceholder` first) using the `FindControl()` method. Next, create a new task item using the `Microsoft.SharePoint.SPContext` static object.

6. With the code behind created, now you need to wire it up. First, add an **Inherits** attribute to the **Page** directive with the following contents, pointing to the class in the assembly:

```
Lab1.PageTemplate, Lab1, Version=1.0.0.0, Culture=neutral, PublicKeyToken=d4e5777b16a5749f
```

Next, wire up the event handler to call when the button is clicked by adding the method **CreateTask_OnClick** to the **OnClick** attribute on the button. The button markup should look like this:

```
<asp:Button ID="btnCreateTask" runat="server" OnClick="CreateTask_OnClick"
Text="CreateTask" />
```

7. Make sure your code behind compiles by building the project. If not, go back and fix any errors.

- Next, the assembly needs to be added to the WSS solution package. The assembly is already being added to the *.WSP file, now it needs to be added to the manifest.xml file so SharePoint knows what to do with it. Open the **manifest.xml** file and add the following XML code within the XML **<Solution>** elements:

```
<Assemblies>
  <Assembly DeploymentTarget="GlobalAssemblyCache" Location="Lab1.dll">
    <SafeControls>
      <SafeControl Namespace="Lab1" TypeName="*" Safe="True" />
    </SafeControls>
  </Assembly>
</Assemblies>
```

At this point you have satisfied the requirements of the lab. However, go one step further and add code so that when the Feature is activated, a new navigation item is added that points to the new page instance.

Exercise 5: Adding Feature event handlers to add navigation elements

In this exercise you will add code that is called when the Feature is activated and deactivated to add/remove an item from the WSS 3.0 site's navigation to easily browse to the page created by the Feature.

- When you created the project, Visual Studio added a class file to the root: `Class1.cs`. Rename **Class1.cs** to **FeatureReceiver.cs**. Visual Studio will automatically rename the class within the code file to the same name as the file.
- Add the following two using statements to the top of the **FeatureReceiver.cs** file to save yourself some typing down the road:

```
using Microsoft.SharePoint;
using Microsoft.SharePoint.Navigation;
```

- Because you need to add code that will execute when the Feature is activated / deactivated, you need to create a Feature Receiver. To do this, the class needs to inherit from **Microsoft.SharePoint.SPFeatureReceiver** and implement four methods. Modify the contents of the **FeatureReceiver.cs** file so that it contains the following code:

```
public class FeatureReceiver : SPFeatureReceiver {
    // nothing to do on install/uninstall
    public override void FeatureInstalled (SPFeatureReceiverProperties properties) {
    }
    public override void FeatureUninstalling (SPFeatureReceiverProperties properties) {
    }
    public override void FeatureActivated (SPFeatureReceiverProperties properties) {
    }
    public override void FeatureDeactivating (SPFeatureReceiverProperties properties) {
    }
}
```

- First, you need to add the code that will create a new link upon activation. Add the following code to the **FeatureActivated()** event handler:

```
// get reference to the current site's top navigation
```

```

SPWeb site = properties.Feature.Parent as SPWeb;
if (site == null)
    return;
SPNavigationNodeCollection quickLaunch = site.Navigation.QuickLaunch;

// create new nav element for new page
SPNavigationNode taskCreatorPage = new SPNavigationNode("Task Creator",
"CustomPages/TaskCreator.aspx", false);

quickLaunch.AddAsLast(taskCreatorPage);

```

5. Next, when the Feature is deactivated the link should be removed. Add the following code to the **FeatureDeactivating()** event handler:

```

// get reference to the current site's top navigation
SPWeb site = properties.Feature.Parent as SPWeb;
if (site == null)
    return;
SPNavigationNodeCollection quickLaunch = site.Navigation.QuickLaunch;

for (int i = quickLaunch.Count-1; i >= 0; i--) {
    if (quickLaunch[i].Title == "Task Creator") {
        // delete the node
        quickLaunch[i].Delete();
        return;
    }
}

```

6. With the Feature receiver created, now the Feature must be configured to call each method at the appropriate time. To do this, you will specify the assembly and class containing the Feature receiver in the Feature definition file. Open the **feature.xml** file and add the following two attributes to the XML **<Feature>** element:

```

ReceiverAssembly="Lab1, Version=1.0.0.0, Culture=neutral, PublicKeyToken=d4e5777b16a5749f"
ReceiverClass="Lab1.FeatureReceiver"

```

That is it! Save all changes and build the project. Assuming you have a clean build, it is now time to deploy the solution and test the results.

Exercise 6: Deploy WSS solution package and test

In this final exercise, you will deploy the WSS solution package and verify your work.

1. First the WSS solution package must be deployed. Open a command prompt and navigate to the following directory:

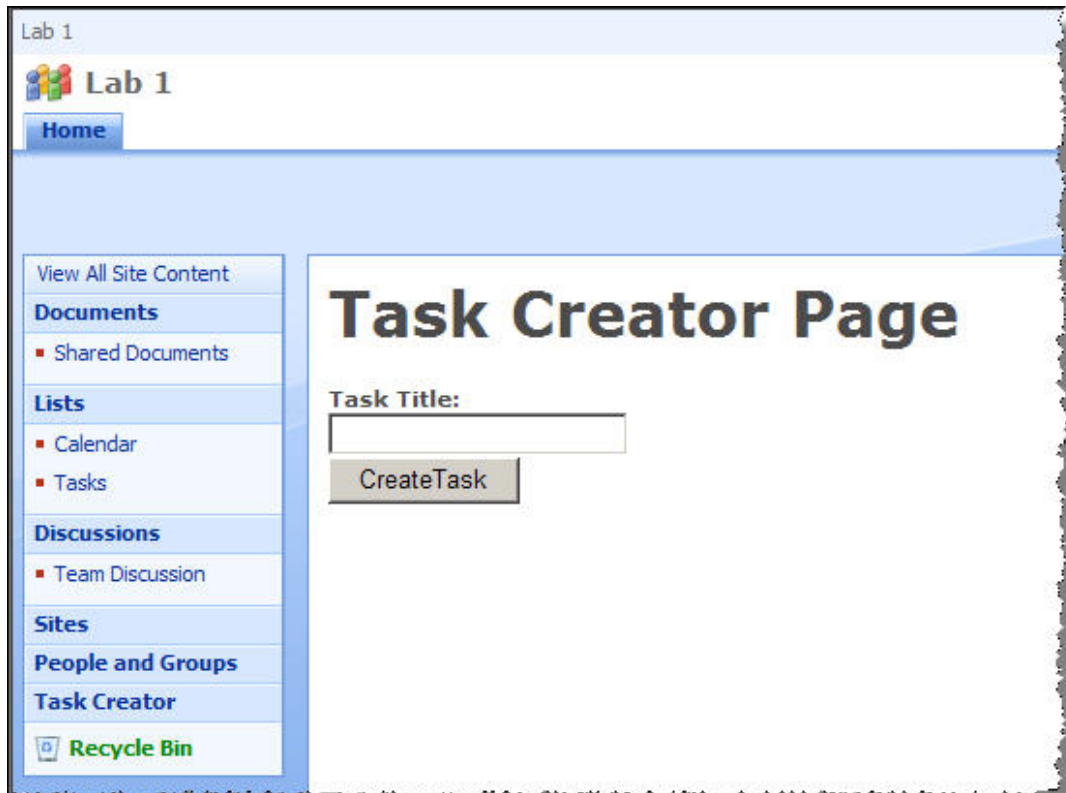
```
c:\Program Files\Common Files\Microsoft Shared\web server extensions\12\BIN
```

2. Enter the following command into the command line window and hit **Enter**:

```
stsadm -o addsolution -filename c:\Student\Labs\Lab01_WssPrimer\Lab\wsp\Debug\Lab1.wsp
```

3. Launch Central Administration by selecting **Start » All Programs » Microsoft Office Server » SharePoint 3.0 Central Administration**.
4. From the **Central Administration** site, select the **Operations** tab and then select **Solution management** under the **Global Configuration** section.
5. On the **Solution Management** page, click the link on **lab1.wsp**.

6. On the **Solution Properties** page, select **Deploy Solution**.
7. On the **Deploy Solution** page, specify **Now** in the **Deploy When?** section, **http://litwareinc.com** in the **Deploy To?** section and click **OK**.
8. The deployment of the WSS solution package deployed the Lab1 Feature to the [...] \12\TEMPLATE\FEATURES folder and added the assembly to the GAC. Now test the Feature by browsing to the **http://litwareinc.com/sites/lab1** site and select **Site Actions » Site Settings**.
9. On the **Site Settings** page, select **Site features** under the **Site Administration** section.
10. On the **Site Features** page, click **Activate** on the **Lab 1 - Working with Provisioned Pages Feature**. After activating the Feature, click the **Home** link in the top navigation bar.
11. Notice how the **Quick Launch** has a new navigation item at the end called **Task Creator**. Click that link, it will take you to the page that was created by the Feature, as shown in the following image:



12. Enter a value in the textbox and click **Create Task**. Now navigate to the **Tasks** list in the site and you will see the item that was just added to the list from your page!

Lab 02: MOSS 2007 & Web Content Management Overview

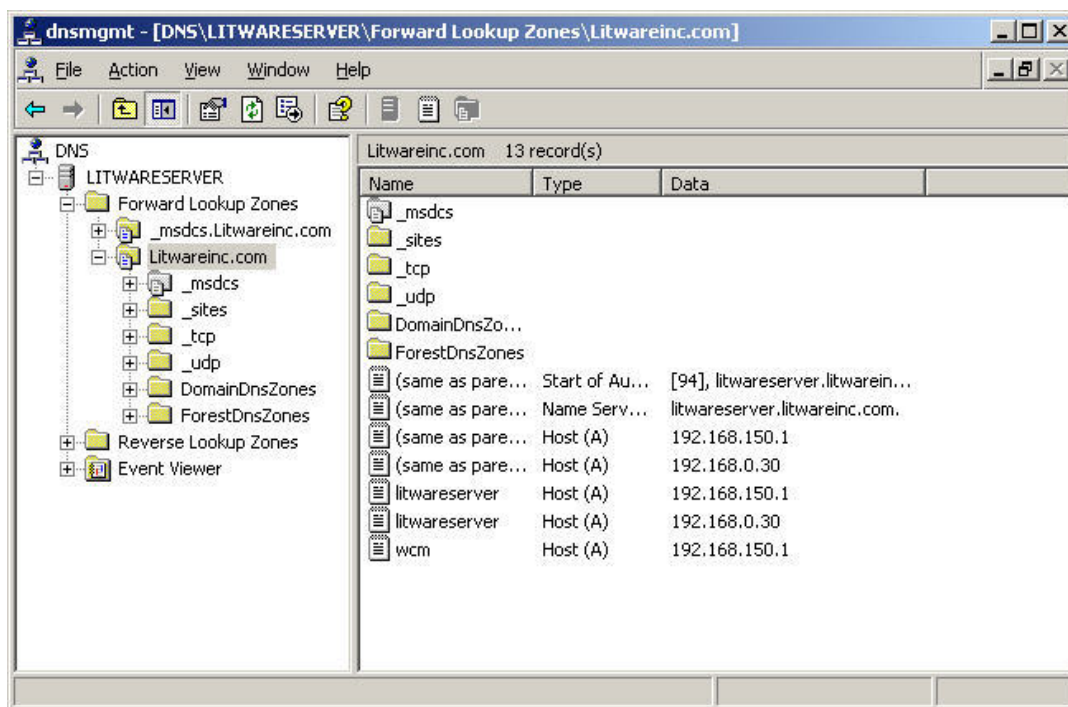
Lab Time: 45 minutes

Lab Overview: This lab will help you get familiar with the Microsoft Office SharePoint Server (MOSS) 2007 Web Content Management (WCM) features and capabilities. In addition, you will also spend some time working within the `Microsoft.SharePoint.Publishing` namespace, where everything WCM can be found within the SharePoint object model. In this lab you will first create a new Web application and site collection based off the Publishing Portal site template and add a few content pages using the Web authoring tools. Then you will dive into the `Microsoft.SharePoint.Publishing` namespace to interrogate the Publishing site and some aspects within it.

Exercise 1: Creating a new Web application & provisioning a Publishing site collection

In this exercise you will create a new Web application that you will use throughout the remainder of the course.

1. If you haven't already, make sure you are logged in as **LITWAREINC\Administrator** (with a password of **pass@word1**).
2. Before you create a Web application, add a new DNS entry on the server for the new URL you will use. Select **Start » Administrative Tools » DNS**.
3. In the **dnsmgmt** utility, expand the tree on the left-hand side of the utility to **DNS » LITWARESERVER » Forward Lookup Zones » Litwareinc.com**. Right-click **Litwareinc.com** in the tree in the left-hand panel of the utility and select **New Host (A)...**. Enter the following information in the **New Host** dialog and click **Add Host** (when finished, your DNS settings should look like the following image):
 - **Name:** wcm
 - **IP Address:** 192.168.150.1



4. Launch Central Administration by selecting **Start » All Programs » Microsoft Office Server » SharePoint 3.0 Central Administration**.
5. From the **Central Administration** site, select the **Application Management** tab and then select **Create or extend Web application** under the **SharePoint Web Application Management** section.
6. On the **Create or Extend Web Application** page, select **Create a new Web application**.
7. On the **Create New Web Application** page, use the following information to create a new Web application and click **OK** (if not listed, leave default setting):
 - **IIS Web Site:** Create a new IIS web site
 - **Description:** (leave alone... this will automatically be changed as you sent the next two values)
 - **Port:** 80
 - **Host Header:** wcm.litwareinc.com
 - **Application Pool:** Use existing application pool & **SharePointDefaultAppPool (Litwareinc\SP_WorkerProcess)**
 - **Database Server:** LitwareServer
 - **Database Name:** WSS_WCM
 - **Database Authentication:** Windows authentication (recommended)
8. Upon the completion of creating the Web application, you are redirected to the **Application Created** page. Select the **Create Site Collection** link.

9. On the **Create Site Collection** page, complete the required information as follows and click **OK** to create a new site collection:

- **Web Application:** <http://wcm.litwareinc.com>
- **Title:** Litware Inc.
- **Description:** Litware Inc. Publishing Site
- **Web Site Address:** <http://wcm.litwareinc.com/>
- **Template Selection:** Publishing Portal (found under the Publishing tab)
- **Primary Site Collection Administrator:** LITWAREINC\administrator
- Make sure to click the icon or press **[CTRL]+[K]** to validate the account... this may take a second or two to resolve. It should resolve to **Litware Admin Guy**.

10. When the site collection process is complete, you will be redirected to a page with a link to the new site collection. Click the link to open the site in a new window.

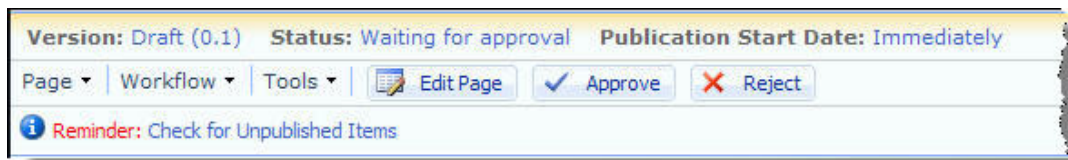
At this point you now have a new Web application containing a single site collection that's been created using the Publishing Portal site template. You will use this site throughout the remainder of the course.

Exercise 2: Create content pages

With a site created, this exercise will familiarize you with the process of creating some content pages.

11. If you haven't already, navigate to the Litware Inc. Publishing site at **<http://wcm.litwareinc.com>**.
12. Select **Press Releases** in the top horizontal navigation. This is where you will create your content pages.
13. To create a new content page, select **Site Actions » Create Page**.
14. On the **Create Page** page, specify the following information and click **Create**:
- **Title:** Press Release 1
 - **Description:** Description of press release 1.
 - **URL Name:** should be automatically filled, but if not, enter PressRelease1
 - **Page Layout:** (Article Page) Article page with image on left
15. Use the field controls on the page to specify the **Page Image**, **Image Caption**, **Article Date**, **Byline**, and **Page Content**. Once you have entered the desired content, select **Submit for Approval** at the top of the page to start the page approval workflow.
16. On the **Start "Parallel Approval": PressRelease1** page, click **Start**.
17. The page will then load with the **Press Release 1** page you just created, but not in edit mode. Notice how the Page Editing Toolbar (PET) Quick Access buttons have changed (refer to the following image). This is because the buttons are aware of the

different states of the page. To advance the page through the workflow, select the **Approve** button in the PET.



18. On the **Workflow Tasks: Please approve PressRelease1** page, optionally enter any comments and click the **Approve** button. The browser will refresh with the published page in display mode.
19. Repeat this process a few more times to create a total of three content pages. Each time, pick a different name, URL and enter different content. When selecting the page layout on the **Create Page** page, make sure you pick only page layouts that are associated with the **Article Page** content type, as indicated in the following image:



In the page layouts module we will explore this dialog further. For now, know that associated content type is the part in parentheses and the name of the page layout is to the right.

At this point, you have added some sample content to your Publishing site. Next you will access this site and content using the SharePoint object model.

Exercise 3: Working with the Microsoft.SharePoint.Publishing namespace

In this exercise you will create a console application that displays information about the Publishing site collection created in the last exercise.

1. Open **Visual Studio** and create a new **C# Console Application** project named **Lab2**. You should create the project within the following path so the project files will reside within the same directory structure as all the other lab exercises:

```
c:\Student\Labs\02_WcmArchitecture\Lab
```

2. In this lab you will be working with common SharePoint objects as well as objects specific to the `Microsoft.SharePoint.Publishing` namespace. In order to do this, you will need to add two assembly references to the project: **Windows SharePoint**

Services (Microsoft.SharePoint.dll) which you will find under the **.NET** tab in the **Add Reference** dialog and **Microsoft.SharePoint.Publishing.dll** (no component name specified). To add the **Microsoft.SharePoint.Publishing.dll** reference you will need to pick the file from the following location on the file system using the **Browse** tab in the **Add Reference** dialog: **c:\Program Files\Common Files\Microsoft Shared\web server extensions\12\ISAPI.**

3. Make your life a little easier by adding the following using statements to the top of the **Program.cs** file:

```
using Microsoft.SharePoint;
using Microsoft.SharePoint.Publishing;
```

4. The first thing you will do is output some information about the Publishing site collection you created. Add the following code to the static **Main** method in **Program.cs**:

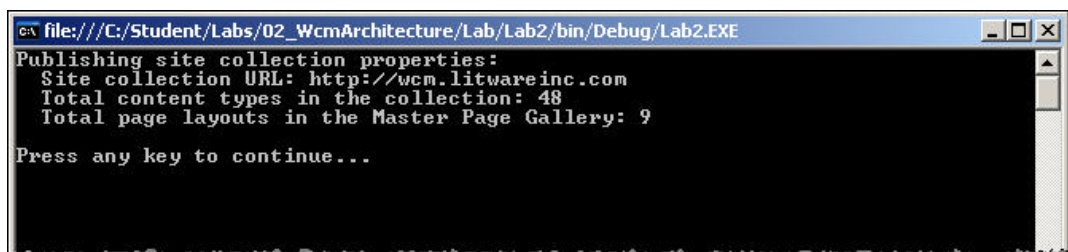
```
// get reference to the litware publishing site
SPSite siteCollection = new SPSite("http://wcm.litwareinc.com");

// get reference to the publishing site
PublishingSite publishingSiteCollection = new PublishingSite(siteCollection);

// write out some information about the publishing site
Console.Out.WriteLine("Publishing site collection properties:");
Console.Out.WriteLine(" Site collection URL: {0}", publishingSiteCollection.Site.Url);
Console.Out.WriteLine(" Total content types in the collection: {0}",
publishingSiteCollection.ContentTypes.Count.ToString());
Console.Out.WriteLine(" Total page layouts in the Master Page Gallery: {0}",
publishingSiteCollection.PageLayouts.Count.ToString());
Console.Out.WriteLine("");

Console.Out.WriteLine("Press any key to continue...");
Console.ReadLine();
```

Press **F5** to build and run the code... you should see something similar to the following image:



5. Now get a reference to the root web, first making sure it's a **PublishingWeb**, and output some information about the Pages list. Add the following code to **Program.cs** after **Console.Out.WriteLine("")** in the code added in the previous step:

```
// get reference to the publishing web..
SPWeb site = siteCollection.RootWeb;
if (PublishingWeb.IsPublishingWeb(site)) {
    PublishingWeb publishingSite = PublishingWeb.GetPublishingWeb(site);

    // write out some information about the publishing web
    Console.Out.WriteLine("Publishing web properties:");
```



```
Console.Out.WriteLine(" Pages list name: {0}", publishingSite.PagesListName);
Console.Out.WriteLine(" Total pages in the list: {0}",
publishingSite.PagesList.ItemCount.ToString());
Console.Out.WriteLine("");
}
```

Press **F5** to build and run the code... you should see something similar to the following image:

6. Finally, add some code just to the end of the **IF** statement added in the previous step to display all the pages in the Pages library:

```
// write out all pages
Console.Out.WriteLine("Publishing pages within the root web's Pages library:");
foreach (PublishingPage page in publishingSite.GetPublishingPages()) {
    Console.Out.WriteLine(" Page: {0}", page.Name);
    Console.Out.WriteLine(" Page URL: {0}", page.Url);
    Console.Out.WriteLine(" Underlying SPLISTItem ID: {0}", page.ListItem.ID.ToString());
}
```

Press **F5** to build and run the code... you should see something similar to the following image:

7. Now... see if you can modify the code to point to the Press Releases subsite where you created the pages in the previous exercise to output the same information displayed in steps 5 & 6 in this exercise. If you get stuck, refer to the solution of this lab.

Tip: You only have to modify a single line of code!

Lab 03: Authentication, Authorization & Security

Lab Time: 60 minutes

Lab Overview: This lab walks you through creating a new Publishing site but creating two authentication paths for accessing the site. This is a common scenario for content-centric Internet facing sites. Companies want the ability to have certain employees access the site using their corporate AD credentials to create, edit, review, approve and publish content. In addition, there is a need to have the same site be available to the world as an anonymous site. Some companies also have password protected areas of their site. Your IT staff will not be happy when they hear your team has decided to put every user who creates an account on the public website in the corporate Active Directory... that just is not safe, secure... or sane! At the end of this lab, you will have two paths into your site, one non-anonymous using AD for authentication and another, anonymous access with some password protected areas using Forms Based Authentication (FBA) for authentication.

Exercise 1: Creating a new Web application & provisioning a Publishing site collection

In this exercise you will create a new Web application that will be the starting point for the rest of this lab. In the previous lab, you created a new DNS entry and Web application and site collection using the Publishing Portal site template. Refer to those same instructions for the following steps.

1. Create a new DNS **A** record where **Name** = **extranet** and **IP** = **192.168.150.1**.
2. Create a new Web application with the following values:
 - **IIS Web Site:** Create a new IIS web site
 - **Description:** (leave alone... this will automatically be changed as you sent the next two values)
 - **Port:** 80
 - **Host Header:** extranet.litwareinc.com
 - **Application Pool:** Use existing application pool & SharePointDefaultAppPool (Litwareinc\SP_WorkerProcess)
 - **Database Server:** LitwareServer
 - **Database Name:** WSS_Extranet
 - **Database Authentication:** Windows authentication (recommended)
3. Create a new site collection within the Web application created in step 2 above using the following values:
 - **Web Application:** http://extranet.litwareinc.com
 - **Title:** Litware Inc.
 - **Description:** Litware Inc. Publishing Site Web Site

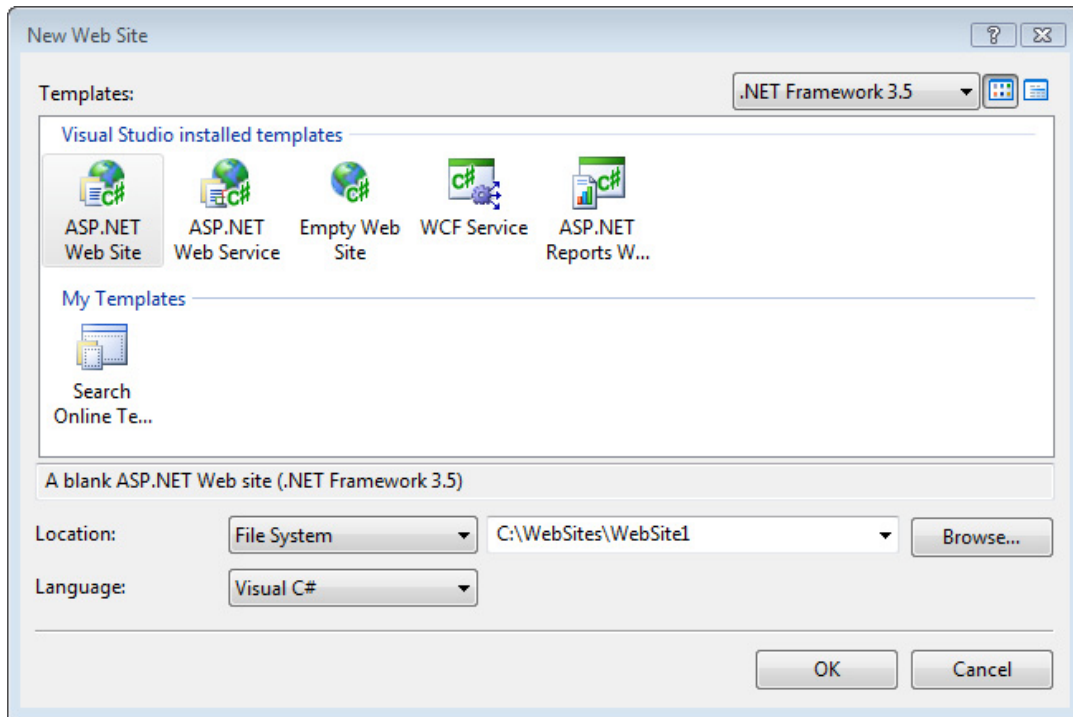
- **Address:** http://extranet.litwareinc.com
- **Template Selection:** Publishing Portal (found under the Publishing tab)
- **Primary Site Collection Administrator:** LITWAREINC\administrator

At this point you now have a new Web application and site collection. This site will be used as the "authoring site" where your company employees will go to edit the site content.

Exercise 2: Creating & configuring the FBA ASP.NET 2.0 database

Before you can setup an alternate access point for your authoring site for the world to see (with much more restrictive permissions). In this exercise you will create a new ASP.NET 2.0 database that will retain the users and roles used on the externally facing site (which you will create in the next exercise).

1. First, you need to create the ASP.NET 2.0 database that will contain the users and roles before you configure any providers or create users/roles. Run the utility provided by Microsoft to launch the ASP.NET SQL Server Setup Wizard... the utility can be found here:
c:\Windows\Microsoft.NET\Framework\v2.0.5027\aspnet_regsql.exe.
When prompted, use the following values:
 - **Server:** LitwareServer
 - **Windows Authentication**
 - **Database:** LitwareFBA
2. With the database created, you need to grant a user access to the database. This will be the user account your SharePoint site will access the database as. To get this user account, check the identity of the application pool containing Web application created in exercise 1 previously. To save time, the application pool you used is the **SharePointDefaultAppPool** which is configured to execute using the **LITWAREINC\SP_WorkerProcess** account. Using **SQL Server Management Studio** (available from **Start » All Programs » Microsoft SQL Server 2005 » SQL Server Management Studio**), add the **LITWAREINC\SP_WorkerProcess** account to the **LitwareFBA** database and grant it the following roles:
db_datareader & db_datawriter.
3. The next step involves creating the necessary membership & role providers, as well as creating a few users and roles in the database. Microsoft has included a simple web application to assist in managing the database, however it is only accessible when launched via Visual Studio. It's not so bad as it also gives you a good place to test your providers without adding the complexity of SharePoint.
4. Open Visual Studio and create a new **Visual C#, ASP.NET Web Site** on the **File System** in the following location:
c:\Student\Labs\03_AuthenticationAuthorization\Lab\LitwareFbaWebSite.
Refer to the following image for more information:



5. The first step is to establish a connection to the FBA database. Replace the existing **<connectionStrings />** XML element with the following:

```
<connectionStrings>
  <add name="LitwareFba"
        connectionString="server=LitwareServer; database=LitwareFBA; Integrated
Security=SSPI;"
        providerName="System.Data.SqlClient" />
</connectionStrings>
```

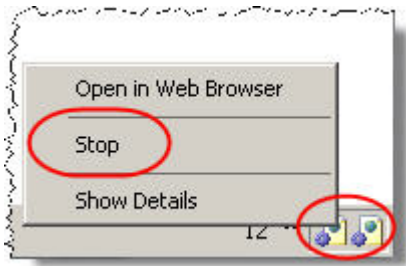
6. Next, add the following XML within the **<system.web>** elements to define the membership and role providers:

```
<!-- membership provider -->
<membership defaultProvider="LitwareFbaSqlMembershipProvider">
  <providers>
    <add name="LitwareFbaSqlMembershipProvider"
          type="System.Web.Security.SqlMembershipProvider, System.Web,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
          connectionStringName="LitwareFba"
          enablePasswordRetrieval="false"
          enablePasswordReset="true"
          requiresQuestionAndAnswer="false"
          applicationName="/"
          requiresUniqueEmail="false"
          passwordFormat="Hashed"
          maxInvalidPasswordAttempts="5"
          minRequiredPasswordLength="1"
          minRequiredNonalphanumericCharacters="0"
          passwordAttemptWindow="10"
          passwordStrengthRegularExpression="" />
  </providers>
</membership>

<!-- role provider -->
```

```
<roleManager enabled="true" defaultProvider="LitwareFbaSqlRoleProvider">
  <providers>
    <add name="LitwareFbaSqlRoleProvider"
      type="System.Web.Security.SqlRoleProvider, System.Web, Version=2.0.0.0,
      Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
      connectionStringName="LitwareFba"
      applicationName="/" />
  </providers>
</roleManager>
```

7. With the connection string and providers configured, you can now use Visual Studio to launch the Web application you can use to manage the database. Within Visual Studio, select **Website » ASP.NET Configuration**. The first thing you need to do is switch the site from **Integrated Authentication** to **Forms Authentication** by selecting the **Security** link and then **Select Authentication Type** in the **Users** container. Make sure the option **From The Internet** is selected... this is another way to say "Forms Based Authentication." Finally click **Done**.
8. With the security all configured, test the providers. Select the **Provider** tab, then click **Select a different provider for each feature (advanced)** and click the **Test** link next to the two providers defined the **web.config**: **LitwareFbaSqlMembershipProvider** & **LitwareFbaSqlRoleProvider**. If either test reports an error, go back and double check the **web.config**. Make sure you close the browser and stop the local running instances of ASP.NET (found in the system tray... refer to the following image) every time you rerun the test:



9. Now add two new users to the database. If it is not already active, select the **Security** tab and then **Create user** within the **Users** container.
10. On the **Create User** page, enter the following information for the new user, make sure the **Active User** checkbox is **checked** and click **Create User**:
 - **User Name:** brian.perry
 - **Password & Confirm Password:** pass@word1
 - **E-mail:** brian.perry@litwareinc.com
11. Repeat the step above for a second user which will be the administrator account on the anonymous access/FBA site:
 - **User Name:** FbaAdministrator
 - **Password & Confirm Password:** pass@word1
 - **E-mail:** fba.admin@litwareinc.com

At this point you have now confirmed you have a valid connection string, membership and role provider definitions. You also have created a database that contains a single user account which you will use later to test the site.

Exercise 3: Extending a new Web application with different authentication & enabling anonymous access

In this exercise you will create a second Web application that will provide a second link into the site collection created in exercise 1. It is this second Web application that you will configure to use the alternate authentication mechanism.

1. Create a new DNS A record where **Name = internet** and **IP = 192.168.150.1** (refer to lab #2, exercise #1 if you need a refresher on how to do this).
2. Launch Central Administration by selecting **Start » All Programs » Microsoft Office Server » SharePoint 3.0 Central Administration**.
3. From the **Central Administration** site, select the **Application Management** tab and then select **Create or extend Web application** under the **SharePoint Web Application Management** section.
4. On the **Create or Extend Web Application** page, select **Extend an existing Web application**.
5. On the **Extend Web Application to Another IIS Web Site** page, use the following information to extend an existing Web application by creating a new Web application:
 - **Web Application:** http://extranet.litwareinc.com
 - **Description:** (leave alone... this will automatically be changed as you sent the next two values)
 - **Port:** 80
 - **Host Header:** internet.litwareinc.com
 - **Load Balanced URL / Zone:** Internet
6. Now it is time to configure the Web applications so they can both access the FBA ASP.NET 2.0 database. To do this you will add the connection string, membership provider and role provider definitions created in the Visual Studio Web Site project previously in this lab into the `web.config` files for each Web application. Open the **web.config** for the **extranet.litwareinc.com** site (found in this folder: **c:\Inetpub\wwwroot\wss\VirtualDirectories\extranet.litwareinc.com80**) and add the following XML below just after the closing **</SharePoint>** element and opening **<system.web>** element:

Tip: Save yourself some time typing and copy the XML from the `web.config` in the Visual Studio Web Site project created previously in exercise 2. This applies to all XML in this step.

```
<connectionStrings>
```

```
<add name="LitwareFba"
      connectionString="server=LitwareServer; database=LitwareFBA; Integrated
Security=SSPI;"
      providerName="System.Data.SqlClient" />
</connectionStrings>
```

Next, add the following XML just after the opening XML `<system.web>` element:

```
<!-- membership provider -->
<membership defaultProvider="LitwareFbaSqlMembershipProvider">
  <providers>
    <add name="LitwareFbaSqlMembershipProvider"
          type="System.Web.Security.SqlMembershipProvider, System.Web,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
          connectionStringName="LitwareFba"
          enablePasswordRetrieval="false"
          enablePasswordReset="true"
          requiresQuestionAndAnswer="false"
          applicationName="/"
          requiresUniqueEmail="false"
          passwordFormat="Hashed"
          maxInvalidPasswordAttempts="5"
          minRequiredPasswordLength="1"
          minRequiredNonalphanumericCharacters="0"
          passwordAttemptWindow="10"
          passwordStrengthRegularExpression="" />
  </providers>
</membership>



<!-- role provider -->
<roleManager enabled="true" defaultProvider="LitwareFbaSqlRoleProvider">
  <providers>
    <add name="LitwareFbaSqlRoleProvider"
          type="System.Web.Security.SqlRoleProvider, System.Web, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
          connectionStringName="LitwareFba"
          applicationName="/" />
  </providers>
</roleManager>
```

Save your changes. Now do the same thing to the **web.config** for the **internet.litwareinc.com** site (found in this folder: **c:\Inetpub\wwwroot\wss\VirtualDirectories\internet.litwareinc.com80**).

7. With the two sites' Web applications configured, you now need to configure the Central Administration site because you may need to manage the security on the Web applications from within Central Administration, therefore it needs the same changes to communicate with the FBA ASP.NET 2.0 database. Repeat the previous step for the **web.config** for the **Central Administration** site (found in this folder: **c:\Inetpub\wwwroot\wss\VirtualDirectories\####**).

However there is one difference. Change the **defaultProvider** attribute on the XML **<roleManager>** element to **AspNetWindowsTokenRoleProvider**. This is necessary because Central Administration still uses Windows Authentication for the role provider.

With all the **web.config** modifications complete, now you can turn on FBA on the <http://internet.litwareinc.com> site.

8. From the **Central Administration** site, select the **Application Management** tab and then select **Authentication providers** under the **Application Security** section.
9. On the **Authentication Providers** page, if it is not selected already, change the **Web Application** to **http://extranet.litwareinc.com**. Then select the **Internet** zone link.
10. On the **Edit Authentication** page, use the following information to complete the form and click **Save**:
 - **Authentication Type:** Forms
 - Check **Enable Anonymous Access**
 - **Membership Provider Name:** LitwareFbaSqlMembershipProvider
 - **Role Manager Name:** LitwareFbaSqlRoleProvider
11. Now you need to add users to the site. Browse to the **http://extranet.litwareinc.com** site and select **Site Actions » Site Settings » People And Groups** and then select **New** button.
12. On the **Add Users: Litware Inc.** page, use the following information to complete the form and then click **OK**:
 - **Add Users:** brian.perry (press [CTRL]+[K] or select the icon  to validate the name)
 - Give Permission / Add users to a SharePoint group: Litware Inc. Visitors [Read]
13. See if everything is working correctly. Browse to **http://extranet.litwareinc.com...** you should be able to open the site with no problem. Next, browse to **http://internet.litwareinc.com...** you should be prompted to sign in. Use the credentials for the **brian.perry** account.
14. The final touch is to configure the **http://internet.litwareinc.com** site to allow for anonymous users to access the site... not prompting them to login to the site. In a previous step, the Web application was configured for anonymous access... but that does not mean SharePoint is allowing it. To configure the site for anonymous access, you need to login as a user who has administrative level rights. No user has those rights on the **http://internet.litwareinc.com** site yet though... so you're caught in a catch-22. No fear... SharePoint now includes a way to grant rights to an entire web application without having rights to that Web application (note, this is ~only~ a farm administration level setting). To do this, open **Central Administration**, select the **Application Management** tab and then select **Policy for Web application** in the **Application Security** section. Verify you have the correct Web application selected (in this case, **http://extranet.litwareinc.com**).
15. On the **Policy for Web Application** page, select **Add Users**.
16. On the **Add Users** page, make sure the **http://extranet.litwareinc.com** Web application is selected, select the **Internet** zone and click **Next**. Then, enter the user **FbaAdministrator** and press [CTRL]+[K] or select the icon  to validate the

name. Then check **Full Control** for the permission and click **Finish**. The FbaAdministrator account now has full unrestricted access to the <http://internet.litwareinc.com> site, the Internet zone for <http://extranet.litwareinc.com>.

17. Now browse to **<http://internet.litwareinc.com>** site, login as the **FbaAdministrator** and select **Site Actions » Site Settings**, then select **Modify All Site Settings**.
18. On the **Site Settings** page, select **Advanced Permissions**.
19. On the **Advanced Permissions** page, select **Settings » Anonymous Access**.
20. On the **Change Anonymous Access Settings: Litware Inc.** page, select **Entire Web Site** and click **OK**.
21. Now open a new browser, browse to **<http://internet.litwareinc.com>** and you should not have to login to the site.

At this point you have now completed setting up two authentication paths into your site. One path allows your corporate employees to access the site using their corporate AD credentials and the other allows for anonymous access and FBA authentication for certain protected areas.

Exercise 4: Configuring a password protected area of the anonymous access site

In this exercise, you will configure a subsite within the <http://internet.litwareinc.com> site to require authentication before browsing to that site. If the user does not have permission or is not logged in, they will not see the subsite.

1. Configure the **Press Releases** site so that it is not available to anonymous users. Browse to **<http://internet.litwareinc.com/PressReleases>** and login using the **FbaAdministrator** account. Then select **Site Actions » Site Settings » Modify All Site Settings** and then select **Advanced Permissions**.
2. On the **Permissions: Press Releases** page, select **Actions » Edit Permissions** to break permission inheritance in order to enable anonymous access.
3. On the **Permissions: Press Releases** page, select **Settings » Anonymous Access**.

Note: If you do not see an option for anonymous access, you may need to break permission inheritance from the Press Releases site from its parent site.

4. On the **Change Anonymous Access Settings** page, select **Nothing** and click **OK**.
5. Open a new browser and navigate to the **<http://internet.litwareinc.com>**. Notice how the Press Releases section no longer appears? Now login as one of the users you added to the site and notice how it comes back.

Lab 04: Creating Master Pages & Customizing Navigation

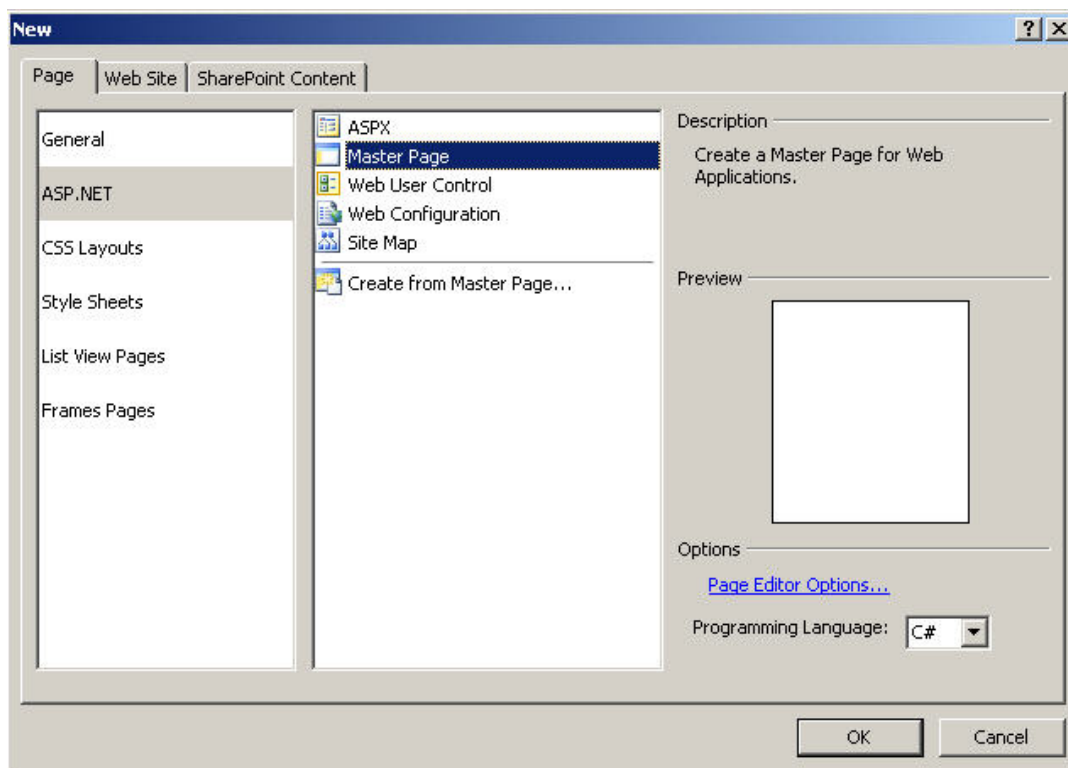
Lab Time: 45 minutes

Lab Overview: In this lab you will walk through creating and adding a master page to the Master Page Gallery two ways: first using SharePoint Designer and second, using Visual Studio and Features.

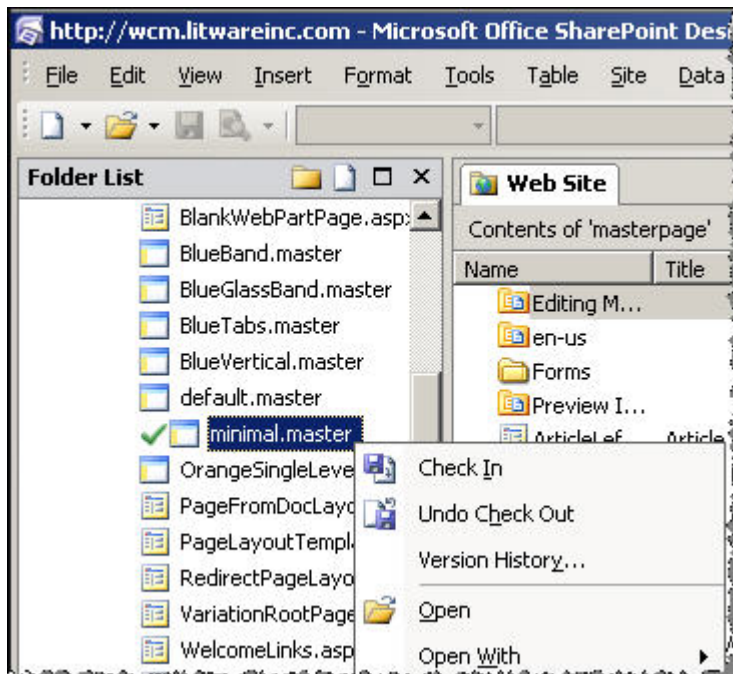
Exercise 1: Creating a minimal master page with SharePoint Designer

In this exercise you will create a minimal master page using Office SharePoint Designer 2007. Creating master pages this way will start the page off as a customized page rather than one based off a template.

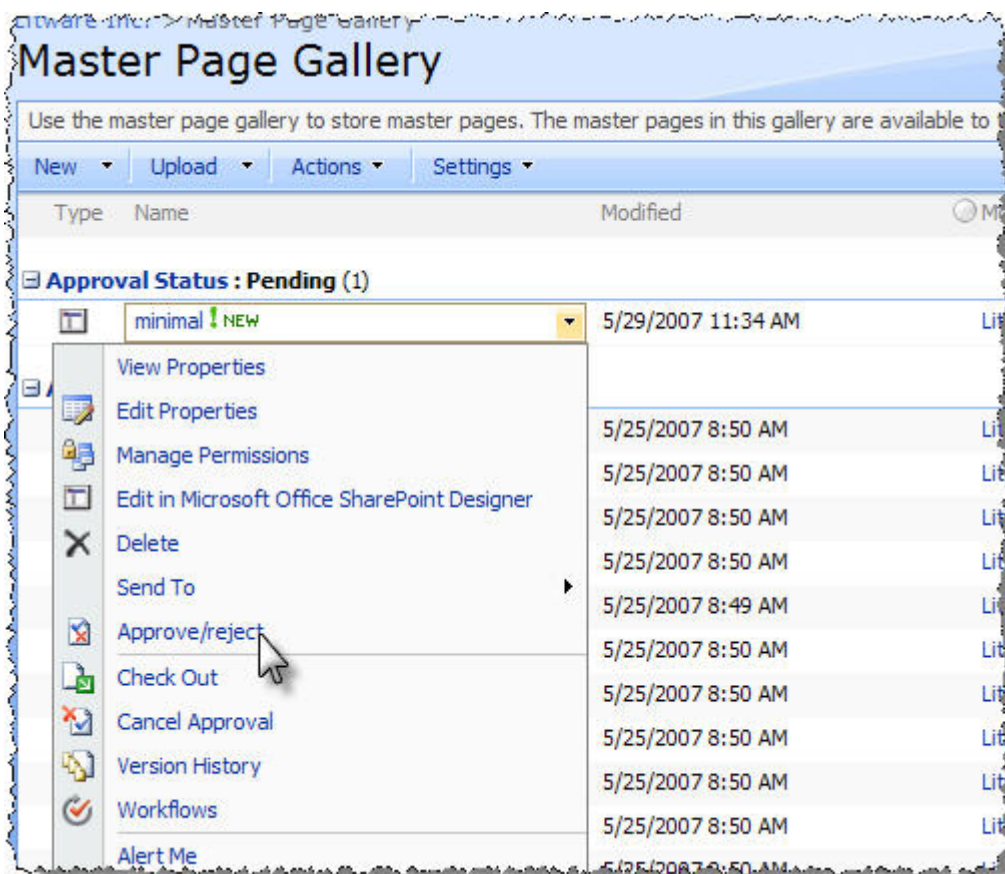
1. Open **SharePoint Designer** by selecting **Start » All Programs » Microsoft Office » SharePoint Designer 2007**.
2. If SharePoint Designer automatically loads a site when started, close the site by selecting **File » Close Site**.
3. Open the Publishing site created in a prior lab by selecting **File » Open Site....** In the **Open Site** dialog, enter **http://wcm.litwareinc.com** in the **Site name** field and click **Open**.
4. To create a new master page, select **File » New....** In the **New** dialog, select the **Page** tab, then **ASP.NET** in the first column, **Master Page** in the middle column and finally click **OK** (refer to the image below):



5. Next, open the sample minimal master page **minimal.master** file located in the **Resources** folder within this lab
(**c:\Student\Labs\04_MasterPagesNavigation\Resources\minimal.master**) in something other than SharePoint Designer (notepad or Visual Studio will work just fine) because it is not made to open SharePoint master pages from the file system.
6. Copy the entire contents of the **minimal.master** file and paste them into the new master page created in step 4, replacing the default contents added by SharePoint Designer.
7. Save the changes to the master page by selecting **File » Save As....** You want to save the master page into the site's **Master Page Gallery**. Make sure the **Current Site** button is selected in the **Save As** dialog on the left panel and browse to the following path within the site:
http://wcm.litwareinc.com/_catalogs/masterpage. Name the new master page **minimal.master** using the **Field name:** field and click **Save**.
8. At this point the master page is still checked out and unpublished. While you are the one that created it so you can immediately use it within the site, anyone else hitting the site will receive an error because the file will not be checked in and/or published. In your local development environment this may not be an issue, but for good practice, go through the process of checking in and publishing the master page. Right-click **minimal.master** in the **Folder List** tool window and select **Check In** as shown in the following image:



Then select **Publish a major version** and click **OK**. A dialog will appear asking if you want to view/modify the approval status of the master page. Click **Yes** which will open a new browser window loading the Master Page Gallery with your master page at the top of the list. From the ECB menu of the minimal master page, select **Approve/reject**, as shown in the following image:



9. On the **Master Page Gallery: minimal page**, select **Approved**. This item will become visible to all users.
10. At this point the master page has been approved and can now be seen by anyone browsing the site. Configure the site to use the new minimal master page by selecting **Site Actions » Site Settings » Modify All Site Settings**.
11. On the **Site Settings** page, select **Master page** from the **Look and Feel** section, select **minimal.master** for the **Site Master Page** and click **OK**. Click the **Litware Inc.** logo at the top-left corner of the page to browse back to the homepage of the site to see the new master page in use.

At this point you have now created a master page using SharePoint Designer. The downside to this approach is that your master page has started off as a customized page that lives exclusively in the SharePoint content database. In the next exercise you will create a master page that starts as a page template using Visual Studio and WSS Features.

Exercise 2: Creating a minimal master page with Visual Studio and Features

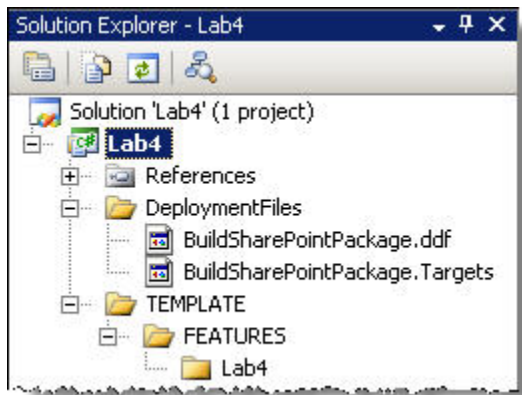
In this exercise you will create a master page template and deploy it with a SharePoint Feature. Similar to the Visual Studio project created in lab 1 exercise 2, in this exercise you will create a Visual Studio project that does not compile any code but simply packages all the files into a WSS solution package.

1. Open Visual Studio and create a new **C# Empty Project** project named **Lab4** (the **Empty Project** template is available when you highlight the **Visual C#\Windows** project type in the **New Project** dialog). You should create the project within the following path so the project files will reside within the same directory structure as all the other lab exercises:

c:\Student\Labs\04_MasterPagesNavigation\Lab

Notice in this case you are using the Empty Project template rather than the Class Library template for this project. The reason why is that the Empty Project template is a little more slimmed down.

2. Now you need to create the folder structure that will contain the Feature. The Feature you will create is going to live within the Features folder nested within SharePoint's "12" system folder. To make your life easier, create a mirror of the "12" system folder structure from the TEMPLATE folder on down (**TEMPLATE\FEATURES\Lab4**). In addition, create a folder at the root of the project named **DeploymentFiles** and copy **BuildSharePointPackage.ddf** and **BuildSharePointPackage.targets** from the lab **Resources** folder into this directory. The following image shows what your project should look like after creating the necessary folder structure:



3. The next step is to create the necessary files for your feature. The first file to add is the master page. Copy the files **Sample.master** & **SampleMasterPreview.gif** from the lab **Resources** folder into the **Lab4** Feature folder within the "12" system folder in the Visual Studio project.
4. Now create the feature definition file by adding a new **XML** file to the project and naming it **feature.xml**. This file should reside in the same **Lab4** folder as the master page and image. Add the following XML markup to the feature.xml file:

```
<?xml version="1.0" encoding="utf-8" ?>
<Feature xmlns="http://schemas.microsoft.com/sharepoint/"
  Id="30D0CA6D-899A-4d31-B8D8-1FE4844E2380"
  Title="Lab 4 - Working with Provisioned Master Pages"
  Hidden="FALSE"
  Scope="Site"
  Version="1.0.0.0">

  <ElementManifests>
    <ElementManifest Location="elements.xml" />
  </ElementManifests>
</Feature>
```

```

        <ElementFile Location="Sample.master" />
        <ElementFile Location="SampleMasterPreview.gif" />
    </ElementManifests>

</Feature>

```

- Next, you need to add the elements file that will provision the master page and preview image into the Master Page Gallery and associate the preview image with the master page. Add a new XML file named **elements.xml** and add it to the same **Lab4** folder as the other files. Add the following XML markup to the elements.xml file (the HTML/XML comments are not necessary to enter... they are there to explain each piece of the XML):

```

<?xml version="1.0" encoding="utf-8" ?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <!-- add the master page to the Master Page Gallery -->
  <Module Url="_catalogs/masterpage" RootWebOnly="TRUE">
    <!-- provision the master page into the Master Page Gallery -->
    <File Url="Sample.master"
      Name="SampleFromFeature.master"
      Type="GhostableInLibrary">

      <!-- specify the master page's list item content type -->
      <Property Name="ContentType"
        Value="$Resources:cmscore,contenttype_masterpage_name;" />
      <!-- specify the URL to the preview image, provisioned below -->
      <Property Name="PublishingPreviewImage"
        Value="~SiteCollection/_catalogs/masterpage/Preview
Images/Litware/SampleMasterPreviewFeature.gif, ~SiteCollection/_catalogs/masterpage/Preview
Images/Litware/SampleMasterPreviewFeature.gif" />
      <!-- specify the title of the master page -->
      <Property Name="Title"
        Value="SampleFromFeature.master" />
    </File>
  </Module>

  <!-- Add the preview image to the Master Page Gallery -->
  <Module Url="_catalogs/masterpage/Preview Images/Litware"
    RootWebOnly="TRUE">
    <!-- provision the preview image into the Master page gallery -->
    <File Url="SampleMasterPreview.gif"
      Name="SampleMasterPreviewFeature.gif"
      Type="GhostableInLibrary">

      <!-- specify the title of the preview image -->
      <Property Name="Title"
        Value="SampleMasterPreviewFeature.gif" />
    </File>
  </Module>
</Elements>

```

- Now you need to create the WSS solution package. The following few steps are similar to the steps outlined in the first lab, but not exact. Because you used the Empty Project template, there will be a slight difference as your build process in this lab won't actually compile anything. To start, you need to edit the project file to call the custom targets file.

7. You first need to unload the project by right-clicking it in the **Solution Explorer** tool window and selecting **Unload Project**. With the project unloaded, right-click the project again in the **Solution Explorer** tool window and select **Edit Lab4.csproj**.
8. Here is where things are a bit different from the first lab. In the opening XML **<Project>** element, change the attribute **DefaultTargets** value from **Build** to **BuildSharePointPackage**.
9. Now, scroll down to the only XML **<Import>** element. Change the **Project** attribute value to be the following:

```
<Import Project="DeploymentFiles\BuildSharePointPackage.targets" />
```

Save all your changes and close the **Lab4.csproj** file.

10. Right-click the project in the **Solution Explorer** tool window and select **Reload project**. You may receive prompts to close the project file (select **OK**) and a security warning (select **Load project normally**, uncheck **Ask me for every project in this solution** and click **OK**).
11. SharePoint expects to find a manifest.xml file within the root of every WSS solution package. Add a new XML file named **manifest.xml** to the **DeploymentFiles** folder in the project and add the following XML code to the file:

```
<?xml version="1.0" encoding="utf-8" ?>
<Solution xmlns="http://schemas.microsoft.com/sharepoint/"
  SolutionId="29254E7F-954B-4312-9108-66C3919754F0"
  DeploymentServerType="WebFrontEnd"
  ResetWebServer="FALSE">

  <FeatureManifests>
    <FeatureManifest Location="Lab4\feature.xml" />
  </FeatureManifests>

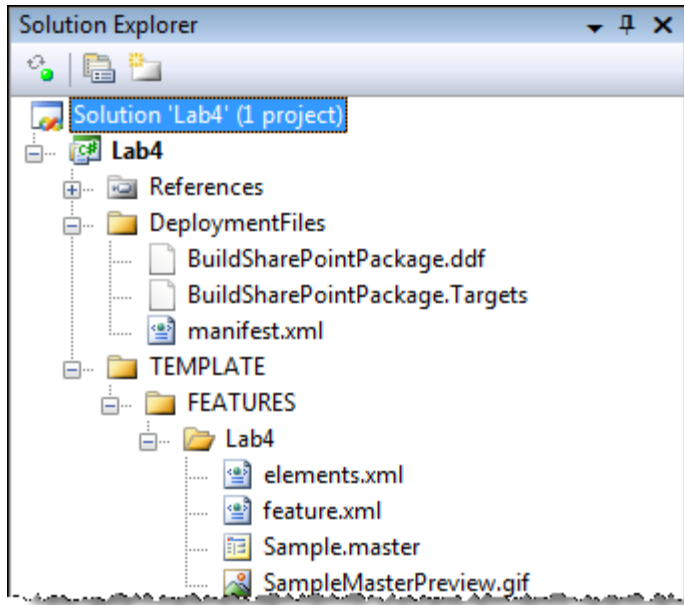
</Solution>
```

12. At this point all the files necessary for deployment have been created, filled with the necessary information (including our Feature) and your Visual Studio project has been configured to automatically build WSS solution packages. Now you need to edit the *.ddf file to tell makecab.exe to include the necessary files. Add the following code between the comments (the two lines with semicolons) in the **DeploymentFiles\BuildSharePointPackage.ddf** file:

```
DeploymentFiles\manifest.xml

;Set DestinationDir=Lab4
TEMPLATE\FEATURES\Lab4\feature.xml
TEMPLATE\FEATURES\Lab4\elements.xml
TEMPLATE\FEATURES\Lab4\Sample.master
TEMPLATE\FEATURES\Lab4\SampleMasterPreview.gif
```

The Visual Studio project should now look like the following image:



13. Now it's time to build deploy the WSS solution package. Build the project in **Visual Studio** to create the WSP file.

14. First the WSS solution package must be deployed. Open a command prompt and navigate to the following directory:

```
c:\Program Files\Common Files\Microsoft Shared\web server extensions\12\BIN
```

15. Enter the following command into the command line window and hit Enter:

```
stsadm -o addsolution -filename  
c:\Student\Labs\04_MasterPagesNavigation\Lab\wsp\Debug\Lab4.wsp
```

16. Launch **Central Administration** by selecting **Start » All Programs » Microsoft Office Server » SharePoint 3.0 Central Administration**.

17. From the **Central Administration** site, select the **Operations** tab and then select **Solution management** under the **Global Configuration** section.

18. On the **Solution Management** page, click the link on **lab4.wsp**.

19. On the **Solution Properties** page, select **Deploy Solution**.

20. On the **Deploy Solution** page, specify **Now** in the **Deploy When?** section and click **OK**.

21. Test the Feature by browsing to the <http://wcm.litwareinc.com/> site and select **Site Actions » Site Settings » Modify All Site Settings**.

22. On the **Site Settings** page, select **Site collection features** under the **Site Collection Administration** section.

23. On the **Site Collection Features** page, click **Activate** on the **Lab 4 - Working with Provisioned Master Pages Feature**. After activating the Feature, click the **Home** link in the top navigation bar.

24. Now change the master page for the Publishing site. Select **Site Actions » Site Settings » Modify All Site Settings**, then select **Master page** from the **Look and Feel** section on the **Site Settings** page. Select the **SampleFromFeature.master** master page in the **Site Master Page** section (yes, the preview image isn't the right size, but you see how to create a preview image now). Finally, browse to the homepage of **<http://wcm.litwareinc.com>** to see the new master page. You should see a slightly different master page with an orange bar in the upper left (*this is a copy of the OrangeSingleLevel.master master page*).
25. Finally, take a look at the files provisioned. Select **Site Actions » Manage Content and Structure**. On the **Site Content and Structure** page, select the **Master Page Gallery**. Once the page refreshes, notice the master page you added at the bottom of the list. Additionally, select the subfolder **Preview Images** in the **Master Page Gallery**. Within the **Litware** folder you will find the preview image.

You have now created a master page using the template method. The file within the Master Page Gallery is still just referencing the template on the file system (within the Feature folder **Lab4**) until it is customized using SharePoint Designer.

Lab 05: Creating Custom Page Layouts

Lab Time: 60 minutes

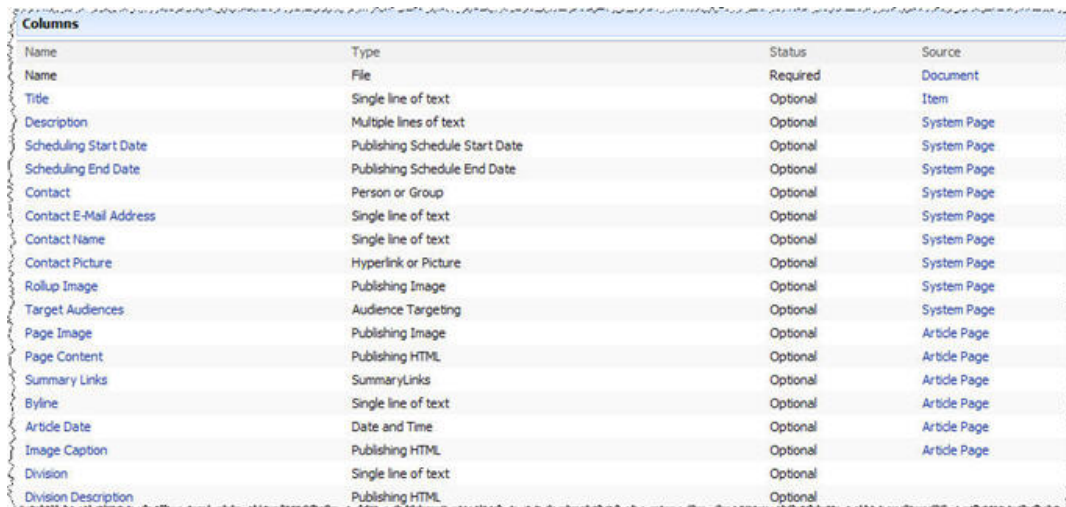
Lab Overview: In this lab you will practice creating a new content type and site columns based off the Publishing Page content type. Then you will create a page layout using SharePoint Designer based off this content type. After using the browser and SharePoint Designer to create a new content type and page layout, you will then do the same thing except only using Visual Studio and Features.

Exercise 1: Creating new site columns & a content type based off the Page content type

In this exercise you will create a new content type using the SharePoint browser interface. The content type you create will contain additional site columns from the OOTB Page content type.

1. First you need to create two new site columns. Open a browser and navigate to **<http://wcm.litwareinc.com>**. Then select **Site Actions » Site Settings » Modify All Site Settings**.
2. On the **Site Settings** page, select **Site columns** from the **Galleries** section. On the **Site Column Gallery** page, select **Create**.
3. On the **New Site Column: Litware Inc.** page, use the following information to create a new site column and click OK:
 - **Name and Type - Column name:** Division
 - **Name and Type - The type of information in this column is:** Single line of text
 - **Group: New group:** WCM401
4. Repeat the previous step using the following information to create another site column and click OK:
 - **Column name:** Division Description
 - **The type of information in this column is:** Full HTML content with formatting and constraints for publishing
 - **Group: Existing group:** WCM401
5. Using the breadcrumb navigation, select **Site Settings**. On the **Site Settings** page, select **Site content types** from the **Galleries** section. On the **Site Content Type Gallery** page, select **Create**.
6. On the **New Site Content Type** page, use the following information to create a new content type and click OK:
 - **Name:** Division Article
 - **Select parent content type from:** Page Layout Content Types

- Parent Content Type: Article Page
 - Group: New Group: WCM401
7. On the Site Content Type: Division Article page, select Add from existing site columns.
 8. On the Add Columns to Site Content Type: Division Article page, select the two fields created in steps 3 & 4 (found in group WCM401) and click OK.
 9. You should now see the two columns listed along with all the other site columns inherited by this content type as shown in the following image:



Name	Type	Status	Source
Name	File	Required	Document
Title	Single line of text	Optional	Item
Description	Multiple lines of text	Optional	System Page
Scheduling Start Date	Publishing Schedule Start Date	Optional	System Page
Scheduling End Date	Publishing Schedule End Date	Optional	System Page
Contact	Person or Group	Optional	System Page
Contact E-Mail Address	Single line of text	Optional	System Page
Contact Name	Single line of text	Optional	System Page
Contact Picture	Hyperlink or Picture	Optional	System Page
Rollup Image	Publishing Image	Optional	System Page
Target Audiences	Audience Targeting	Optional	System Page
Page Image	Publishing Image	Optional	Article Page
Page Content	Publishing HTML	Optional	Article Page
Summary Links	SummaryLinks	Optional	Article Page
Byline	Single line of text	Optional	Article Page
Article Date	Date and Time	Optional	Article Page
Image Caption	Publishing HTML	Optional	Article Page
Division	Single line of text	Optional	
Division Description	Publishing HTML	Optional	

At this point you now have created a few new site columns and a new content type that includes these site columns and inherits from the Page content type.

Exercise 2: Creating a new page layout using SharePoint Designer

In this exercise you will create a new page layout associated with the content type created in the previous exercise using SharePoint Designer. Then you will create a new content page based off the page layout and content type.

1. Open **SharePoint Designer** and open the site **http://wcm.litwareinc.com**.
2. Now you need to create a new page layout based off the content type created in the previous step. From within SharePoint Designer, select **File » New....**
3. In the **New** dialog, select the **SharePoint Content** tab, then **SharePoint Publishing** in the left-hand panel and **Page Layout** in the center panel. Use the following information to complete the remainder of the dialog and click OK:
 - **Content Type Group:** WCM401
 - **Content Type Name:** Division Article
 - **URL Name:** DivisionArticleImageLeft.aspx
 - **Title:** Division article with image on left

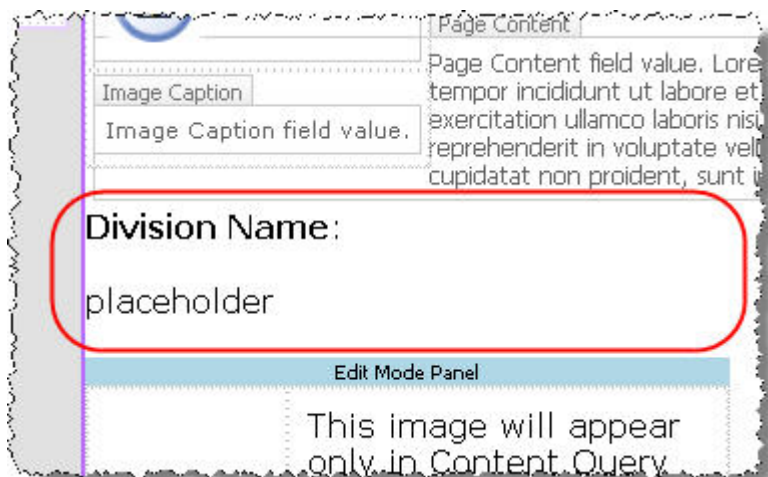
4. Now you need to add some structure to this page. Cheat a bit by copying the contents of the **ArticleLeft.aspx** page layout and paste them into **DivisionArticleImageLeft.aspx**, replacing everything that was there (do this only within the **Code** view, ~not~ within Design or Split mode).
5. Now you need to add the two fields added to the content type. These will be added just below the main content of the page. Switch to Code mode if you have not already and add the following HTML just after the closing `<div>` tag for the page content (on or about line #50), as shown by the image below (the circled portion is the new code you should add):


```

48 <div class="pageContent">
49 <PublishingWebControls:RichHtmlField id="Content" runat="server" />
50 </div>
51 <div>
52 <strong>Division Name</strong>:
53 <p>placeholder</p>
54 </div>
55 <PublishingWebControls:editmodepanel runat="server" />
56 <!-- Add field controls here to bind custom fields to the page
57 <table cellpadding="10" cellspacing="0" align="center">

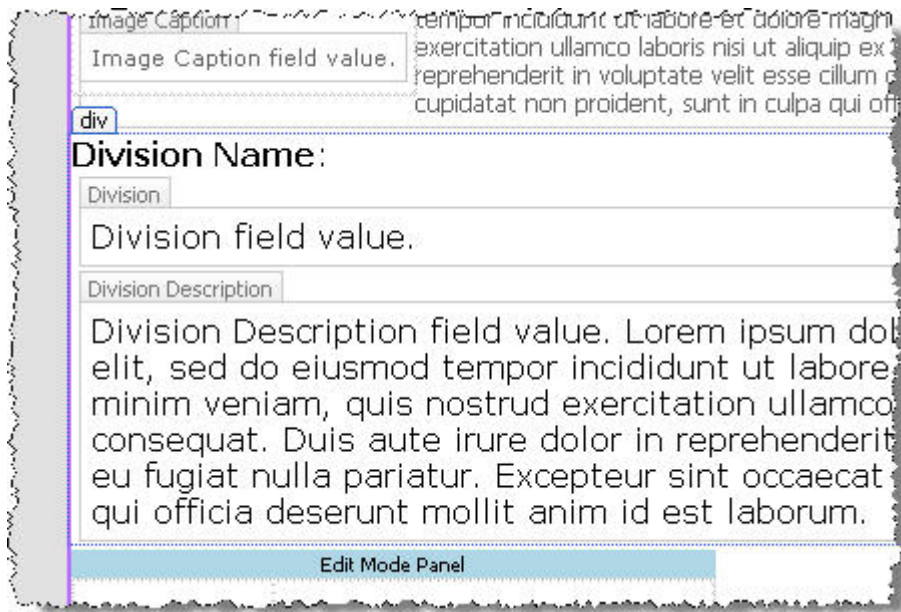
```

6. Save your changes. Now, switch into **Design** view... you should see an image similar to the following one (the highlighted part is where you added the HTML):



7. The next thing to do is add the columns you added to the content type. In the **Toolbox** tool window in SharePoint Designer (right-hand side of the application), click the  icon (found in the upper right corner of the tool window) to give yourself more room to work. Scroll the tool window down to the **SharePoint Controls...** specifically the **Content Fields** within the **SharePoint Controls** group. Notice the two fields you created and added to the content type previously. Drag the two fields into the design surface, placing the **Division** field next to **Division:** in the ASPX file and **Division Description** immediately after **placeholder**. Now remove the placeholder text... you should see something similar to the following image (if yours

are arranged slightly differently, it is not a problem... as long as there are no rendering problems you are in good shape):



8. Save all your changes. Now you need to check in and publish the file (same reasons why you had to / didn't have to with the master page in the last lab). Right-click **DivisionArticleImageLeft.aspx** in the **Folder List** tool window and select **Check In**. Then select **Publish a major version** and click **OK**. A dialog will appear asking if you want to view/modify the approval status of the page layout. Click **Yes** which will open a new browser window loading the Master Page Gallery with your page layout near the top of the list. From the ECB menu of the **DivisionArticleImageLeft.aspx** page layout, select Approve/reject. On the **Master Page Gallery: DivisionArticleImageLeft** page, select **Approved**. This item will become visible to all users.
9. At this point the page layout has been published to the Master Page Gallery and you can now create content pages based off the page layout and content type. Browse to the Press Releases section within the <http://wcm.litwareinc.com> site.
10. From the **Press Releases** section, select **Site Actions » Create Page**. On the **Create Page** page, use the following information to create the new content page and click Create:
 - **Title:** Division Article 1
 - **URL Name:** DivisionArticle1
 - **Page Layout:** (Division Article) Division article with image on left
11. You should now see your page in edit mode. Once you have entered the desired content, select **Submit for Approval** at the top of the page to start the page approval workflow.
12. On the **Start "Parallel Approval": DivisionArticle1** page, click **Start**.

13. The page will then load with the **Division Article 1** page you just created, but not in edit mode. To advance the page through the workflow, select the **Approve** button in the Page Editing Toolbar.
14. On the **Workflow Tasks: Please approve DivisionArticle1** page, optionally enter any comments and click the **Approve** button. The browser will refresh with the published page in display mode.

At this point you have created site columns and a content type using the browser and a page layout using SharePoint Designer. Next you'll see how to do the same thing without the browser or SharePoint Designer.

Exercise 3: Creating new site columns & a content type using Features

In this exercise you will create a Feature that, when activated, will create a few new site columns and content type based from the OOTB content type Page (and including the site columns created).

Since you have now created and customized Visual Studio projects that compile then package the contents of the project into a WSS solution package, as well as creating a project that doesn't compile anything but simply packages the contents into a WSS solution package, we have given you a project to get started with. This project already has the directory structure created, the deployment files added and the changes to the project file to run the automated WSS solution package creation.

1. In Visual Studio, open the **Lab5** solution located in the following directory:

```
c:\Student\Labs\05_PageLayouts\Lab\Lab5.sln
```

2. The first thing you need to do is create the Feature definition file. Create a new XML file named **feature.xml** in the **Lab5** Feature folder in the project, filling it with the following markup:

```
<?xml version="1.0" encoding="utf-8" ?>
<Feature xmlns="http://schemas.microsoft.com/sharepoint/"
  Id="E052E45D-B487-4593-AF2B-D6283E21C43A"
  Title="Lab 5 - Working with Provisioned Site Columns, Content Types and Page Layouts"
  Hidden="FALSE"
  Scope="Site"
  Version="1.0.0.0">

  <ElementManifests>
    <ElementManifest Location="SiteColumns.xml" />
    <ElementManifest Location="ContentType.xml" />
  </ElementManifests>

</Feature>
```

3. Next, create the two element manifest files listed in the Feature definition file. Create two XML files in the **Lab5** Feature folder in the project named **SiteColumns.xml** & **ContentType.xml**.
4. Open the **SiteColumns.xml** file and add the following XML markup (you can omit the XML comments, they are here just as a comment to what the XML is doing):

```
<?xml version="1.0" encoding="utf-8" ?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <!-- new site column off the "site line of text" field type -->
  <Field SourceID="http://schemas.microsoft.com/sharepoint/"
    ID="{DC172FD7-CC9D-493c-A1EB-231B8AF29F3E}"
    Name="Division2"
    DisplayName="Division"
    Group="WCM401Feature"
    Type="Text"
    Required="FALSE"
    Sealed="FALSE"
    Hidden="FALSE" />

  <!-- new site column based off the "publishing html" field type -->
  <Field SourceID="http://schemas.microsoft.com/sharepoint/"
    ID="{8DDDC83E-625C-4d5e-B4EF-0CD8A3A434C2}"
    Name="DivisionDescription"
    DisplayName="Division Description"
    Group="WCM401Feature"
    Type="HTML"
    Required="FALSE"
    Sealed="FALSE"
    Hidden="FALSE" />
</Elements>
```

5. Open the **ContentType.xml** file and add the following XML markup (you might want to read the note after the code first, makes this less painful):

```
<?xml version="1.0" encoding="utf-8" ?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <!-- this content type inherits from the "article page" content type -->
  <ContentType
    ID="0x010100C568DB52D9D0A14D9B2FDCC96666E9F2007948130EC3DB064584E219954237AF3900242457EFB8B
    24247815D688C526CD44D00976A5C165AB04065B1F9AFAE0E4658B5"
    Name="Division Article Feature"
    Group="WCM401Feature">
    <FieldRefs>
      <FieldRef ID="{DC172FD7-CC9D-493c-A1EB-231B8AF29F3E}" Name="Division2" />
      <FieldRef ID="{8DDDC83E-625C-4d5e-B4EF-0CD8A3A434C2}"
        Name="DivisionDescription" />
    </FieldRefs>
    <DocumentTemplate TargetName="/_layouts/CreatePage.aspx" />
  </ContentType>
</Elements>
```

Notice the very long content type ID. This content type is based off the Article Page content type. Don't try to type this in... copy it from the following file:

```
c:\Program Files\Common Files\Microsoft Shared\web server
extensions\12\TEMPLATE\FEATURES\PublishingResources\PublishingContentTypes.xml
```

On or about **line #80**, you will find the **Article Page** content type (or search for the string **\$Resources:cmscore,contenttype_articlepage_name;**, the name of the file). Copy the value from the ID field into the content type ID you are creating in this step. Then you need to add some uniqueness to it... do this by appending **00**, then a new GUID (without the brackets or hyphens). So, the new ID would be something like this:

```
[ContentType ID from Article Page] + 00 + [GUID with no brackets or dashes]
```


Another thing to notice is the GUIDs for the two XML **<FieldRef>** elements. These must match exactly to the ID's of the two site columns created in the previous step.

- Next, add the manifest file for the Feature and the necessary entries in the DDF file to package up the required files. Create a new XML file named **manifest.xml** to the **DeploymentFiles** folder in the project and add the following XML markup:

```
<?xml version="1.0" encoding="utf-8" ?>
<Solution xmlns="http://schemas.microsoft.com/sharepoint/"
  SolutionId="BF7F3B37-953D-4732-8E20-9CE5F65B1F2E"
  DeploymentServerType="WebFrontEnd"
  ResetWebServer="FALSE">

  <FeatureManifests>
    <FeatureManifest Location="Lab5\feature.xml"/>
  </FeatureManifests>

</Solution>
```

- Finally, add the following lines to the **BuildSharePointPackage.ddf** file between the comments:

```
DeploymentFiles\manifest.xml

.Set DestinationDir=Lab5
TEMPLATE\FEATURES\Lab5\Feature.xml
TEMPLATE\FEATURES\Lab5\SiteColumns.xml
TEMPLATE\FEATURES\Lab5\ContentType.xml
```

At this point you have created a Feature that will create new site columns and a new content type that (1) inherits from the Page content type and (2) contains the site columns created by the feature.

Exercise 4: Creating a new page layout using Features

In this exercise you will update the Feature created in the previous exercise to include creating a page layout with an associated preview image. Once this is done, it will then be time to deploy the solution, activate the feature, and test everything!

- The first thing you need to do is add the resource files to the Feature. Copy the **DivisionArticleImageLeft.aspx** and **SamplePageLayoutPreview.gif** files into the **Lab5** Feature folder within the project.
- Next, create a new XML file in the **Lab5** Feature folder named **PageLayout.xml** and add the following XML markup to the file:

```
<?xml version="1.0" encoding="utf-8" ?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <!-- add the page layout to the Master Page Gallery -->
  <Module Url="_catalogs/masterpage"
    RootWebOnly="TRUE">
    <!-- provision the page layout into the Master Page Gallery -->
    <File Url="DivisionArticleImageLeft.aspx"
      Name="DivisionArticleImageLeftFeature.aspx"
      Type="GhostableInLibrary">
      <!-- specify the content type associated this page layout is associated with --
    >
    <Property Name="PublishingAssociatedContentType"
```

```

        Value="";#Division Article
Feature;#0x010100C568DB52D9D0A14D9B2FDCC96666E9F2007948130EC3DB064584E219954237AF3900242457
EFB8B24247815D688C526CD44D00976A5C165AB04065B1F9AFAE0E4658B5;#" />
        <!-- specify the URL to the preview image, provisioned below -->
        <Property Name="PublishingPreviewImage"
            Value="~SiteCollection/_catalogs/masterpage/Preview
Images/Litware/SamplePageLayoutPreviewFeature.gif,
~SiteCollection/_catalogs/masterpage/Preview
Images/Litware/SamplePageLayoutPreviewFeature.gif" />
        <!-- specify the page layout content type -->
        <Property Name="ContentType"
            Value="$Resources:cmscore,contenttype_pagelayout_name;" />
        <!-- specify the title of the page layout -->
        <Property Name="Title"
            Value="Litware Division Article" />
    </File>
</Module>

<!-- add the preview image to the Master Page Gallery -->
<Module Url="_catalogs/masterpage/Preview Images/Litware"
    RootWebOnly="TRUE">
    <!-- provision the preview image into the Master Page Gallery -->
    <File Url="SamplePageLayoutPreview.gif"
        Name="SamplePageLayoutPreviewFeature.gif">
        <!-- specify the title of the preview image -->
        <Property Name="Title"
            Value="SamplePageLayoutPreviewFeature.gif" />
    </File>
</Module>
</Elements>

```

Notice the page layout properties **PublishingAssociatedContentType** value looks a bit odd. That is a delimited string of two values separated by ";#". the first value is the name of the content type and the second is the content type ID.

- Now you need to add the files to the Feature definition file and DDF file for packaging. Open the **feature.xml** file and add the following XML markup to the **<ElementManifests>** node, after the two existing **<ElementManifest>** nodes:

```

<ElementManifest Location="PageLayout.xml" />
<ElementFile Location="DivisionArticleImageLeft.aspx" />
<ElementFile Location="SamplePageLayoutPreview.gif" />

```

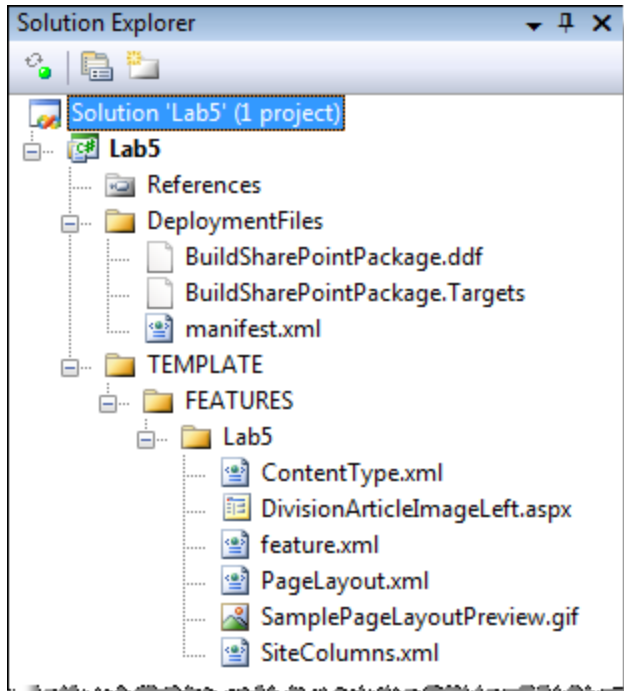
- Open the **BuildSharePointPackage.ddf** file and add the following lines where the other files from the Feature are added to the package:

```

TEMPLATE\FEATURES\Lab5\PageLayout.xml
TEMPLATE\FEATURES\Lab5\DivisionArticleImageLeft.aspx
TEMPLATE\FEATURES\Lab5\SamplePageLayoutPreview.gif

```

- When you save everything, your project should look like the following image:



6. Now it's time to deploy the WSS solution package.
7. First the WSS solution package must be deployed. Open a command prompt and navigate to the following directory:

```
c:\Program Files\Common Files\Microsoft Shared\web server extensions\12\BIN
```

8. Enter the following command into the command line window and hit **Enter**:

```
stsadm -o addsolution -filename c:\Student\Labs\05_PageLayouts\Lab\wsp\Debug\Lab5.wsp
```

9. Launch Central Administration by selecting **Start » All Programs » Microsoft Office Server » SharePoint 3.0 Central Administration**.
10. From the **Central Administration** site, select the **Operations** tab and then select **Solution management** under the **Global Configuration** section.
11. On the **Solution Management** page, click the link on **lab5.wsp**.
12. On the **Solution Properties** page, select **Deploy Solution**.
13. On the **Deploy Solution** page, specify **Now** in the **Deploy When?** section and click **OK**.
14. Test the Feature by browsing to the **http://wcm.litwareinc.com/** site and select **Site Actions » Site Settings » Modify All Site Settings**.
15. On the **Site Settings** page, select **Site collection features** under the **Site Collection Administration** section.
16. On the **Site Collection Features** page, click **Activate** on the **Lab 5 - Working with Provisioned Site Columns, Content Types and Page Layouts Feature**.

17. With the Feature activated, go check the new site columns created in the **Site Column Gallery** and content type in the **Content Type Gallery**. Then create a new page based off the new page layout in the **Press Releases** section.

At this point you have now created site columns, content types and a page layout using the SharePoint browser interface and SharePoint Designer. You then created a Feature that does the same thing but instead of creating everything in a customized sense within the content database, everything is template!

Lab 06: Extending the Out-Of-The-Box Authoring Experience

Lab Time: 60 minutes

Lab Overview: In this lab you will practice extending the out-of-the-box authoring experience in a Publishing site. First you will leverage WSS' custom actions and augment the SharePoint navigation user experience. Next, you will utilize the Edit Mode Panel within a page layout in order to customize the experience for both edit and display modes. Finally, you will extend the Page Editing Toolbar by adding new menu items to the Page Edit Menu and Quick Access Buttons areas.

Exercise 1: Adding new items to SharePoint menus using custom actions

In this exercise you will create a Feature that will add menu items to various SharePoint menus using custom action definitions within an element manifest file within a feature.

Since you have now created and customized Visual Studio projects that compile then package the contents of the project into a WSS solution package, as well as creating a project that doesn't compile anything but simply packages the contents into a WSS solution package, we have given you a project to get started with. This project already has the directory structure created, the deployment files added and the changes to the project file to run the automated WSS solution package creation.

1. In Visual Studio, open the **Lab6Exercise1** solution located in the following directory:

```
c:\Student\Labs\06_Extensibility\Lab\Lab6Exercise1\Lab6Exercise1.sln
```

2. The first thing you need to do is create the Feature definition file. Create a new XML file named **feature.xml** in the **Lab6Exercise1** Feature folder in the project, filling it with the following markup:

```
<?xml version="1.0" encoding="utf-8" ?>
<Feature xmlns="http://schemas.microsoft.com/sharepoint/"
  Id="27966B78-C2E9-4C80-A538-E8AEF0356BDB"
  Title="Lab 6 - Adding New Items To SharePoint Menus Using Custom Actions"
  Hidden="FALSE"
  Scope="Site"
  Version="1.0.0.0">

  <ElementManifests>
    <ElementManifest Location="elements.xml"/>
  </ElementManifests>

</Feature>
```

3. Next, create an element manifest files listed in the Feature definition file. Create a new XML file in the **Lab6Exercise1** Feature folder in the project named **elements.xml**.

4. Open the **elements.xml** file, add the following XML markup (you can omit the XML comments, they are here just as a comment to what the XML is doing):

```
<?xml version="1.0" encoding="utf-8" ?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">

  <!-- create a new custom action group in Site Settings -->
  <CustomActionGroup
    Id="95B43F4D-E66D-4969-809E-D189E7BB388A"
    Location="Microsoft.SharePoint.SiteSettings"
    Title="Lab 6"
    Sequence="1000" />

  <!-- add a new link to site settings -->
  <CustomAction
    Id="834219CB-2595-493D-BA2F-A715A64CBC29"
    GroupId="95B43F4D-E66D-4969-809E-D189E7BB388A"
    Location="Microsoft.SharePoint.SiteSettings"
    Sequence="10"
    Title="Lab 6"
    Description="Takes the user to wcm.litwareinc.com">
    <UrlAction Url="http://wcm.litwareinc.com" />
  </CustomAction>

  <!-- add a new link to the site actions menu -->
  <CustomAction
    Id="4B10BFFB-4E66-4872-88C9-C557F8A2B2A5"
    Location="Microsoft.SharePoint.StandardMenu"
    GroupId="SiteActions"
    Sequence="1000"
    Title="Lab 6"
    Description="Takes the user to wcm.litwareinc.com">
    <UrlAction Url="http://wcm.litwareinc.com" />
  </CustomAction>

  <!-- add a new link to the document library ECB menu of all sites -->
  <CustomAction
    Id="D917A1CF-9F9C-4FCA-855A-910814E38759"
    RegistrationType="List"
    RegistrationId="101"
    ImageUrl="/_layouts/images/plslforw.gif"
    Location="EditControlBlock"
    Sequence="100"
    Title="Lab 6">
    <UrlAction Url="~site/default.aspx?listid={ListId}&itemid={ItemId}"/>
  </CustomAction>
</Elements>
```

5. Next, add the manifest file for the Feature and necessary entries in the DDF file to package up the required files. Create a new XML file named **manifest.xml** to the **DeploymentFiles** folder in the project and add the following XML markup:

```
<?xml version="1.0" encoding="utf-8" ?>
<Solution xmlns="http://schemas.microsoft.com/sharepoint/"
  SolutionId="F24B6CA8-56D5-4500-9E6D-110F8DB3B5BB"
  DeploymentServerType="WebFrontEnd"
  ResetWebServer="FALSE">

  <FeatureManifests>
    <FeatureManifest Location="Lab6Exercise1\feature.xml"/>
  </FeatureManifests>

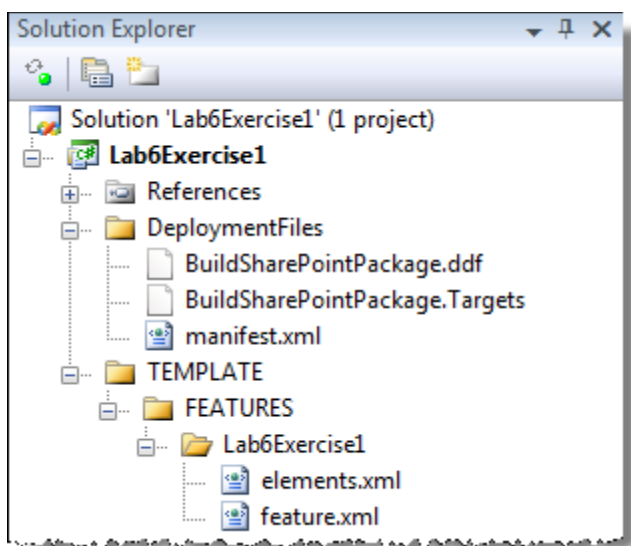
</Solution>
```

6. Finally, add the following lines to the **BuildSharePointPackage.ddf** file between the comments:

```
DeploymentFiles\manifest.xml

.Set DestinationDir=Lab6Exercise1
TEMPLATE\FEATURES\Lab6Exercise1\feature.xml
TEMPLATE\FEATURES\Lab6Exercise1\elements.xml
```

7. When you save everything, your project should look like the following image:



8. Now it's time to build deploy the WSS solution package. Build the project in Visual Studio to create the WSP file.
9. First the WSS solution package must be deployed. Open a command prompt and navigate to the following directory:

```
c:\Program Files\Common Files\Microsoft Shared\web server extensions\12\BIN
```

10. Enter the following command into the command line window and hit Enter:

```
stsadm -o addsolution -filename
c:\Student\Labs\06_Extensibility\Lab\Lab6Exercise1\wsp\Debug\Lab6Exercise1.wsp
```

11. Launch Central Administration by selecting **Start » All Programs » Microsoft Office Server » SharePoint 3.0 Central Administration**.
12. From the **Central Administration** site, select the **Operations** tab and then select **Solution management** under the **Global Configuration** section.
13. On the **Solution Management** page, click the link on **lab6exercise1.wsp**.
14. On the **Solution Properties** page, select **Deploy Solution**.
15. On the **Deploy Solution** page, specify **Now** in the **Deploy When?** section and click **OK**.
16. Test the Feature by browsing to the **http://wcm.litwareinc.com/** site and select **Site Actions » Site Settings » Modify All Site Settings**.

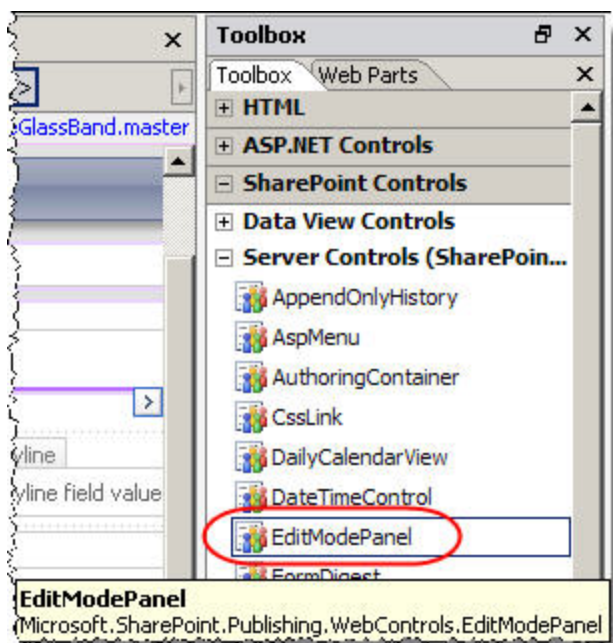
17. On the **Site Settings** page, select **Site collection features** under the **Site Collection Administration** section.
18. On the **Site Collection Features** page, click **Activate** on the **Lab 6 - Adding New Items To SharePoint Menus Using Custom Actions Feature**.
19. With the Feature activated, click through the site to see the actions that were added. You should see a new group on the site's Site Settings page, a new item in the Site Actions menu, and a new item in the ECB menu in document libraries (hint: use the Site Collection Images library which contains a few default images).

In this exercise you saw how to build a feature that created new menu items and menu groups. You can use this technique to add links to the SharePoint navigation menus for your own functionality & administration pages.

Exercise 2: Utilizing the Edit Mode Panel

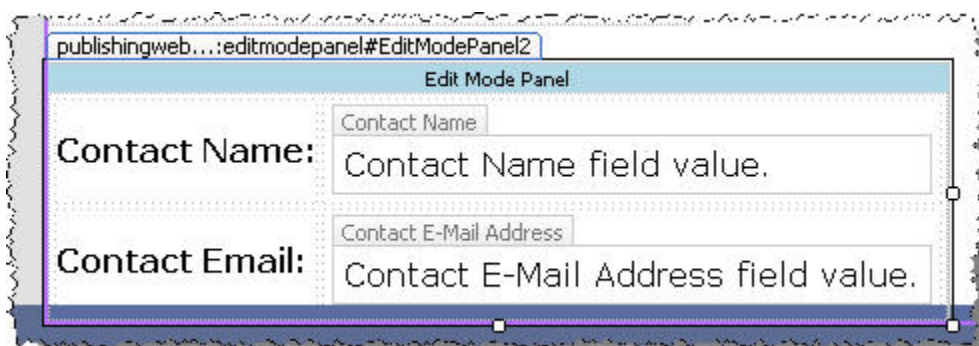
In this exercise you will see how to leverage the edit mode panel to provide content authors a way to display controls to change the page's contact person when in edit mode, but to not show this when in display mode.

1. Launch **SharePoint Designer** and open the site **<http://wcm.litwareinc.com>**.
2. Open the page layout **ArticleLeft.aspx** from within the **_catalogs/masterpage** folder within the **Folder List** tool window. When prompted, check the file out.
3. While this file currently leverages an existing edit mode panel, you will create a new instance of one. From the **Toolbox** tool window, select **EditModePanel** within the **SharePoint Controls / Server Controls** section and drag it onto the design surface just below the existing edit mode panel. The following image shows where you'll find the **EditModePanel**:



*If you have problems dragging the **EditModePanel** control onto the design surface, you may need to create some space for it first. The easiest way to do this is to switch to the **Code** view, add some text at the end of the **PlaceHolderMain** content placeholder section (usually at the end of the file), switch back to the Design view, drag the **EditModePanel** where the text is that you just added, and then remove the text.*

- Next, add a HTML table to the edit mode panel you just added to the page with two rows and two cells in each row. In the first row, add the label **Contact Name:** in the first cell and drag the **Contact Name** control from the **Toolbox** tool window within the **SharePoint Controls / Page Fields** section in the second cell. Then, do the same for the **Contact E-Mail** Address field in the second row. You should have something that looks similar to the following image on the **ArticleLeft.aspx** page layout:



- Save all changes, check-in the file, publish a major version when prompted and approve the publishing request file.
- Now it's time to test the edit mode panel. Browse to the <http://wcm.litwareinc.com/PressReleases> site and create a new page using the page layout (**Article Page**) **Article page with image on left**. Notice that when the page is created, you now have some contact information the content author can add to the page, as shown in the following image:



- Select **Page » Page Settings and Schedule** from the **Page Editing Toolbar**. When the **Page Settings** page loads, scroll down to the **Page Contact** section and select **Enter contact information**, but don't enter any information... just click **OK**.

8. Switch over to edit mode if you aren't already. Enter a name & contact email address, along with other content data on the page and select Check In to Share Draft from the Page Editing Toolbar.
9. Notice that the contact table you created in the page layout is not rendered when the page is not in edit mode; it is not visible in display mode.
10. After the page refreshes, select **Page » Page Settings and Schedule**. Scroll down to the Page Contact section and notice that the information you entered when editing the page is now shown.

In this exercise you saw how you can use the edit mode panel to show a customized edit mode experience for the content author. This same control can be used to show content when in display mode as well using the PageDisplayMode attribute.

Exercise 3: Adding custom menu items to the Page Editing Toolbar's Quick Access Button section

In this exercise you will add a new item to the Quick Access Button section within the Page Editing Toolbar. This item will always be visible and will take the user to the list item

Since you have now created and customized Visual Studio projects that compile then package the contents of the project into a WSS solution package, as well as creating a project that doesn't compile anything but simply packages the contents into a WSS solution package, we have given you a project to get started with. This project already has the directory structure created, the deployment files added, the changes to the project file to run the automated WSS solution package creation and the project has been signed with a key.

1. In **Visual Studio**, open the **Lab6Exercise3** solution located in the following directory:

```
c:\Student\Labs\06_Extensibility\Lab\Lab6Exercise3\Lab6Exercise3.sln
```

2. The first thing you need to do is add a few references to the project. Add the following references to the project by right-clicking the References folder within the Lab6Exercise3 project in the **Solution Explorer** tool window:
 - **Microsoft.SharePoint** (hint: Component Name = Windows SharePoint Services)
 - **Microsoft.SharePoint.Publishing** (hint: Component Name = Microsoft Content Publishing and Management)
 - **System.Web**
3. Now, with the necessary references, create a new class named **PagePropertiesMenuItem.cs** in the root of the **Lab6Exercise3** project. This class will contain the logic for the button that will be placed on the Quick Access Buttons area within the Page Editing Toolbar.

4. Open the **PagePropertiesMenuItem.cs** file. The first thing you need to do is add a few namespace references to make typing a little easier as we add the necessary code. Add the following references to the top of the file:

```
using Microsoft.SharePoint;
using Microsoft.SharePoint.Publishing.WebControls;
using Microsoft.SharePoint.Publishing.WebControls.EditingMenuActions;
```

5. Next, configure the class to inherit from the **Microsoft.SharePoint.Publishing.WebControls.EditingMenuActions.ConsoleAction** class by changing the class declaration to the following:

```
public class PagePropertiesMenuItem : ConsoleAction
```

While you are about to add a few public properties to the **ConsoleAction**, you could alternatively set these properties declaratively with XML which you'll get to later in this exercise. While setting properties via XML is easier as you have no assembly to deploy, it doesn't provide as much control in your custom control. You'll notice that all properties you override only implement the **get{ }** portion of the property... not the **set{ }**. This gives you much more control over what you want to display on your **ConsoleAction**.

6. Next, you need add code to define who can see the menu item and when it is visible in the Page Editing Toolbar. Add the following two property overrides to the **PagePropertiesMenuItem.cs** file which will show the button regardless of the user's permissions (**UserRights = EmptyMask**) and to be shown whenever the Page Editing Toolbar is visible (**RequiredStates = EditingMenuEnabled**):

```
public override SPBasePermissions UserRights {
    get { return SPBasePermissions.EmptyMask; }
}

public override AuthoringStates RequiredStates {
    get { return AuthoringStates.EditingMenuEnabled; }
}
```

7. With the permissions and conditions set, the next thing you should set is the icon to be shown in the Quick Access Button. Add the following code to the following code to the **PagePropertiesMenuItem.cs** file to use the info icon that's included in a typical WSS v3 installation:

```
public override string ImageUrl {
    get {
        return "~/_layouts/images/info16by16.gif";
    }
}
```

8. Finally, you need to set the URL so the button will actually do something when clicked. In this case, you want to specify the URL of the View Properties page within the Pages list and pass in the ID of the page within the list. Add the following code to the following code to the **PagePropertiesMenuItem.cs** file to do this:

```
public override string NavigateUrl {
    get {
        return
String.Format("javascript:window.location='{0}/Pages/Forms/DispForm.aspx?ID={1}';",
        SPContext.Current.Web.Url.ToString(),
```

```
        SPContext.Current.ListItem.ID.ToString());  
    }  
}
```

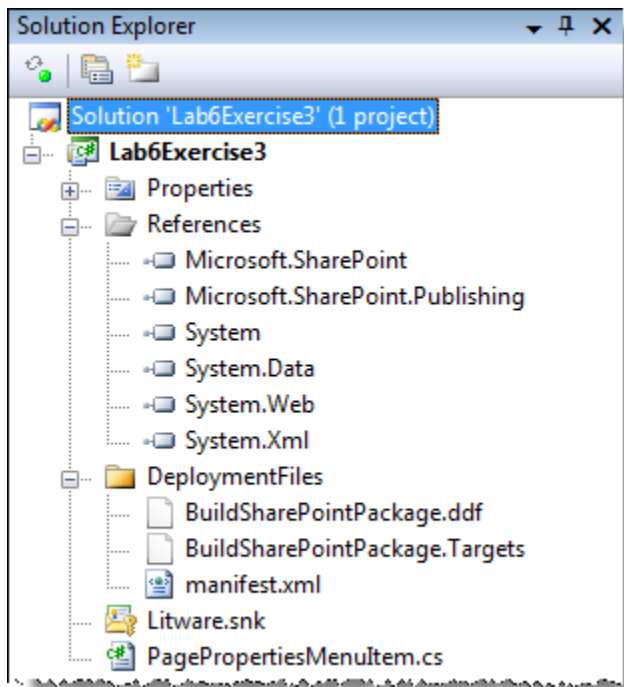
9. Next, you need to add the manifest file for the WSP and add the necessary entries in the DDF file to package up the required files. Create a new XML file named **manifest.xml** in the **DeploymentFiles** folder and add the following XML markup:

```
<?xml version="1.0" encoding="utf-8" ?>  
<Solution xmlns="http://schemas.microsoft.com/sharepoint/"  
  SolutionId="00EE4F8F-0C15-4F3C-B5E7-F2D5A75BB79E"  
  DeploymentServerType="WebFrontEnd"  
  ResetWebServer="FALSE">  
  
  <Assemblies>  
    <Assembly DeploymentTarget="GlobalAssemblyCache" Location="Lab6Exercise3.dll">  
      <SafeControls>  
        <SafeControl Namespace="Lab6Exercise3" TypeName="*" Safe="True" />  
      </SafeControls>  
    </Assembly>  
  </Assemblies>  
  
</Solution>
```

10. Finally, add the following lines to the BuildSharePointPackage.ddf file between the comments:

```
DeploymentFiles\manifest.xml  
bin\debug\Lab6Exercise3.dll
```

11. When you save everything, your project should look like the following image:



12. Build the **Lab6Exercise3** project to compile the code and package everything into a solution package.

Now it's time to deploy the WSS solution package.

13. First the WSS solution package must be deployed. Open a command prompt and navigate to the following directory:

```
c:\Program Files\Common Files\Microsoft Shared\web server extensions\12\BIN
```

14. Enter the following command into the command line window and hit **Enter**:

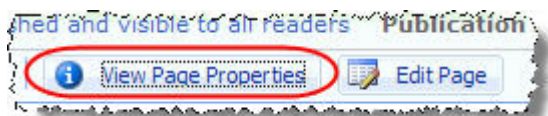
```
stsadm -o addsolution -filename  
c:\Student\Labs\06_Extensibility\Lab\Lab6Exercise3\wsp\Debug\Lab6Exercise3.wsp
```

15. Launch Central Administration by selecting **Start » All Programs » Microsoft Office Server » SharePoint 3.0 Central Administration**.
16. From the **Central Administration** site, select the **Operations** tab and then select **Solution management** under the **Global Configuration** section.
17. On the **Solution Management** page, click the link on **lab6exercise1.wsp**.
18. On the **Solution Properties** page, select **Deploy Solution**. On the **Deploy Solution** page, specify **Now** in the **Deploy When?** section. In the **Deploy To?** section, select **http://wcm.litwareinc.com** and click **OK**.
19. With the assembly added to the GAC and SafeControl entry added, you need to now make SharePoint aware of the ConsoleAction. Launch **SharePoint Designer** and open the site **http://wcm.litwareinc.com**.
20. Open the XML file **CustomQuickAccess.xml** from within the **_catalogs/masterpage/Editing Menu** within the **Folder List** tool window. When prompted, check the file out.
21. Add the following XML to the **CustomQuickAccess.xml** file to declare and add the button to the page:

```
<?xml version="1.0" encoding="utf-8" ?>  
<Console>  
  <references>  
    <reference TagPrefix="wcm401"  
      assembly="Lab6Exercise3, Version=1.0.0.0, Culture=neutral,  
      PublicKeyToken=d4e5777b16a5749f"  
      namespace="Lab6Exercise3" />  
  </references>  
  
  <structure>  
    <ConsoleNode Sequence="1"  
      ConfigMenu="Add"  
      DisplayText="View Page Properties"  
      UseResourceFile="false"  
      Action="wcm401:PagePropertiesMenuItem"  
      ID="wcm401PagePropertiesMenuItemQuickAccess" />  
  </structure>  
</Console>
```

22. Save your changes, check-in the **CustomQuickAccess.xml** file, publish and approve it.
23. Open a browser and navigate to any page within the **http://wcm.litwareinc.com** site and turn on the Page Editing Toolbar if it isn't visible by selecting **Site Actions » Show Page Editing Toolbar**. You should see the new button in the far left part of

the **Quick Access Button** section of the **Page Editing Toolbar** as shown in the following image. Click the button to watch it take you to the display page for the page list item in the Pages list.



In this exercise you created and added a new button to the Quick Access Button area within the Page Editing Toolbar.

Lab 07: Custom Field Types & Controls

Lab Time: 60 minutes

Lab Overview: In this lab you will practice creating a custom field type and associated field control for use within a MOSS Publishing site. The field type will store four data elements as one item that make up a linked string of text with an associated icon. This new field type is called "Resource with Icon". The field control associated with this field type is used to provide a custom editing experience for content authors. Here's what the field will look like when you are finished with this lab:



Exercise 1: Creating a custom field type

In this exercise you will create a custom field type. At the end of this exercise the field type will not be complete and usable until it has an associated field control, but this lab is broken into two exercises for readability and future reference.

Similar to previous labs, since you have already created Visual Studio projects using the automated process of creating WSS solution packages, we have given you a project to get started with.

1. In Visual Studio, open the **Lab7** solution located in the following directory:

```
c:\Student\Labs\07_FieldTypesControls\Lab\Lab7.sln
```

2. The first thing you need to do is to add a few references to the project. Add the following references to the project by right-clicking the **References** folder within the project in the **Solution Explorer** tool window:
 - **Microsoft.SharePoint** (hint: Component Name = Windows SharePoint Services)
 - **Microsoft.SharePoint.Publishing** (hint: Component Name = Microsoft Content Publishing and Management)
 - **System.Configuration**

➤ **System.Web**

- Now, with the necessary references, create a new class named **FieldResourceIcon.cs** in the root of the project. This class will be the "hub" of the custom field type... it is where you will later tell SharePoint to get all information about the field type.
- This custom field type will store four pieces of data as a single data element. To do this, you will need to inherit from the **Microsoft.SharePoint.SPFieldMultiColumn** class. This class has two constructors that you will need to implement... add the following code to the **FieldResourceIcon.cs** class:

```
using System;
using System.Web.UI;
using Microsoft.SharePoint;
using Microsoft.SharePoint.WebControls;

namespace Lab7 {
    public class FieldResourceIcon : SPFieldMultiColumn {
        public FieldResourceIcon (SPFieldCollection fields, string fieldName)
            : base(fields, fieldName) { }

        public FieldResourceIcon (SPFieldCollection fields, string typeName, string
displayName)
            : base(fields, typeName, displayName) { }
    }
}
```

- With the field created, you now need to create a new class that will serialize/deserialize the value to/from the SharePoint content database. Create a new class named **FieldResourceIconValue.cs** in the root of the project and add the following code to it:

```
using System;
using Microsoft.SharePoint;

namespace Lab7 {
    public class FieldResourceIconValue : SPFieldMultiColumnValue {
        private const int NUM_OF_FIELDS = 4;

        public FieldResourceIconValue ()
            : base(NUM_OF_FIELDS) { }

        public FieldResourceIconValue (string value)
            : base(value) { }
    }
}
```

- SharePoint passes multi column fields back and forth using strings with the delimiter **;**. Thankfully, the **Microsoft.SharePoint.SPFieldMultiColumnValue** class handles the parsing of this delimited string for you, as long as you provide the information where each data element is found in the string. Add the following public properties to the class:

```
public string Title {
    get {
        return this[0];
    }
    set {
```



```
        this[0] = value;
    }
}

public string Description {
    get {
        return this[1];
    }
    set {
        this[1] = value;
    }
}

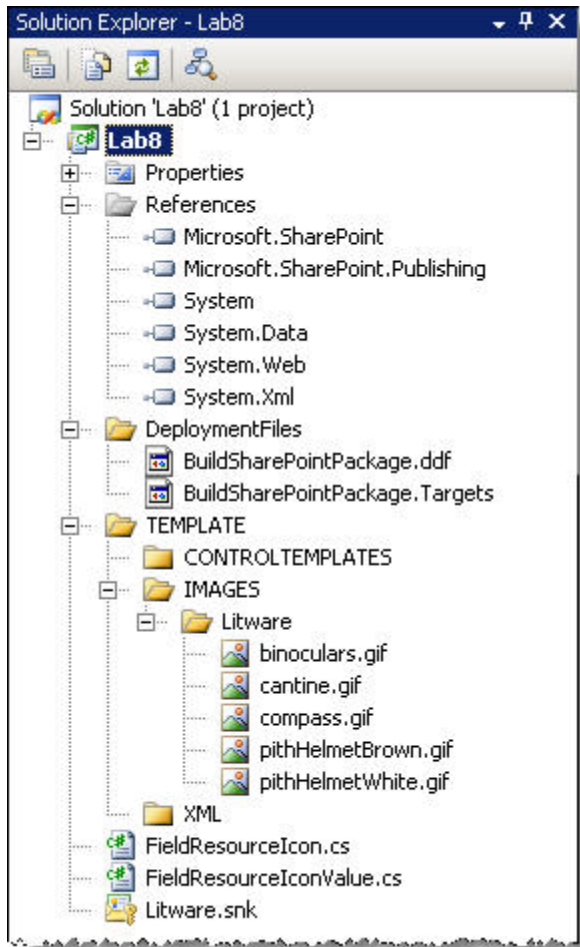
public string UrlTarget {
    get {
        return this[2];
    }
    set {
        this[2] = value;
    }
}

public string Icon {
    get {
        return this[3];
    }
    set {
        this[3] = value;
    }
}
```

7. With the value class created, you now need to wire it up to the field type class. Add the following overridden method to the **FieldResourceIcon.cs** class:

```
public override object GetFieldValue (string value) {
    if (string.IsNullOrEmpty(value))
        return null;
    return new FieldResourceIconValue(value);
}
```

8. At this point the code part of the field type is complete. Now, you need to add the images to the project, create the field type definition and package everything up.
9. Copy the images in the **Resources** folder within this lab (c:\Student\Labs\07_FieldTypesControls\Resources*.gif) into the **TEMPLATE\IMAGES\Litware** folder within the project, as shown in the following image:



10. Next, you need to create the field type definition file. This file will, when deployed, makes SharePoint aware of the new field type, and also specifies how the field should be rendered. Create a new **XML** file within the **TEMPLATE\XML** folder within the project named **fldtypes_Litware.xml**. This file is broken down into three pieces: (1) the definition and meta information about the field, (2) any default settings for the field and (3) the rendering information. For this lab, you will only specify the meta and rendering information. Add the following metadata to the **fldtypes_Litware.xml** file:

```
<?xml version="1.0" encoding="utf-8" ?>
<FieldTypes>
  <FieldType>
    <Field Name="TypeName">FieldResourceIcon</Field>
    <Field Name="ParentType">MultiColumn</Field>
    <Field Name="TypeDisplayName">Resource with Icon</Field>
    <Field Name="TypeShortDescription">Resource with Icon</Field>
    <Field Name="UserCreatable">TRUE</Field>
    <Field Name="ShowInListCreate">TRUE</Field>
    <Field Name="ShowInSurveyCreate">TRUE</Field>
    <Field Name="ShowInDocumentLibraryCreate">TRUE</Field>
    <Field Name="ShowInColumnTemplateCreate">TRUE</Field>
    <Field Name="FieldTypeClass">Lab7.FieldResourceIcon, Lab7, Version=1.0.0.0,
Culture=neutral, PublicKeyToken=d4e5777b16a5749f</Field>
  </FieldType>
</FieldTypes>
```

```
</FieldTypes>
```

11. This metadata tells SharePoint a few things about the field type. First, it specifies the type name, the type it is derived from, and the name of the type. Then it specifies the different conditions under which the field can be created. Finally, it specifies the fully qualified five-part-name of the class (and containing assembly) where the field type can be found.
12. Now you need to specify the rendering information. There are many different rendering patterns you can specify, including one to use for mobile devices, one to use when editing the value of the field, etc.... You will now implement a single rendering pattern that will be used when the field is being displayed (you do not need an edit pattern as it is addressed with a control in exercise 2). The pattern is defined using CAML... add the following CAML code into the **fldtypes_Litware.xml** file just before the closing **</FieldType>** node:

```
<RenderPattern Name="DisplayPattern">
  <Switch>
    <Expr><Column /></Expr>
    <Case Value="" />
    <Default>
      <HTML><![CDATA[
      <HTML><![CDATA[.gif" />&nbsp;<a href="]]></HTML>
      <Column SubColumnNumber="2" HTMLEncode="TRUE" />
      <HTML><![CDATA[" title="]]></HTML>
      <Column SubColumnNumber="1" HTMLEncode="TRUE" />
      <HTML><![CDATA[">]]></HTML>
      <Column SubColumnNumber="0" HTMLEncode="TRUE" />
      <HTML><![CDATA[</a>]]></HTML>
    </Default>
  </Switch>
</RenderPattern>
```

13. The next thing to do is to update the BuildSharePointPackage.ddf file to include all the necessary files in the WSS solution package. Add the following to the **BuildSharePointPackage.ddf** file between the comments:

```
DeploymentFiles\manifest.xml

bin\debug\Lab7.dll

.Set DestinationDir=IMAGES\Litware
TEMPLATE\IMAGES\Litware\binoculars.gif
TEMPLATE\IMAGES\Litware\cantine.gif
TEMPLATE\IMAGES\Litware\compass.gif
TEMPLATE\IMAGES\Litware\pithHelmetBrown.gif
TEMPLATE\IMAGES\Litware\pithHelmetWhite.gif

.Set DestinationDir=XML
TEMPLATE\XML\fldtypes_Litware.xml
```

14. Finally, the last thing to do is to create the WSS solution package manifest. Add a new XML file named **manifest.xml** to the **DeploymentFiles** folder in the project and add the following XML code to the file:

```
<Solution xmlns="http://schemas.microsoft.com/sharepoint/"
  SolutionId="733641DA-95D9-446E-812E-6070947171B2"
  DeploymentServerType="WebFrontEnd"
```

```

ResetWebServer="TRUE">

<Assemblies>
  <Assembly DeploymentTarget="GlobalAssemblyCache" Location="Lab7.dll">
    <SafeControls>
      <SafeControl Namespace="Lab7" TypeName="*" Safe="True" />
    </SafeControls>
  </Assembly>
</Assemblies>

<TemplateFiles>
  <TemplateFile Location="XML\fldtypes_Litware.xml"/>
  <TemplateFile Location="IMAGES\Litware\binoculars.gif" />
  <TemplateFile Location="IMAGES\Litware\cantine.gif" />
  <TemplateFile Location="IMAGES\Litware\compass.gif" />
  <TemplateFile Location="IMAGES\Litware\pithHelmetBrown.gif" />
  <TemplateFile Location="IMAGES\Litware\pithHelmetWhite.gif" />
</TemplateFiles>

</Solution>

```

15. Save all changes... that's it!

In this exercise you created a working field type. Unfortunately it is not very useful as you have not created an editing experience.

Exercise 2: Creating a custom editing experience (field control) for a field type

In this exercise you will create a field control that will provide a customized editing experience for the field type you just created.

1. Just like the display rendering (which you defined as a rendering pattern in the field type definition file), you can specify the editing experience using CAML in the field type definition. However, the markup can quickly get to be quite messy and pose a significant maintenance challenge. Another option is to use a standard ASP.NET 2.0 user control (*.ASCX) file... and link it up with a code behind. This approach is much easier for readability as well as maintenance moving forward.

Because the Visual Studio C# Class Library project template does not support creating an ASP.NET user control, you need to trick it a bit. Create a new text file with the name of **FieldResourceIconControl.ascx** in the **TEMPLATE\CONTROLTEMPLATES** directory within the project.

2. Next, you need to add the control directive to the page by adding the following code to the **FieldResourceIconControl.ascx** file:

```
<%@ Control Language="C#" %>
```

3. Now you need a reference to the **Microsoft.SharePoint.dll** assembly by adding the following code to the **FieldResourceIconControl.ascx** file:

```
<%@ Assembly Name="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c" %>
```

4. In order to specify a rendering template, you need to add a register directive to a specific namespace within the **Microsoft.SharePoint.dll** assembly by adding the following code to the **FieldResourceIconControl.ascx** file:

```
<%@ Register Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c" Namespace="Microsoft.SharePoint.WebControls"
TagPrefix="SharePoint" %>
```

5. Finally, you can now specify the rendering template for the editing experience. Add the following markup to the **FieldResourceIconControl.ascx** file:

```
<SharePoint:RenderingTemplate ID="FieldResourceIconControl" runat="server">
  <Template>
    <table class="ms-form">
      <tr>
        <td align="right">Title:</td>
        <td><asp:TextBox ID="ResourceTitle" runat="server" Size="30" /></td>
      </tr>
      <tr>
        <td align="right">Description:</td>
        <td><asp:TextBox ID="ResourceDescription" runat="server" Size="30" /></td>
      </tr>
      <tr>
        <td align="right">Url:</td>
        <td><asp:TextBox ID="ResourceUrl" runat="server" Size="30" /></td>
      </tr>
      <tr>
        <td align="right">Icon:</td>
        <td><asp:DropDownList ID="ResourceIcon" runat="server" Size="5" /></td>
      </tr>
    </table>
  </Template>
</SharePoint:RenderingTemplate>
```

6. Now that the rendering template has been created, you now need to create the code file that will handle the server-side logic for the control as well as wire it up to the field type. Create a new class in the root of the project named **FieldResourceIconControl.cs** and add the following code to it:

```
using System;
using System.Configuration;
using System.Web.UI.WebControls;
using Microsoft.SharePoint.WebControls;

namespace Lab7 {
  public class FieldResourceIconControl : BaseFieldControl {
  }
}
```

7. First add a constant containing the name of the rendering template and some internal fields that will be used as references to the ASP.NET controls in the rendering template. Add the following code to the **FieldResourceIconControl.cs** file:

```
private const string RENDERING_TEMPLATE_NAME = "FieldResourceIconControl";

protected TextBox _txbTitle;
protected TextBox _txbDescription;
protected TextBox _txbUrlTarget;
protected DropDownList _ddlIcon;
```

8. Next, override the **DefaultTemplateName** property on the **BaseFieldControl** class and return the name of the rendering template by adding the following code just after the fields you just added to the **FieldResourceIconControl.cs** file:

```
protected override string DefaultTemplateName {  
    get {  
        return RENDERING_TEMPLATE_NAME;  
    }  
}
```

9. Like any ASP.NET server control, much of the user interface work is done in the **CreateChildControls()** method... and this server control is no different. You now need to override the **CreateChildControls()** method to (1) verify that there is data in the field, (2) that the current mode of the control is the desired mode (the rendering template you created is only for the new/edit experience, so you want to exclude any other modes), (3) obtain references to the ASP.NET controls in the rendering template and finally (4) initialize any controls needed initialization. Add the following code to the **FieldResourceIconControl.cs** file:

```
protected override void CreateChildControls () {  
    // don't do anything if this is display mode or if the field has nothing assigned to it  
    if (this.Field == null || this.ControlMode == SPControlMode.Display || this.ControlMode  
    == SPControlMode.Invalid)  
        return;  
  
    base.CreateChildControls();  
  
    // get reference to the control  
    this._txbTitle = TemplateContainer.FindControl("ResourceTitle") as TextBox;  
    if (this._txbTitle == null)  
        throw new ConfigurationErrorsException("ResourceTitle TextBox not found. Corrupt  
control template.");  
  
    this._txbDescription = TemplateContainer.FindControl("ResourceDescription") as TextBox;  
    if (this._txbDescription == null)  
        throw new ConfigurationErrorsException("ResourceDescription TextBox not found.  
Corrupt control template.");  
  
    this._txbUrlTarget = TemplateContainer.FindControl("ResourceUrl") as TextBox;  
    if (this._txbUrlTarget == null)  
        throw new ConfigurationErrorsException("ResourceUrl TextBox not found. Corrupt  
control template.");  
  
    this._ddlIcon = TemplateContainer.FindControl("ResourceIcon") as DropDownList;  
    if (this._ddlIcon == null)  
        throw new ConfigurationErrorsException("ResourceIcon DropDownList not found.  
Corrupt control template.");  
  
    // init the drop down list  
    this._ddlIcon.Items.Add(new ListItem("Binoculars", "binoculars"));  
    this._ddlIcon.Items.Add(new ListItem("Cantine", "cantine"));  
    this._ddlIcon.Items.Add(new ListItem("Compass", "compass"));  
    this._ddlIcon.Items.Add(new ListItem("Brown pith helmet", "pithHelmetBrown"));  
    this._ddlIcon.Items.Add(new ListItem("White pith helmet", "pithHelmetWhite"));  
}
```

10. To finish off the server-side part of the rendering control, you need to override the **Value** property to set the ASP.NET controls when the control is loaded as well as

fetch the data from the controls when the property is read. Add the following code to the **FieldResourceIconControl.cs** file:

```
public override object Value {
    get {
        EnsureChildControls();

        // fetch values from rendering control return back as an object
        FieldResourceIconValue field = new FieldResourceIconValue();
        field.Title = this._txbTitle.Text.Trim();
        field.Description = this._txbDescription.Text.Trim();
        field.UrlTarget = this._txbUrlTarget.Text.Trim();
        field.Icon = this._ddlIcon.SelectedValue;

        return field;
    }
    set {
        EnsureChildControls();

        // if something stored in the value, init controls in rendering control
        if (value != null && !string.IsNullOrEmpty(value.ToString())) {
            FieldResourceIconValue field = new FieldResourceIconValue(value.ToString());
            this._txbTitle.Text = field.Title;
            this._txbDescription.Text = field.Description;
            this._txbUrlTarget.Text = field.UrlTarget;

            this._ddlIcon.Items.FindByValue(field.Icon).Selected = true;
        }
    }
}
```

11. Finally, you can now wire the control up to the field type created in exercise 1. Add the following code to override the **FieldRenderingControl()** property on the field type found within the **FieldResourceIcon.cs** file:

```
public override BaseFieldControl FieldRenderingControl {
    get {
        BaseFieldControl fieldControl = new FieldResourceIconControl();
        fieldControl.FieldName = this.InternalName;
        return fieldControl;
    }
}
```

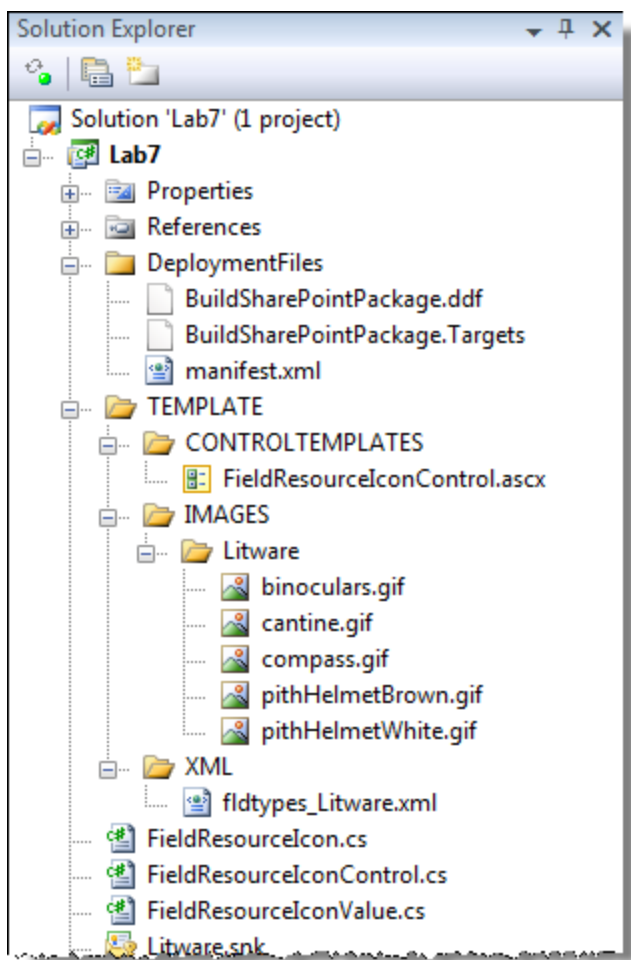
12. Now all that's left to do is include this file in the packaging of the WSS solution package! Add the following lines to the **BuildSharePointPackage.ddf** file to add the rendering template to the package:

```
.Set DestinationDir=CONTROLTEMPLATES
TEMPLATE\CONTROLTEMPLATES\FieldResourceIconControl.ascx
```

13. And lastly, add the following line to the **manifest.xml** file to tell SharePoint what to do with this file (this line should go with the other existing **<TemplateFile>** nodes):

```
<TemplateFile Location="CONTROLTEMPLATES\FieldResourceIconControl.ascx" />
```

14. When you save everything, your project should look like the following image:



Now it's time to deploy the WSS solution package.

15. First the WSS solution package must be deployed. Open a command prompt and navigate to the following directory:

```
c:\Program Files\Common Files\Microsoft Shared\web server extensions\12\BIN
```

16. Enter the following command into the command line window and hit Enter:

```
stsadm -o addsolution -filename  
c:\Student\Labs\07_FieldTypesControls\Lab\wsp\Debug\Lab7.wsp
```

17. Launch Central Administration by selecting **Start » All Programs » Microsoft Office Server » SharePoint 3.0 Central Administration**.

18. From the **Central Administration** site, select the **Operations** tab and then select **Solution management** under the **Global Configuration** section.

19. On the **Solution Management** page, click the link on **lab7.wsp**.

20. On the **Solution Properties** page, select **Deploy Solution**.

21. On the **Deploy Solution** page, specify **Now** in the **Deploy When?** section and click **OK**.

With the solution deployed, you now need to use the field type & control. To do this, you will add the field to an existing content type and page layout. Specifically, you'll add it to the page layout created by the utility you used in a previous lab.

22. Browse to the <http://wcm.litwareinc.com/> site and select **Site Actions » Site Settings » Modify All Site Settings**.

23. Select **Site content types** from the **Galleries** section.

24. On the **Site Content Type Gallery** page, scroll to the bottom and select **Widget Product Page** under the **Widget Content Builder** group.

*Note: The **Widget Content Builder** is included in the Student\Resources\Sample Data\WidgetContentBuilder folder. You can run that Feature (it is a hidden Feature so activation must be done via STSADM.EXE, or just run the batch file) or you can use the **Article Page** content type in the **Page Layout Content Types** group.*

25. On the **Site Content Type: Widget Product Page** page, scroll to the bottom and select **Add from new site column**.

26. Use the following information to create a new site column and click **OK**:

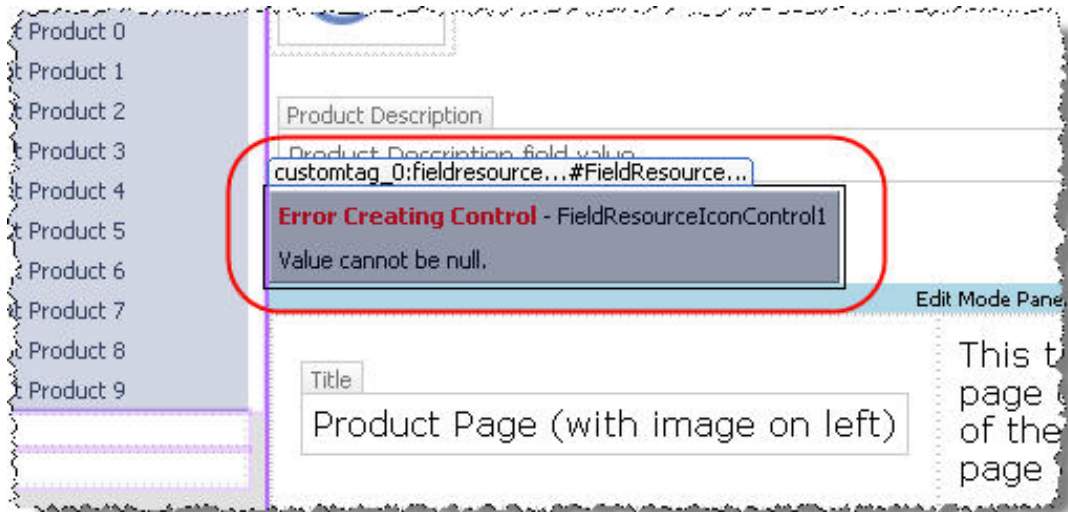
- **Column name:** ResourceWithIcon
- **The type of information in this column is:** Resource with Icon
- **Put this site column into:** Existing group: Widget Content Builder

27. With the content type updated, now you need to update an associated page layout. Check out and open the page layout **ProductPageLeft.aspx** (or whichever page layout matches the selected content type) in **SharePoint Designer** and switch to the **Code** view.

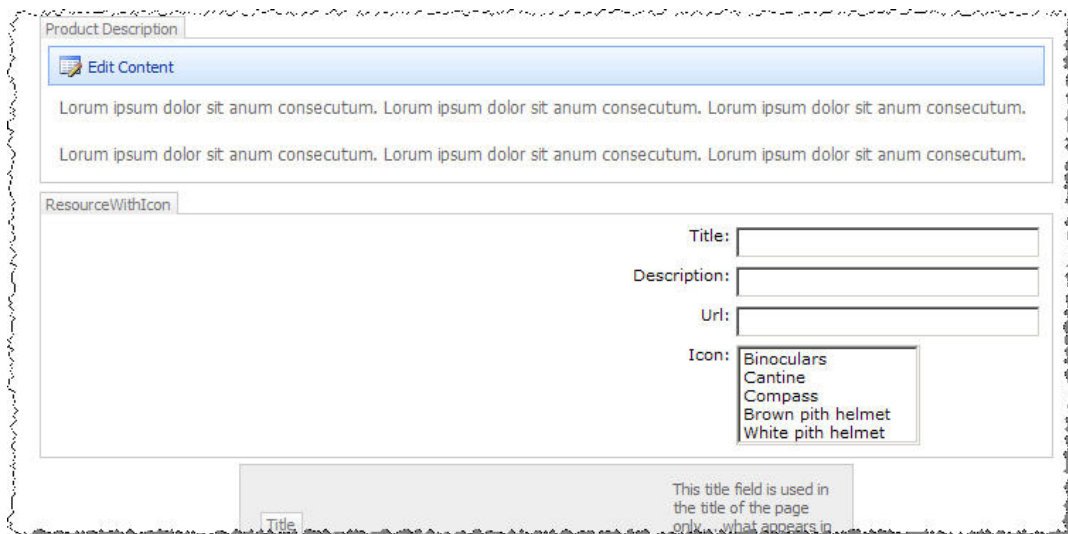
28. Scroll down to the **<PublishingWebControls:RichHtmlField>** field control used for the **ProductDescription** field is used (~ line 45). After the closing **<div>** add another opening & closing **<div>**, like the following image shows:

```
43 <div style="clear:both">&nbsp;</div>
44 <div class="pageContent">
45 <PublishingWebControls:RichHtmlField FieldName="ProductDescription" runat="server">
46 </div>
47 <div></div>
48 </div>
49 <PublishingWebControls:editmodepanel runat="server" id="editmodepanel1">
```

29. Switch back to **Design** view. Grab the **ResourceWithIcon** field from the content type and drop it into the **<div>** you just created as shown in the following image (ignore the error message):



30. Save your changes, go back to your browser and navigate to one of the Widget product pages such as **<http://wcm.litwareinc.com/Widgets/Pages/WidgetProduct0.aspx>**. Select **Site Actions » Edit Page**.
31. Scroll down and you should see the new field control, as shown in the following image:



32. Enter some information into the field control and then select **Page » Save and Stop Editing** to see the display mode of the field control, as shown in the following image:



In this exercise you created a custom field control and associated it with the custom field type you created in the first exercise.

Lab 08: Leveraging Publishing & Custom Web Parts

Lab Time: 60 minutes

Lab Overview: In this lab you will practice working with the additional Web Parts provided in Publishing sites, specifically the Content Query Web Part. At the end of the lab, you will practice creating a custom Web Part when the provided Web Parts don't satisfy your business requirements.

Exercise 1: Adding dummy content to a Publishing site

In this exercise you will use a provided WSS solution package that will create new site columns, content types, page layout as well as a new section within the top-level web in the Publishing site collection with dummy content for use within the remainder of this lab. You need some content in order to effectively work with the Content Query Web Part.

1. A sample application, packaged as a WSS solution package, is provided along with the source code that will create a new site within the Publishing site collection's top-level site named **Widgets** and fill this site with 10 content pages. The sample application is deployed using a Feature and Feature receiver. Upon activation, the Feature will create all the necessary site infrastructure assets, the Widgets site and content pages. Upon deactivation, everything is removed from the site.
2. The solution, and associated Visual Studio project, can be found in the following directory:

```
c:\Student\Resources\Sample Data\WidgetContentBuilder
```

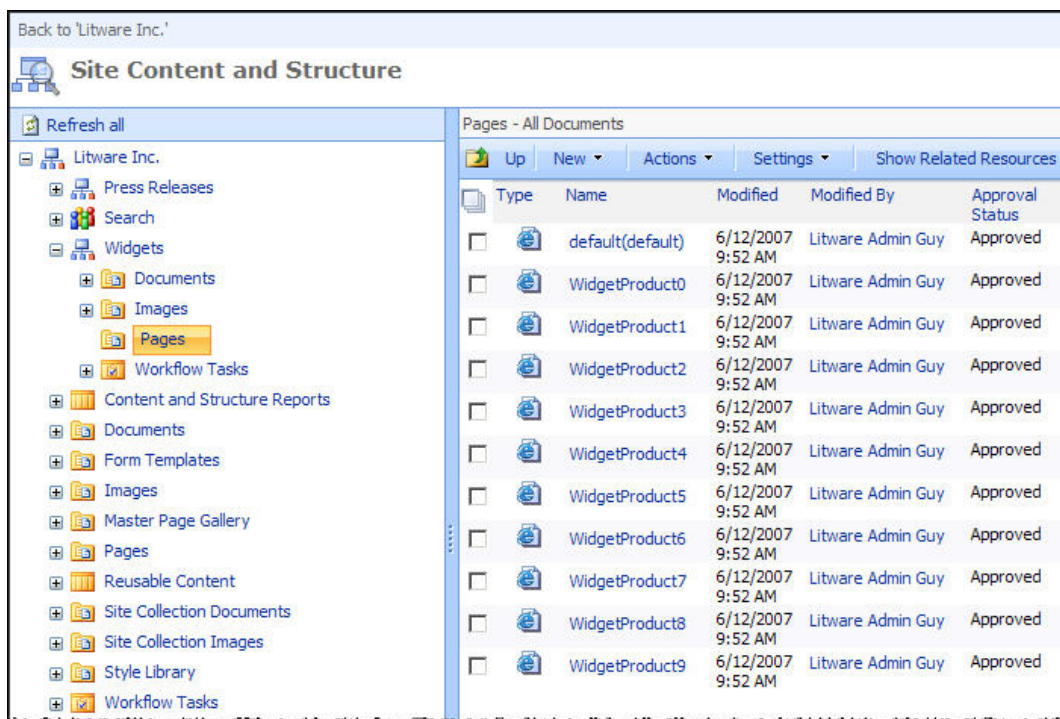
3. Within that directory you will find a file **wsp-install.bat**. Executing this file should (A) add the solution to the SharePoint farm's solution store, (B) deploy the solution and (C) activates the **WidgetContentBuilder** Feature for the Publishing site created in a previous lab. The WSP file can be found in the following directory:

```
c:\Student\Resources\Sample Data\WidgetContentBuilder\wsp\Debug
```

Note: The Feature is deployed as a hidden Feature. Activation/deactivation must be done using STSADM from the command line.

*Note 2: If you created your Publishing site using a different URL other than **http://wcm.litwareinc.com**, you will need to edit the **wsp-install.bat** file before executing the batch file. Another option is to let the activation line in the batch file fail, then manually activate the Feature from the command line.*

4. After activating the **WidgetContentBuilder** Feature (either using the batch file or from the command line), you should see a new **Widgets** site within the **http://wcm.litwareinc.com** site collection with 10 approved content pages as the following image shows:

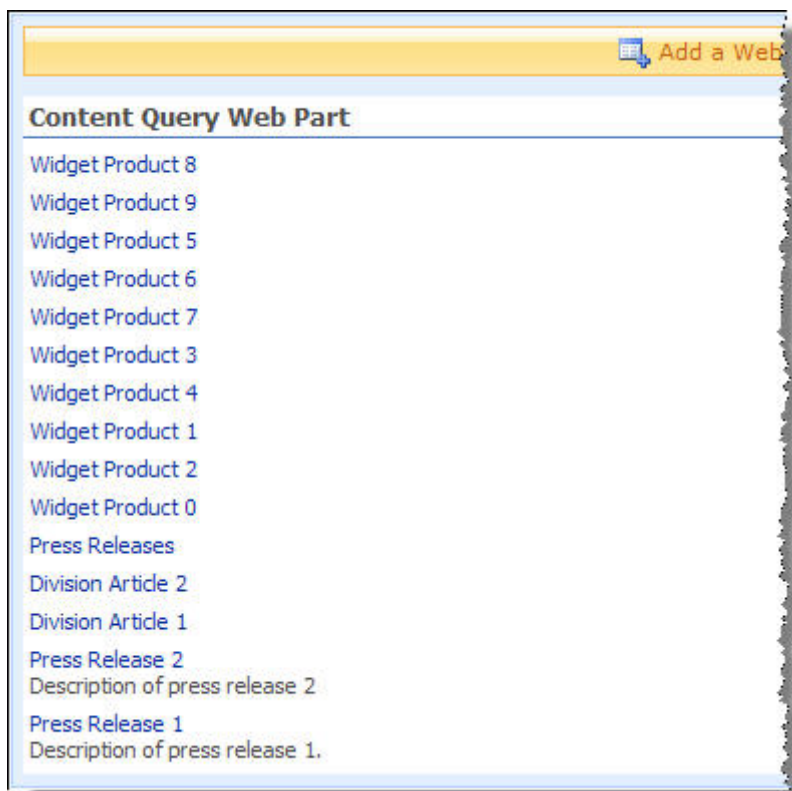


At this point, you have sample content that can be used for testing the Content Query Web Part.

Exercise 2: Implementing and customizing the Content Query Web Part

In this exercise you will add the Content Query Web Part to a page and perform some simple and advanced customization tasks on it. In the end, you would like to see a sorted list (by name) of the first 4 products from the North America division with the first bit of the page content listed as a teaser with a link to get the full post.

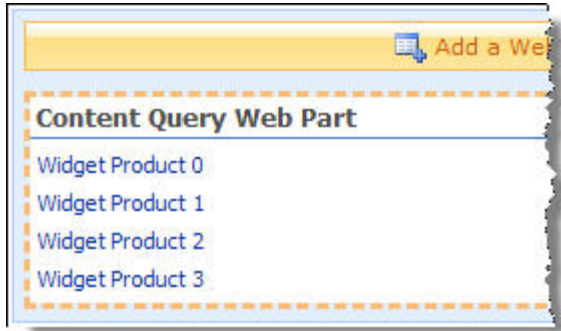
1. To start, add a Content Query Web Part to the Widget site's default page. Browse to the following page: **<http://wcm.litwareinc.com/Widgets/Pages/default.aspx>**.
2. Switch into edit mode by selecting **Site Actions » Edit Page**.
3. Scroll to the bottom of the page and click the **Add a Web Part** orange colored bar in the **Top** Web Part zone. Under the **All Web Parts / Default** section, check the box next to **Content Query Web Part** and click **Add**.
4. By default, the Content Query Web Part will display all content across the entire site collection... not what you want (as shown in the following image):



The first thing to do is to filter the scope and type of content.

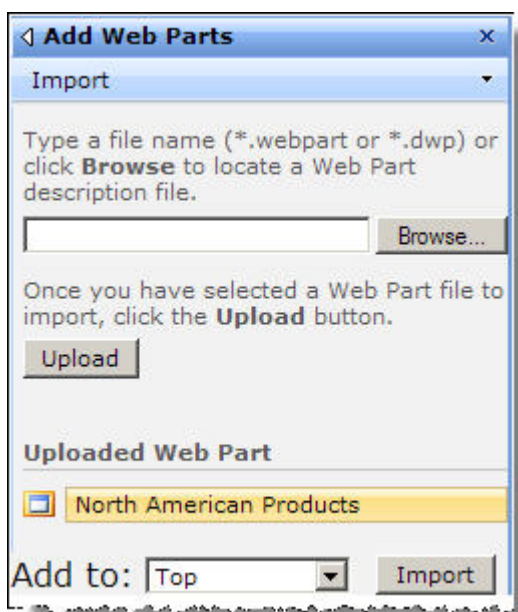
5. Select the **edit** menu in the upper-right corner of the **Content Query Web Part** and select **Modify Shared Web Part**. Expand the **Query** section and use the following information to select the desired settings:
 - **Source:** Show items from the following sites and all subsites: Widgets
 - **List Type:** Show items from this list type: Pages Library
 - Content Type:
 - i. **Show items of this content type group:** Widget Content Builder
 - ii. **Show items of this content type:** Widget Product page
 - iii. **Include child content types:** unchecked
6. Expand the **Additional Filters** section and set a single filter using the following information:
 - **Show items when:** [Division] [is equal to] [North America] (*don't include the brackets... they are used simply for grouping the instructions*)
7. Expand the **Grouping and Sorting** section and set a single filter using the following information:
 - **Group items by:** None
 - **Sort items by:** Name
 - **Show items in ascending order**

- **Limit the number of items to display:** checked
 - **Item limit:** 4
8. Once all the settings that can be set using the Content Query Web Part ToolPane are set, click **OK** at the bottom of the **ToolPane**. The resulting contents should display four pages of the Widget product pages 0-3 as shown in the following image:



9. This is as far as you can get without rolling up your sleeves and getting dirty with the code. To edit some of the more power properties of the Content Query Web Part, you need to edit the XML by hand. To do this, export the Web Part by selecting **edit** and then **Export....** Save the ***.webpart** file anywhere you like... such as the desktop.
10. Open the saved ***.webpart** file in Visual Studio. First, change the name of the Web Part. Search for the property **Title** (~ line 27) and change the contents of the XML element to **North American Products**.
11. Next, you need to add a few extra fields to the XML created by the Content Query Web part... search for the property **CommonViewFields** (~ line 68) and change the contents of the XML element to **ProductDescription,RichHTML;**. This tells the Content Query Web Part to pull an additional field (along with the data type) from the content and add to the resulting XML output: ProductDescription.
12. The Content Query Web Part uses XSL to transform the XML it generates to resulting HTML for rendering in the browser. The Publishing Portal site template adds quite a few style sheets to the Style Library list when the site is created. While you could customize one of these, it's better if you create your own XSL file as you may want to use one of the provided styles at a later point in your project. To use your own style sheet, you need to tell the Content Query Web Part to import the XSL style sheet you will create.
- Search for the property **ItemXslLink** (~ line 84) and change the contents of the XML element to **/Style Library/XSL Style Sheets/WCM401.xsl**.
13. Create a new XSL file and add it to the Style Library list. To do this, open **SharePoint Designer** and open the site **http://wcm.litwareinc.com**. It is easier to start by copying the existing ItemStyle.xsl file... select the file **Style Library\XSL Style Sheets\ItemStyle.xsl** by **right-clicking** it and selecting **Copy**, then **right-click** it again and select **Paste**. The file was pasted with the name of **ItemStyle_copy(1).xsl**... **rename** this file to **WCM401.xsl**.

14. Check out the **WCM401.xsl** file you just created by right-clicking it and selecting **Check Out** then open the file.
15. The opening **<xsl:stylesheet>** node contains a few schema declarations. The remainder of the file contains a few **<xsl:template>** sections. The first template, **Default**, is the one you will customize. Delete all other templates from the **WCM401.xsl** file. The file should now contain only ~47 lines of XML. Change the name attribute in the **<xsl:template>** node from **Default** to **WidgetProductList**. Also change the match attribute from ***** to **Row[@Style='WidgetProductList']**.
16. Save your changes to the XSL file.
17. Now, make sure the *.webpart file is valid. If you aren't already on the Widget's site default page, browse to it (<http://wcm.litwareinc.com/Widgets/Pages/default.aspx>). Now you need to import the *.webpart file. From the Page Editing Toolbar, select **Page » Add Web Parts » Import**. In the **ToolPane**, click **Browse** and select the *.webpart file you have been working with. Once selected, click **Upload** in the ToolPane. After the page issues a postback, you should see your Web Part listed below the Upload button in the Task Pane, as shown in the following image:

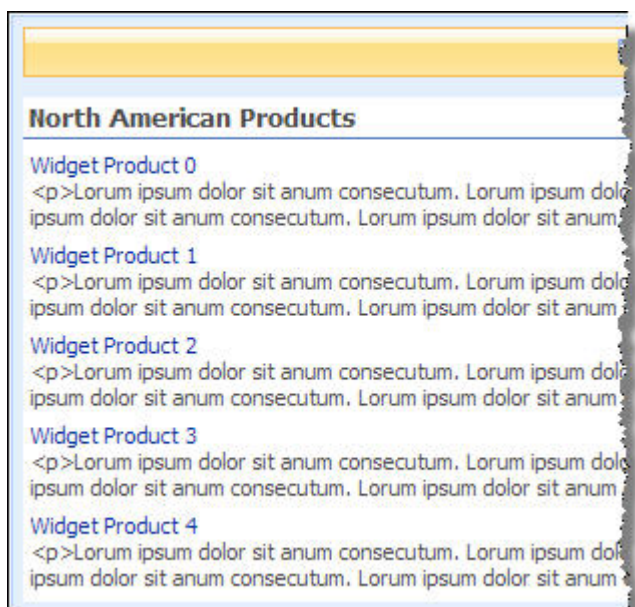


18. Add the Web Part to the page by either dragging it into a Web Part zone, or by selecting the zone from the ToolPane and clicking **Import**. You should see an instance of the new Content Query Web Part with a title of **North American Products** that contains no rendered results. That's because you need to set the item style to use. Do this by selecting **edit » Modify Shared Web Part** from the **North American Products** Content Query Web Part. Expand the **Presentation** section and scroll down to the **Styles** subsection. Notice the **Item style** is set to **WidgetProductList...** that's the only style in our XSL style sheet. You just need to apply it by clicking **OK** or **Apply** in the ToolPane.

The **North American Products** Web Part should now contain the same rendered results as the original Content Query Web Part. This is because you have simply copied the default rendering from the ItemStyle.xsl style sheet.

Now you are set to customize the XSL style sheet.

19. Go back to **SharePoint Designer** and open **/Style Library/XSL Style Sheets/WCM401.xsl** if it is not already open. The XSL template is broken into two sections. The first section (containing four `<xsl:variable>` elements) is used for validating XSL variables. The second section (starting with `<div>`) is used for rendering the output.
20. Change the description to show the contents of the article. Change the `<xsl:value-of select="@Description" />` node to `<xsl:value-of select="@ProductDescription" />` and save your changes.
21. Go back to the browser and reload (do not refresh) the page by clicking the **Widgets** link in the navigation. You will now see your modified style being used as shown in the following image:



Now we are showing the contents of the ProductDescription field... but the rendering is not ideal. Next you'll format the contents to not show the markup and show only the first 150 characters.

22. Go back to **SharePoint Designer** and open **/Style Library/XSL Style Sheets/WCM401.xsl** if it is not already open. You need to create an XSL function to parse a string, removing any markup. Add the following XSL just before the closing `</xsl:stylesheet>` node (you don't need to include the HTML comments, they are provided for extra documentation and readability):

```
<!-- function to remove any HTML markup from a string -->
<xsl:template name="PurgeMarkup">
  <!-- single parameter of the function -->
  <xsl:param name="contentToProcess" />
```

```

<xsl:choose>
  <!-- when a '<' is encountered... -->
  <xsl:when test="contains($contentToProcess, '&lt;')">
    <!-- assign a new variable the string from just after the opening '<' to
         just after the closing '>' and recursively call this function again -->
    <xsl:variable name="nextContentToProcess">
      <xsl:call-template name="PurgeMarkup">
        <xsl:with-param name="contentToProcess" select="substring-
after($contentToProcess, '&gt;')"/>
      </xsl:call-template>
    </xsl:variable>

    <!-- have the function return a string = concatenation of the
         string before and after processing -->
    <xsl:value-of select="concat(substring-before($contentToProcess, '&lt;'),
$nextContentToProcess)"/>
  </xsl:when>

  <!-- otherwise the content must not have an opening '<' so just kick it back -->
  <xsl:otherwise>
    <xsl:value-of select="$contentToProcess" />
  </xsl:otherwise>
</xsl:choose>
</xsl:template>

```

23. Then you need to add a new XSL variable that will contain the text that has been processed by the function you just created. Add the following XSL code after the other **<xsl:variable>** nodes in the top of your **WidgetProductList** XSL template (just before the opening **<div>** tag):

```

<!-- assign the variable 'ProductDescription' the processed value
      of the data field 'ProductDescription' passed by the CQWP
      that now contains no HTML markup -->
<xsl:variable name="ProductDescription">
  <xsl:call-template name="PurgeMarkup">
    <xsl:with-param name="contentToProcess" select="@ProductDescription" />
  </xsl:call-template>
</xsl:variable>

```

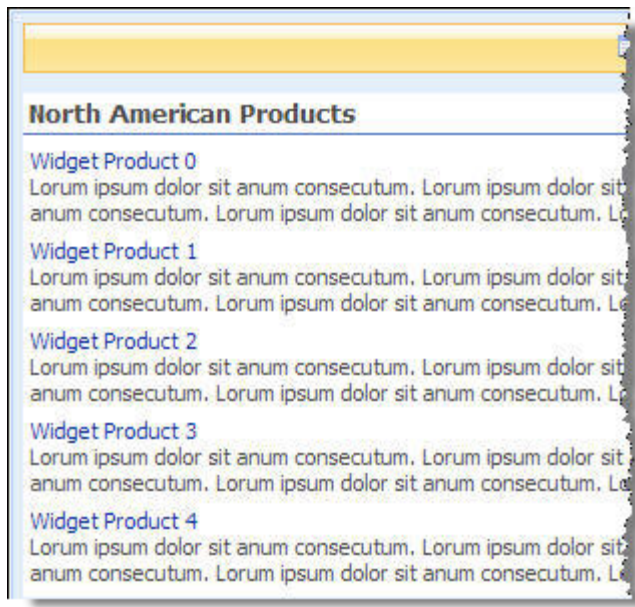
24. Finally, change the part of your rendering markup to use the variable instead of the value coming from the Content Query Web Part by changing the @ symbol to a \$ so the XSL node that displays the description looks like the following markup:

```

<xsl:value-of select="$ProductDescription" />

```

25. Save you changes, go back to the browser and select the **Widgets** site... as the following images shows, you should no longer see the HTML markup:

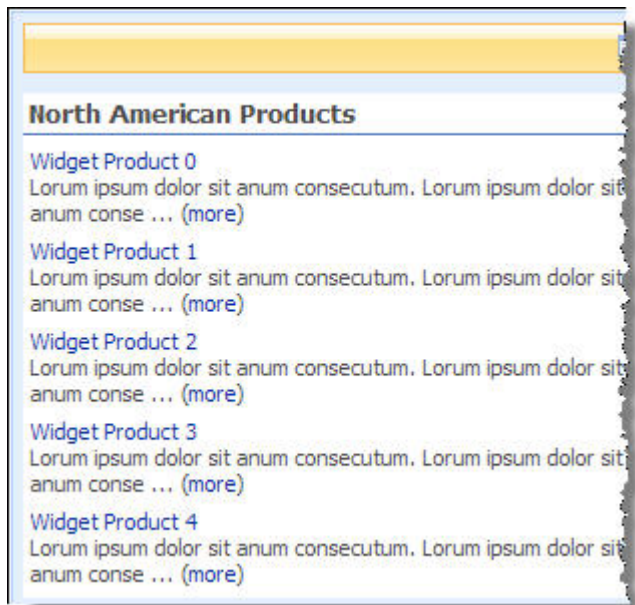


Almost done... now you need to truncate the text down to a small teaser.

26. Go back to SharePoint Designer and change the part of the **WidgetProductList** that displays the product description text to the following markup:

```
<div class="description">
  <xsl:value-of select="substring($ProductDescription, 1, 150)" />
  ... (<a href="{ $SafeLinkUrl}" target="{ $LinkTarget}">more</a>)
</div>
```

27. Save your changes, go back to the browser and select the Widgets site... as the following image shows, all your work for this Content Query Web Part is complete!



In this exercise you learned how to configure the Content Query Web Part as well as create a custom style sheet for use within a specific Content Query Web Part.

Exercise 3: Creating a custom Web Part

In this exercise you will create a custom Web Part because inevitably, you will reach a point where the provided Web Parts don't meet your business requirements.

Similar to previous labs, since you have already created Visual Studio projects using the automated process of creating WSS solution packages, we have given you a project to get started with.

1. In Visual Studio, open the **Lab8** solution located in the following directory:

```
c:\Student\Labs\08_WebParts\Lab\Lab8.sln
```

2. The first thing you need to do is create a new class that will be the Web Part. Create a new class named **CustomWebPart.cs** in the root of the project, replacing the default code with the following code:

```
using System;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;

namespace Lab8
{
    public class CustomWebPart: WebPart
    {
    }
}
```

3. Next, override the **CreateChildControls()** method to add two controls to the Web Part: a label displaying the current date & time and a button that when clicked, triggers a postback with a server side handler. Add the following code to the **CustomWebPart** class:

```
protected override void CreateChildControls()
{
    base.CreateChildControls();

    // create a new label containing the date & time and add
    // to to the controls collection
    Label lblDateTime = new Label();
    lblDateTime.Text = DateTime.Now.ToString();
    Controls.Add(lblDateTime);

    // create a new button that, when clicked, changes the web part
    // title to the current date & time
    Button btnDateTime = new Button();
    btnDateTime.Text = "Set Title to Current Date/Time";
    btnDateTime.Click += new EventHandler(OnDateTime_Click);
    Controls.Add(btnDateTime);
}
```

4. Finally, add the following code to implement the server side handler when the button is clicked:

```
protected void OnDateTime_Click(object sender, EventArgs e)
{
    this.Title = DateTime.Now.ToString();
}
```

5. The Web Part code is now complete. The only thing missing for the assembly is to flag it to the .NET Framework that it can be called/executed by assemblies that are not fully trusted. This is a requirement for Web Parts as the SharePoint assemblies are not running in full trust... rather a more restricted level of trust. Add the following line of code to the end of the **AssemblyInfo.cs** file to tell the .NET Framework that assemblies that aren't fully trusted can call & execute the Web Part assembly:

```
[assembly: System.Security.AllowPartiallyTrustedCallers]
```

6. Now you need to address the deployment part of the project. You will use a Feature to deploy this Web Part to a specific site. Create a new XML file named **feature.xml** in the **Lab8** Feature folder in the project, filling it with the following markup:

```
<?xml version="1.0" encoding="utf-8" ?>
<Feature xmlns="http://schemas.microsoft.com/sharepoint/"
  Id="5A461434-720A-463c-BB26-CC523A9902F9"
  Title="Lab 8 - Custom Web Parts"
  Hidden="FALSE"
  Scope="Site"
  Version="1.0.0.0">

  <ElementManifests>
    <ElementManifest Location="elements.xml" />
    <ElementFile Location="customWebPart.webpart" />
  </ElementManifests>

</Feature>
```

7. Next, create the element manifest listed in the Feature definition file. Create an XML file in the **Lab8** folder named **elements.xml** and fill it with the following XML markup:

```
<?xml version="1.0" encoding="utf-8" ?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Module Url="_catalogs/wp"
    RootWebOnly="TRUE">
    <File Url="CustomWebPart.webpart"
      Type="GhostableInLibrary">
      <Property Name="Group" Value="WCM401" />
      <Property Name="Title" Value="Lab 7 - Custom Web Part" />
    </File>
  </Module>
</Elements>
```

8. Finally, create another XML file named **customWebPart.webpart** in the **Lab8** folder and fill it with the following XML markup:

```
<?xml version="1.0" encoding="utf-8" ?>
<webParts>
  <webPart xmlns="http://schemas.microsoft.com/WebPart/v3">
    <metaData>
      <type name="Lab8.CustomWebPart, Lab8, Version=1.0.0.0, Culture=neutral,
        PublicKeyToken=d4e5777b16a5749f" />
      <importErrorMessage>Error importing the Web Part.</importErrorMessage>
    </metaData>
    <data>
      <properties>
        <property name="Title" type="string">Custom Web Part</property>
      </properties>
```

```

    </data>
  </webPart>
</webParts>

```

9. Now it is time to package the Feature and Web Part assembly up for deployment. Add a new XML file named **manifest.xml** to the **DeploymentFiles** folder in the project and add the following XML code to the file:

```

<?xml version="1.0" encoding="utf-8" ?>
<Solution xmlns="http://schemas.microsoft.com/sharepoint/"
  SolutionId="A305AC7A-4F65-4684-AC3B-62031FDBFE82"
  DeploymentServerType="WebFrontEnd"
  ResetWebServer="FALSE">

  <FeatureManifests>
    <FeatureManifest Location="Lab8\feature.xml" />
  </FeatureManifests>

  <Assemblies>
    <Assembly DeploymentTarget="WebApplication" Location="Lab8.dll">
      <SafeControls>
        <SafeControl Namespace="Lab8" Safe="True" TypeName="*" />
      </SafeControls>
    </Assembly>
  </Assemblies>

</Solution>

```

10. Finally, open the **BuildSharePointPackage.ddf** file in the **DeploymentFiles** folder and add the following lines between the comments:

```

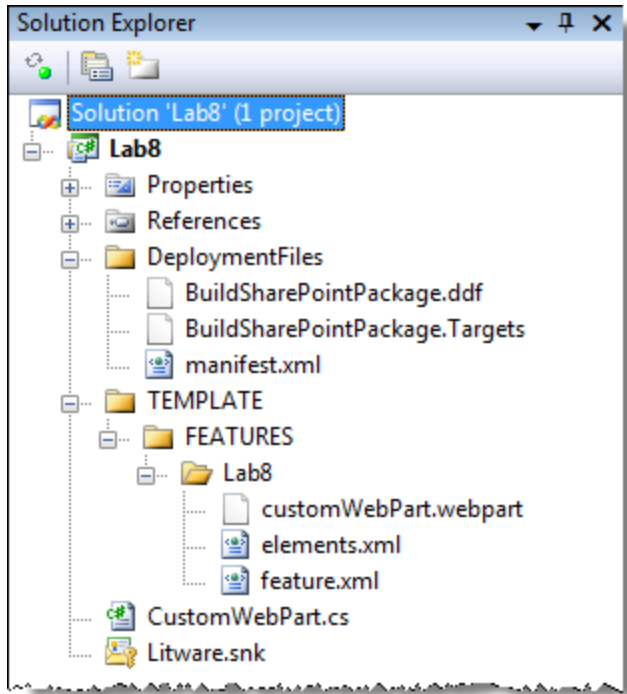
DeploymentFiles\manifest.xml

bin\debug\Lab8.dll

.Set DestinationDir=Lab8
TEMPLATE\FEATURES\Lab8\feature.xml
TEMPLATE\FEATURES\Lab8\elements.xml
TEMPLATE\FEATURES\Lab8\customWebPart.webpart

```

11. When you save everything, your project should look like the following image:



12. Now it's time to deploy the WSS solution package.

13. First the WSS solution package must be deployed. Open a command prompt and navigate to the following directory:

```
c:\Program Files\Common Files\Microsoft Shared\web server extensions\12\BIN
```

14. Enter the following command into the command line window and hit **Enter**:

```
stsadm -o addsolution -filename c:\Student\Labs\08_WebParts\Lab\wsp\Debug\Lab8.wsp
```

15. Launch Central Administration by selecting **Start » All Programs » Microsoft Office Server » SharePoint 3.0 Central Administration**.

16. From the **Central Administration** site, select the **Operations** tab and then select **Solution management** under the **Global Configuration** section.

17. On the **Solution Management** page, click the link on **lab8.wsp**.

18. On the **Solution Properties** page, select **Deploy Solution**.

19. On the **Deploy Solution** page, specify **Now** in the **Deploy When?** section and click **OK**.

20. Test the Feature by browsing to the <http://wcm.litwareinc.com/> site and select **Site Actions » Site Settings » Modify All Site Settings**.

21. On the **Site Settings** page, select **Site collection features** under the **Site Collection Administration** section.

22. On the **Site Collection Features** page, click **Activate** on the **Lab 8 - Custom Web Parts Feature**.

23. With the Feature activated, go back to the homepage of the site and add the Web Part to any Web Part zone. Notice when you click the button, the title of the Web Part changes.

In this exercise you created a custom Web Part that was deployed with a WSS Feature.

Lab 10: Understanding Workflow Foundation & Creating Custom Workflows

Lab Time: 120 Minutes

Lab Overview: In this lab, you will create a real world, practical workflow that is needed in many organizations that leverage a content management system such as Web Content Management within Office SharePoint Server 2007. Many organizations need to have a group of individuals review a piece of content before it is published for the rest of the world to see. However, within Litware, it is not important that specific people review a piece of content, rather what is important is that a majority of the reviewers approve the content before it is published. What Litware needs is a workflow where they can assign three people from the legal department to review a piece of content before it is published. However, it doesn't matter who reviews it, as long as a majority (2/3) of the reviewers approve or reject it.

The workflow that you will create in this lab will leverage InfoPath 2007 forms as the interaction vehicle between the users and workflow hosted within SharePoint. Upon initiation, it will prompt the user for three user names for the three reviewers as well as some instructions. A task will be created and assigned to each person. The reviewers then can approve or reject the workflow and enter some comments when they submit their decision. Once a majority is decided, which could be before all three decide, the workflow will terminate and approve or reject the page. To create this workflow you will do the following (7 exercises):

1. Create & configure the project
2. Create the workflow process & bind any necessary fields to the activities
3. Create & publish InfoPath 2007 forms
4. Add code to integrate the InfoPath forms and necessary logic
5. Package the workflow up for deployment
6. Implement the workflow
7. Test the workflow

Due to the nature of workflow and InfoPath form integration with workflows in SharePoint, this is a very large lab. Know that the complete solution is available to you if you want to copy & paste some code or if you need to refer to anything. Also, all steps are critical... and in the order they are presented. Workflows in SharePoint, especially those leveraging InfoPath forms are very finicky... so be astute when working through this lab!

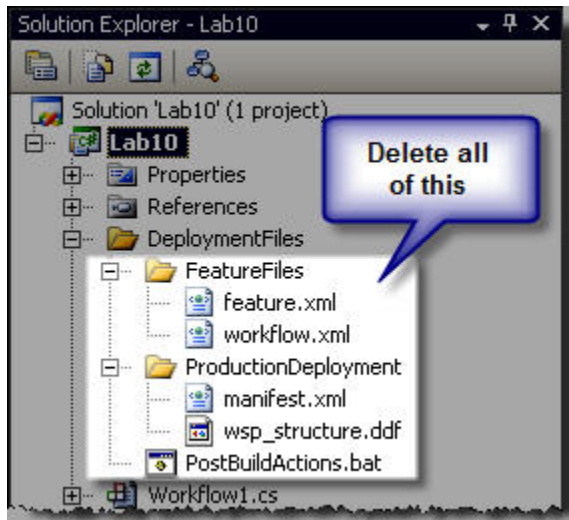
Exercise 1: Setup the sequential workflow project template

In this exercise you will create the project that will contain the workflow using a template provided with the MOSS 2007 SDK. The project that the template creates is very different than what you're used to working with in this course, so in this first exercise you will also do a few tweaks to get in a more familiar territory.

* Keep in mind that most of the stuff you do in this exercise is to apply consistency with this Visual Studio project and the others you've created in other labs in this course. With the exception of the project template selection, everything else is optional. However, the remainder of the lab is written assuming you did the steps in this exercise.

* You can optionally skip this exercise and go straight to exercise 2 using a sample that is included in the Resources folder. If you elect this approach, note the special instructions at the beginning of Exercise 2.

1. Open **Visual Studio** and create a new project using the template **SharePoint Server Sequential Workflow** which is found within the **Visual C# \ SharePoint** in the **Project types:** pane. Give this project a name of **Lab10**.
2. The first thing to do is to get rid of all the extra stuff included in the project template. Delete everything you see that is highlighted in the following image... what you'll be left with is a single **Workflow1.cs** file and an empty **DeploymentFiles** folder:



3. Next, copy the files **BuildSharePointPackage.ddf** & **BuildSharePointPackage.Targets** into the **DeploymentFiles** folder in the project from the following folder:

```
c:\Student\Labs\10_Workflow\Resources
```

4. Now you need to configure the project to run the custom MSBuild targets file every time the project is compiled in order to automatically create the WSP file used for deployment. Unload the project by right-clicking **Lab10** in the **Solution Explorer** tool window and selecting **Unload Project**. Then right-click the unloaded project in the **Solution Explorer** tool window and select **Edit Lab10.csproj**. Scroll to the bottom of the page and add the following XML just after the existing **<Import>** nodes:

```
<Import Project="DeploymentFiles\BuildSharePointPackage.targets" />
<Target Name="AfterBuild">
  <CallTarget Targets="BuildSharePointPackage" />
</Target>
```

Save your changes, right-click the unloaded project in the **Solution Explorer** tool window & select **Reload Project**. When prompted with a **Security Warning for Lab10** dialog, select **Load project normally**, uncheck **Ask me for every project in this solution** and click **OK**.

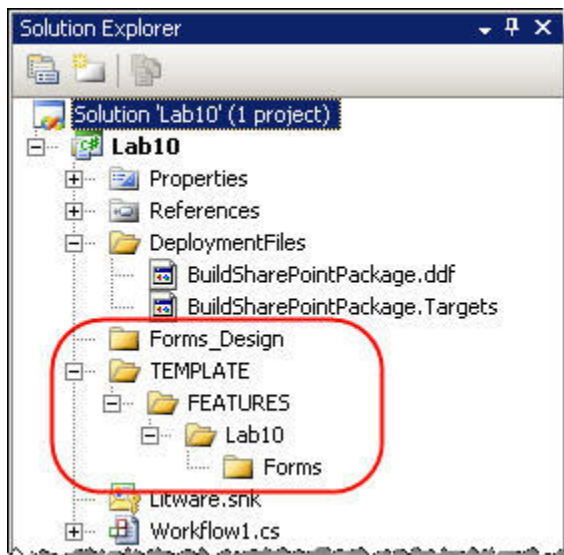
- Next, copy the **Litware.snk** key file into the root of the project from the following folder:

```
c:\Student\Labs\10_Workflow\Resources
```

Now, right-click the **Lab10** project in the **Solution Explorer** window and select **Properties**. Select the **Signing** tab, verify that the **Sign the assembly** checkbox is **checked**, and select **Litware.snk** from the **Choose a strong name key file** selector.

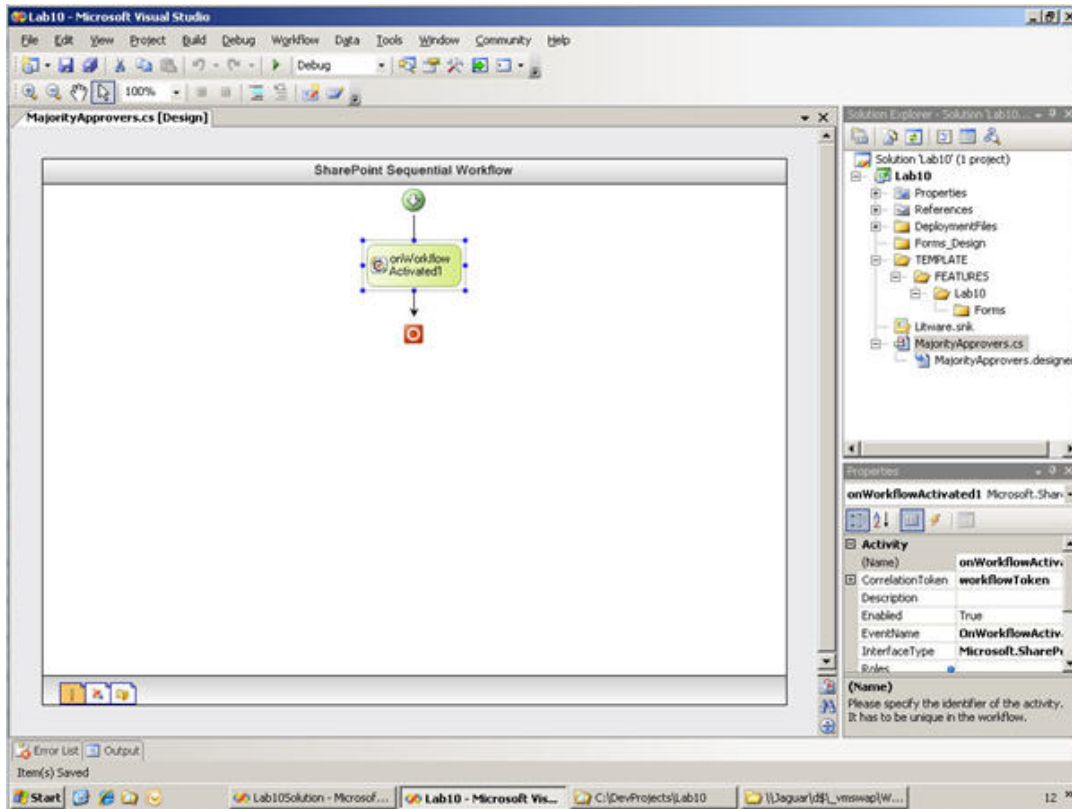
Still from within the project properties page, select the **Build Events** tab and clear the contents of the **Post-build event command line:** textbox.

- Save all changes and close the project property page.
- Finally, create the following folder structure within the **Lab10** project:



- Finally, the last step is to rename the **Workflow1.cs** file to **MajorityApprovers.cs**.
- Now, you need to clean up two issues with this renaming of the workflow refactoring. Double-click **MajorityApprovers.cs** to open the workflow designer. Notice the red circle with an exclamation point in the upper right-hand corner of the only activity on the design surface: **onWorkflowActivated1**. Select the activity and press **F4** to open the **Properties** tool window in case it isn't open already.
- Expand the **CorrelationToken** property and select **MajorityApprovers** in the **OwnerActivityName** property.
- Click the **ellipses** button [...] in the value field of the **WorkflowProperties** property in the **Properties** tool window to show the visual data binder dialog. Select **workflowProperties** on the **Bind to an existing member** tab and click **OK**.

12. There's one more place this needs to be changed. Open the **MajorityApprovers.designer.cs** file and look for the code **activitybind2.Name = "Workflow1"** ... change that to **"MajorityApprovers"**.
13. Save all your changes... your project should now look similar to the following image... notice that the red error indicator on the **onWorkflowActivated1** activity is now gone:



At this point, your project is now ready to start building the workflow! So let's get started...

Exercise 2: Create the workflow process & bind any necessary fields to the activities

In this exercise you will design your workflow process by adding activities to the design surface and connecting them together. You will also create any necessary data bound fields that will be used throughout the project.

If you elected to skip exercise 1, you can copy the contents within the following folder into the Lab folder where you normally do the labs in this course:

c:\Student\Labs\10_Workflow\Resources

1. The first thing you need to do is design your workflow. To do this, you will drag activities onto the design surface. First, double-click **MajorityApprovers.cs** file to open the workflow designer.

2. Next, if it isn't already visible, select **View » Toolbox** to show the toolbox containing all the activities.
3. First, drag a **LogToHistoryListActivity** from the **SharePoint - Workflow** tab in the toolbox and attach it between the **onWorkflowActivated1** & red terminating box.

Using the **Properties** tool window, set the **(name)** property to **logWorkflowStarted**.

4. Next, select the **[...]** button in the **HistoryDescription** property to open the data binding dialog. Because you don't have any fields in your class, you need to create one. Select the **Bind to a new member** tab, enter a name of **HistoryDescription**, select **Create Field** and click **OK**.
5. Repeat the process for the **HistoryOutcome** property, creating a new field named **HistoryOutcome**.
6. Next, drag a **Parallel** activity from the **Windows Workflow** tab in the **Toolbox** and add it just after the **LogToHistoryListActivity** previously added. Change the **name** of the **Parallel** activity from **parallelActivity1** to **createTasksParallel**.
7. At this point your workflow will create three tasks. You will give the three tasks that this workflow keeps track of the unique names of Alpha, Beta & Charlie. But first, notice how only two **Sequence** activities are shown by default in the **Parallel** activity. You need to add another by right-clicking the icon just below the **createTasksParallel** activity name in the designer and selecting **Add Branch**.

Now you can add activities to each branch. In the first branch, drag a **CreateTask** activity into **sequenceActivity1** and change its name to **createAlphaApprovalTask**.

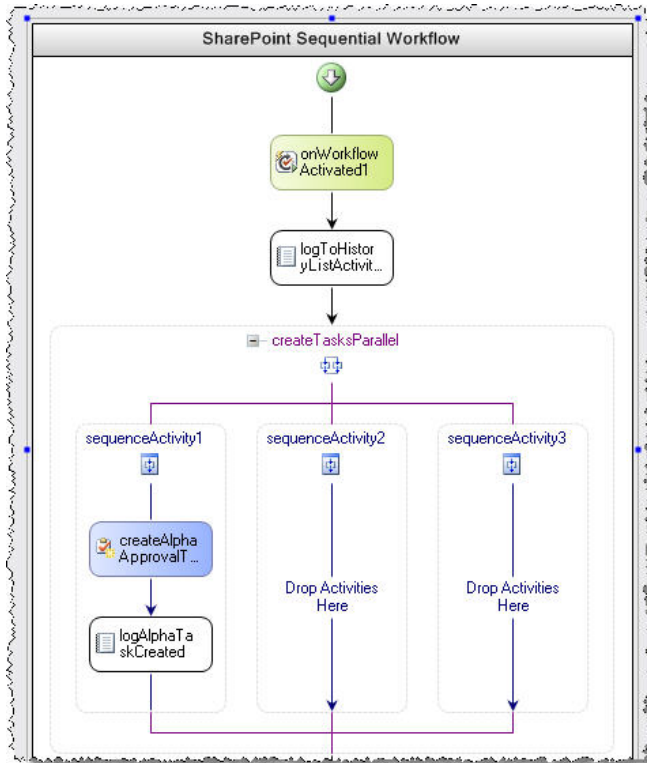
8. In order to keep track of the different tasks and activities in the workflow that will correspond to the tasks, the workflow uses correlation tokens. Enter **alphaTaskToken** in the **CorrelationToken** property for the **createAlphaApprovalTask** activity. Once you click outside of the **CorrelationToken** property, you will notice it suddenly has an expando icon to the left of the name. Click it to select the **OwnerActivityName** to **MajorityApprovers**.
9. For the **TaskId** property of the **createAlphaApprovalTask** activity, click the **[...]** button to add a new field named **AlphaTaskId** using the same process outlined previously in this exercise.

Do the same thing for the **TaskProperties**, creating a new field named **AlphaTaskProperties**.

10. Now, add a **LogToHistoryActivity** just after the **createAlphaApprovalTask** activity and name it **logAlphaTaskCreated**.
11. Next, set the **HistoryDescription** & **HistoryOutcome** properties to the previously created fields **HistoryDescription** & **HistoryOutcome** by selecting them from the **Bind to an existing member** tab when you bring the data binding dialog up.

You may be wondering "Why are we setting the same fields to two different activities?" Don't worry about it for now. We are going to use event handlers on the activities to set the values of these fields before the activity executes.

At this point, your workflow should look like the following image:

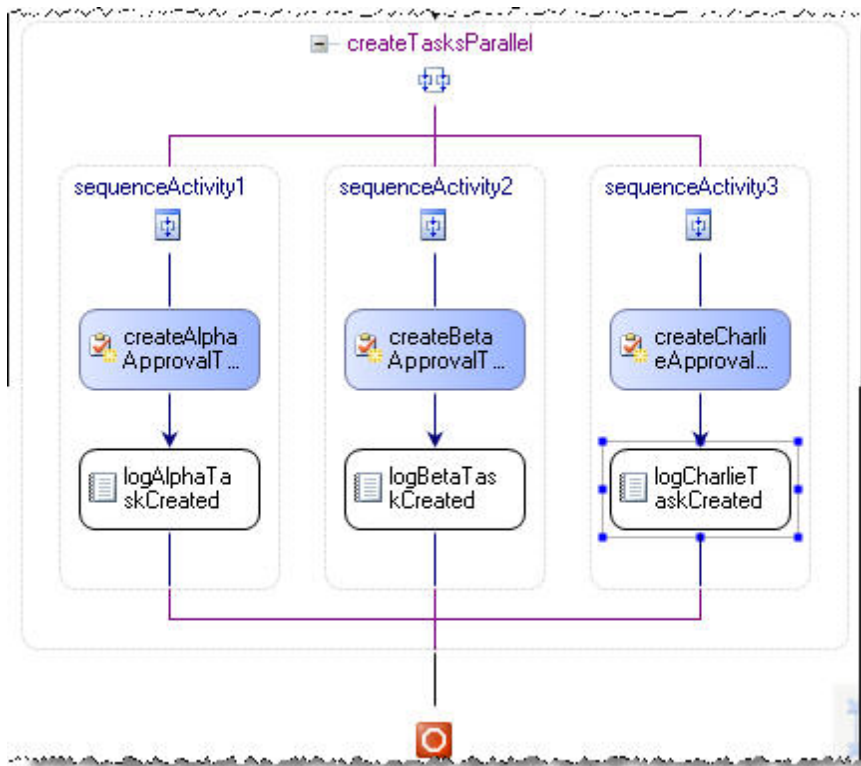


12. Now, repeat the same process for **sequenceActivity2** and **sequenceActivity3**, adding a new **CreateTask** and **LogToHistoryListActivity** and creating all the new necessary fields. The following table should help you sent the values:

sequenceActivity2	
CreateTask	
(Name)	createBetaApprovalTask
CorrelationToken	betaTaskToken
Owner ActivityName	MajorityApprovers
TaskId	BetaTaskId (new field)
TaskProperties	BetaTaskProperties (new field)
LogToHistoryListActivity	
(Name)	logBetaTaskCreated
HistoryDescription	HistoryDescription (existing field)
HistoryOutcome	HistoryOutcome (existing field)

sequenceActivity3	
CreateTask	
(Name)	createCharlieApprovalTask
CorrelationToken	charlieTaskToken
Owner ActivityName	MajorityApprovers
TaskId	CharlieTaskId (<i>new field</i>)
TaskProperties	CharlieTaskProperties (<i>new field</i>)
LogToHistoryListActivity	
(Name)	logCharlieTaskCreated
HistoryDescription	HistoryDescription (<i>existing field</i>)
HistoryOutcome	HistoryOutcome (<i>existing field</i>)

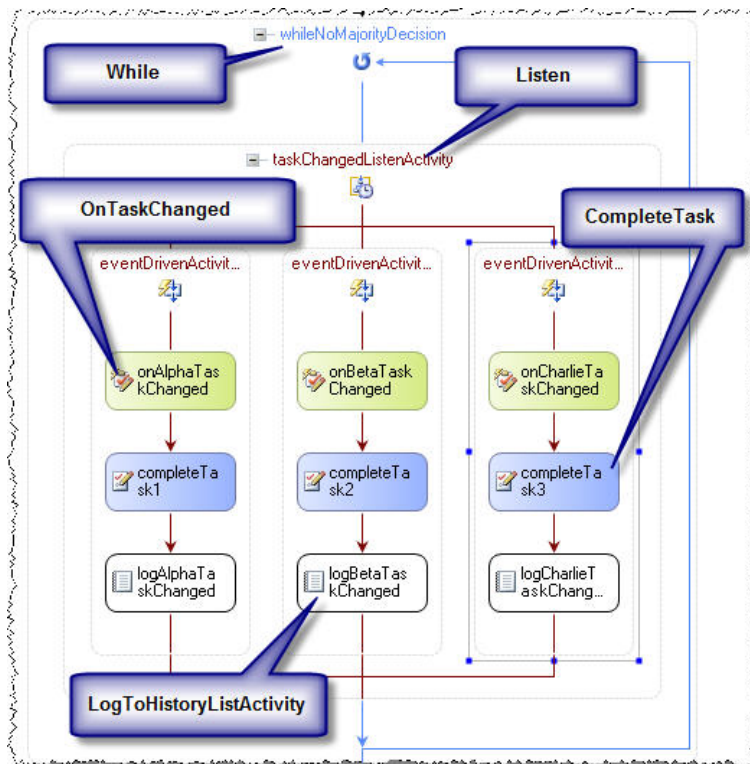
At this point your workflow should look like the following image:



13. With the task creation portion of the workflow complete, it is now time to implement the part that will listen for task changes and wait until a two-thirds (2/3) majority has responded in favor of (approve) or against (reject) publishing the item. To do this, you are going to add quite a few activities. Instead of walking through the process of every single activity, use the following image to guide the design process. The tables after the image detail the properties that need to be set. This structure should be placed just after the **createTasksParallel** activity and before the red termination icon.

Note: You will need to add a new eventDrivenActivity branch to the Listen activity.

Note: Don't worry about the error that will display on the While activity. You are not at the point where you want to add the logic for this activity.



While

(Name)	whileNoMajorityDecision
--------	-------------------------

Listen

(Name)	taskChangedListenActivity
--------	---------------------------

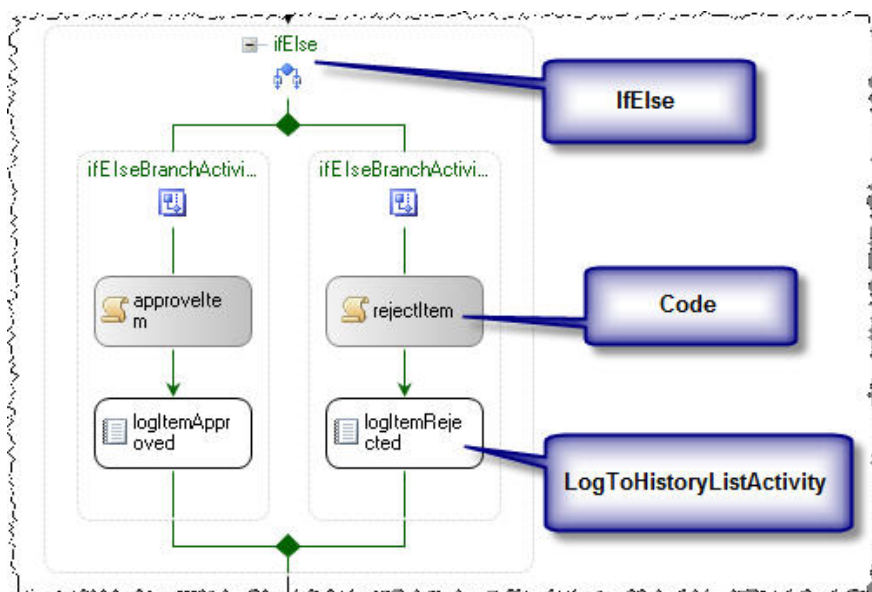
eventDrivenActivity1 (left-hand one)	
OnTaskChanged	
(Name)	onAlphaTaskChanged
CorrelationToken	alphaTaskToken
Owner ActivityName	MajorityApprovers
AfterProperties	AlphaAfterTaskProperties (new field)
TaskId	AlphaTaskId (existing field)
CompleteTask	
(Name)	completeAlphaTask
CorrelationToken	alphaTaskToken
Owner ActivityName	MajorityApprovers
TaskId	AlphaTaskId (existing field)
LogToHistoryListActivity	
(Name)	logAlphaTaskCreated
HistoryDescription	HistoryDescription (existing field)
HistoryOutcome	HistoryOutcome (existing field)

eventDrivenActivity2 (middle one)	
OnTaskChanged	
(Name)	onBetaTaskChanged
CorrelationToken	betaTaskToken
Owner ActivityName	MajorityApprovers
AfterProperties	BetaAfterTaskProperties (new field)
TaskId	BetaTaskId (existing field)
CompleteTask	
(Name)	completeBetaTask
CorrelationToken	betaTaskToken
Owner ActivityName	MajorityApprovers
TaskId	BetaTaskId (existing field)
LogToHistoryListActivity	
(Name)	logBetaTaskCreated
HistoryDescription	HistoryDescription (existing field)
HistoryOutcome	HistoryOutcome (existing field)

eventDrivenActivity3 (right-hand one)	
OnTaskChanged	
(Name)	onCharlieTaskChanged
CorrelationToken	charlieTaskToken
Owner ActivityName	MajorityApprovers
AfterProperties	CharlieAfterTaskProperties (new field)
TaskId	CharlieTaskId (existing field)
CompleteTask	
(Name)	completeCharlieTask
CorrelationToken	charlieTaskToken
Owner ActivityName	MajorityApprovers
TaskId	CharlieTaskId (existing field)
LogToHistoryListActivity	
(Name)	logCharlieTaskCreated
HistoryDescription	HistoryDescription (existing field)
HistoryOutcome	HistoryOutcome (existing field)

14. Next, you need to add the part of the workflow that will do the programmatic approval or rejection of the request. To do this, again, refer to the following image and tables to create the necessary activities. This section should be placed after the previously added While activity and just before the red termination icon.

Note: Again, don't worry about the error that will display on the IfElse activity. You are not at the point where you want to add the logic for this activity.



ifElseBranchActivity (left-hand one)	
Code	
(Name)	approveItem
LogToHistoryListActivity	
(Name)	logItemApproved
HistoryDescription	HistoryDescription (existing field)
HistoryOutcome	HistoryOutcome (existing field)

ifElseBranchActivity (right-hand one)	
Code	
(Name)	rejectItem
LogToHistoryListActivity	
(Name)	logItemRejected
HistoryDescription	HistoryDescription (existing field)
HistoryOutcome	HistoryOutcome (existing field)

15. Finally, add a single LogToHistoryListActivity at the very end of the workflow and set the following properties:

LogToHistoryListActivity	
(Name)	logWorkflowComplete
HistoryDescription	HistoryDescription (existing field)
HistoryOutcome	HistoryOutcome (existing field)

At this point, your workflow is now completely designed. The next step, before working with the code part of the workflow, is to switch over to InfoPath 2007 and create the forms that will be used in this workflow.

Exercise 3: Create & publish InfoPath 2007 forms

In this exercise you will create two new InfoPath 2007 forms that will be used in the initiation and task edit portion of the workflow. With MOSS 2007 installed, you should prefer InfoPath 2007 forms development over building ASPX forms because so much work is done for you by the system. However, with that added benefit comes some tricky pieces with respect to InfoPath forms. Make sure you follow the instructions in this section very closely.

You will build the initialization form first followed by the task edit form.

1. Open **Office InfoPath 2007** and in the **Getting Started** dialog, select **Design a Form Template**.

2. In the **Design a Form Template** dialog, select **Form Template, Based on: Blank** and check **Enable browser-compatible features only**.
3. When the new form loads, select the Task Pane **Data Source** in on the right-hand side of the application. Create a new field by right-clicking **myFields** and selecting **Add....** Use the following table to create the new field:
 - **Name:** alphaApprover
 - **ApproverType:** Field (element)
 - **Data type:** Text (string)
 - **Cannot be blank (*):** checked
4. Repeat the step above creating three more fields named **betaApprover**, **charlieApprover** and **instructions** with the same attributes.
5. Now, rename the top-level node **myFields** to **InitForm**.
6. Next, create a form similar to the one shown in the following image. Don't worry about the button details or large instructions textbox... you will address those in a moment.

A shortcut to get controls on the page that are linked with the fields in the data source is to right-click the root node in the data source and select Controls. InfoPath will automatically drop the controls on the design surface. You can then rearrange them at will.

*Note: You'll find the button control on the **Controls** Task Pane... you can toggle to the Controls Task Pane by clicking the down arrow in the upper right corner of the current Task Pane.*

For an item using this workflow to be approved, it must be approved by a majority of approvers.

Enter three different people who will review the item and approve or decline it:

Approver 1:	LITWAREINC\	
Approver 2:	LITWAREINC\	
Approver 3:	LITWAREINC\	

All approvers must be unique; you can't assign the same person the role of two different approvers.

Instructions to approvers:

Create & Assign Approval Tasks & Start Workflow

7. Let's format the *instructions to approvers* textbox. Right-click the textbox for the instructions and select **Text Box Properties....** Switch to the **Display** tab and check **Multi-line** then click **OK**.
8. Now you need to add some logic to the button. Right-click the button and select **Button Properties....** On the **General** tab, change the **Label:** value to **Create & Assign Approval Tasks & Start Workflow**.
9. Then click the **Rules...** button. On the **Rules** dialog, click **Add**. On the **Rule** dialog, click **Add Action....**
10. On the **Action** dialog, select **Action: Submit using a data connection**, then click **Add....** On the **Data Connection Wizard**, select **Create a new connection to: Submit data** and click **Next >**. Then select **To the hosting environment, such as an ASP.NET page or hosting application** and click **Next >** followed by **Finish**.
11. Now, click **OK** to get back to the **Rule** dialog. Click **Add Action....** On the **Action** dialog, select **Close the form** and **OK** out of all the dialogs.
12. The last thing you need to do is configure the security on the form so it can run within SharePoint. Select **Tools » Form Options**. On the **Category** for **Security and Trust**, uncheck **Automatically determine security level (recommended)**, select **Domain** and click **OK**.

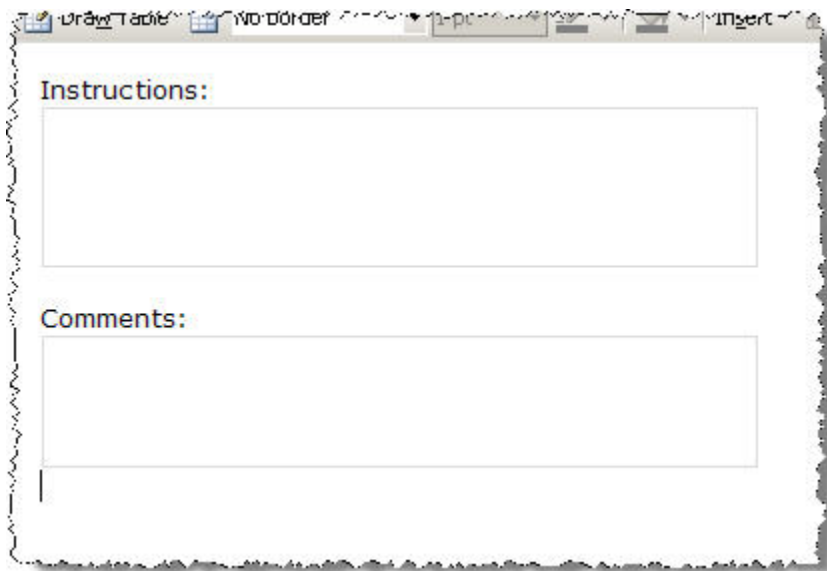
*** It is critical you follow the next two steps explicitly... this is where many errors occur.

13. Now save the form to the **Forms_Design** folder you created in the Visual Studio project containing your workflow giving it a name of **InitForm.xsn**.
14. Next, you need to publish the form. Select **File** then **Publish**. In the **Publishing Wizard**, select **To a network location** and click **Next >**. Browse to the **Forms** folder within the **Feature** folder you created in the Visual Studio project containing your workflow, giving it a name of **InitForm.xsn**. Set the **Form template name: InitForm** and click **Next >**.
15. Clear the **alternate access path** on the next step and click **Next >**. *** *If you don't do this, your form won't work.*

You should get a warning about users not being able to open the form. Ignore this and click **OK**. If you didn't get it, you didn't set the trust level to Domain!
16. Finally, click **Publish** followed by **Close**.
17. To save yourself a step later, go ahead and grab the form's URN before closing it. Select **File » Properties**. Copy the URN located in the **ID** textbox and paste it somewhere for future reference (such as creating a new text file in Notepad)... just make sure you label it with something like **InitForm**). Close the dialog by clicking **OK**.
18. One last thing to do with this form before you continue on... to make your life easier when you get to the code part of this lab, select **File » Save As Source Files...**, create a new directory within **Forms_Design** called **Source Files** and save the files to that location. Then copy the **myschema.xsd** file from the **Source Files** folder to the **Forms_Design** folder and rename it **InitFormSchema.xsd**.

At this point your initialization form is complete. Next you need to create the task form.

1. If you closed InfoPath, open Office InfoPath 2007 and in the **Getting Started** dialog, select **Design a Form Template**.
2. In the **Design a Form Template** dialog, select **Form Template, Based on: Blank** and check **Enable browser-compatible features only**.
3. When the new form loads, select the Task Pane **Data Source** in on the right hand side of the application. Create a new field by right-clicking **myFields** and selecting **Add...** Use the following table to create the new field:
 - **Name:** instructions
 - **Type:** Field (element)
 - **Data type:** Text (string)
 - **Cannot be blank (*):** checked
4. Repeat the step above creating three more fields named **comments** and **decision** with the same attributes (except don't make comments required).
5. Now, rename the top-level node **myFields** to **TaskForm**.
6. Next, create a form similar to the one shown in the following image. Make both the instructions & comments textboxes multiline and larger input fields:



7. Unlike the initialization form you created previously, the task form needs some extra work in order for SharePoint to pass data back and forth to the form. This is because the form can't know about any changes to the task list that may occur after deployment. The way you do this is with an XML file named **ItemMetadata.xml**. Switch over to Visual Studio for a moment and create a new XML file named **ItemMetadata.xml** at the root of project and add the following XML markup to the file:

```
<?xml version="1.0" encoding="utf-8" ?>
<z:row xmlns:z="#RowsetSchema"
  ows_instructions="" />
```

8. Notice the field is prefixed with **ows_**. This is the field that is used in the form that needs to be passed into the form. Now that you have created the ItemMetadata.xml file, you need to add the XML file to the task form as a new data source. Jump back to InfoPath and select **Tools » Data Connections....** From the **Data Connections** dialog, click **Add**.

When the wizard loads, select **Create a new connection to: Receive data** and click **Next >**.

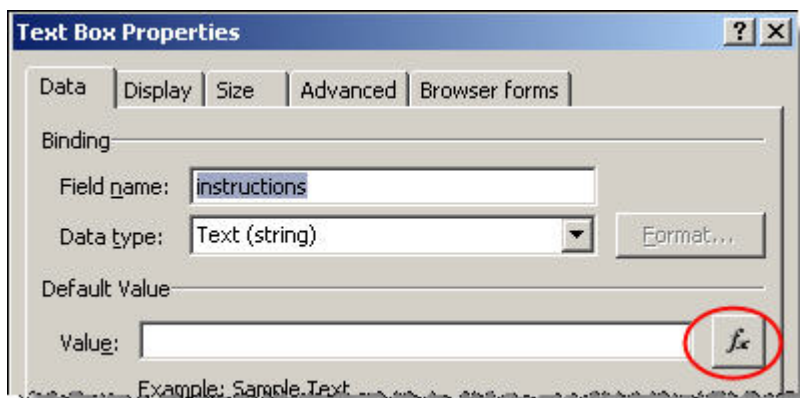
Specify you want to **receive data from an XML document** and select **Next >**.

Browse to the location of the **ItemMetadata.xml** file. Then click **Resource Files...**, followed by **Add...** and select the same file.

OK out of the dialog back to the wizard and click **Next >**.

Accept the defaults on the next screen in the wizard (**Name =ItemMetadata & Automatically receive data when form is opened** is checked) and click **Finish**. Close the **Data Connections** dialog.

9. With a data source created, you now need to configure your form to pull the instructions from the XML file and insert it into the field. Right-click the **instructions** textbox and select **Properties....**
10. Click the **function** button to the right of the **Value** textbox in the **Default Value** section.



11. Click the **Insert Field or Group...** button change the **Data source:** to **ItemMetadata (Secondary)** and select **:ows_instructions** and **OK** out of the function dialog.
12. On the **Display** tab, check the **Read-only** checkbox as we don't really want approvers changing their instructions and **OK** out of the dialog.
13. The last design thing you need to do to the form is you need to add two buttons to either approve or reject the task. Add two buttons at the bottom of the form.
14. Right-click the first button and select **Button Properties....** Set the **Label:** to **Approve**.
Click the **Rules...** button to bring up the **Rules** dialog. Click the **Add...** button and then the **Add Action...** button.

Select the **Action:** to **Set a field's value**, pick the decision field from the **Main** data source and click **OK**. In the **Value** textbox, type **approve** and click **OK**.

Click **Add Action....** On the **Action** dialog, select **Action: Submit using a data connection**, then click **Add....** On the **Data Connection Wizard**, select **Create a new connection to: Submit data** and click **Next >**. Then select **To the hosting environment, such as an ASP.NET page or hosting application** and click **Next >** followed by **Finish**.

Now, click **OK** to get back to the **Rule** dialog. Click **Add Action....** On the **Action** dialog, select **Close the form** and **OK** out of all the dialogs.

15. Do the same thing to the other button, except name it **Reject**, set the value of the **decision** field to **reject**, and use the existing data connection you created in the last step to submit the form to SharePoint.
16. The last thing you need to do is configure the security on the form so it can run within SharePoint. Select **Tools » Form Options**. On the **Category for Security and Trust**, uncheck **Automatically determine security level (recommended)**, select **Domain** and click **OK**.

*** *It is critical you follow the next two steps explicitly... this is where many errors occur.*

17. Now save the form to the **Forms_Design** folder you created in the Visual Studio project containing your workflow giving it a name of **TaskForm.xsn**.
18. Next, you need to publish the form. Select **File » Publish**. In the **Publishing Wizard**, select **To a network location** and click **Next >**. Browse to the **Forms** folder within the **Feature** folder you created in the Visual Studio project containing your workflow, giving it a name of **TaskForm.xsn**. Set the **Form template name: TaskForm** and click **Next >**.

Clear the **alternate access path** on the next step and click **Next >**. *** If you don't do this, your form won't work.

19. You should get a warning about users not being able to open the form. Ignore this and click **OK**. *If you didn't get it, you didn't set the trust level to Domain!*

Finally, click **Publish** followed by **Close**.

20. To save yourself a step later, go ahead and grab the form's URN before closing it. Select **File » Properties**. Copy the URN located in the **ID** textbox and paste it somewhere for future reference (such as creating a new text file in Notepad)... just make sure you label it with something like TaskForm).

Close the dialog by clicking **OK**.

At this point you have created the two forms that will be leveraged by your workflow and published them to the Forms folder within the Feature.

Exercise 4: Add code to integrate the InfoPath forms and necessary logic

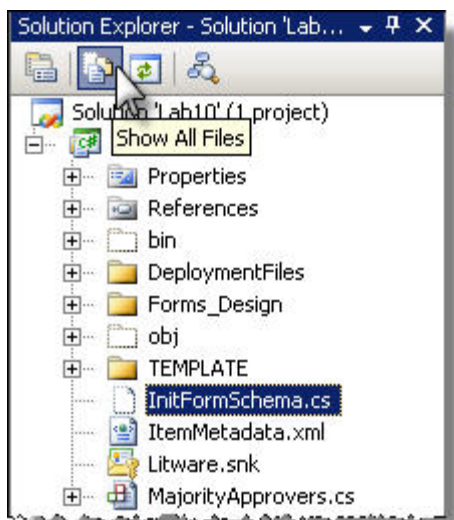
In this exercise you will integrate the InfoPath forms into the workflow as well as add all necessary logic to the code behind of the workflow. Hope you like writing code!

1. Switch back to Visual Studio where the MajorityApprovers workflow should still be open. You will now start adding code to each of the event handlers on all activities that add some logic to the workflow. You will work through the workflow from the top to the bottom. While there are quite a few class-scoped fields to create, you will add them only when necessary.
2. Before you start adding code, do one thing that was alluded to in the last exercise that would save you some headache. What you will do is create a new C# class from the InitFormSchema.xsd file you created when creating the IntiForm InfoPath form. This will save you from having to create a namespace and query the XML passed from SharePoint, rather you can just deserialize the data into the class.

Open a **Visual Studio Command Prompt**, change directory into the root of your workflow project and enter the following command:

```
xsd.exe [path to init form schema file]/InitFormSchema.xsd /c
```

3. Now, go back to Visual Studio and click the icon to **Show all files** icon, right click the **InitFormSchema.cs** file and select **Include In Project**.



4. Open the **InitFormSchema.cs** file. You'll notice the class, **InitForm**, is not in a namespace. Add the following code just after the using statement to add it to the namespace:

```
namespace Lab10  
{
```

Finally, add a final closing **}** to the end of the file to close the namespace. Save your changes to the **InfoFormSchema.cs** file.

5. Right-click the **onWorkflowActivated** activity at the top of the workflow and select **Generate Handlers**. Within this method you will add code that will deserialize the data from the initialization form passed by SharePoint to the workflow in the **workflowProperties.InitiationData** property. This makes it much easier to work

with the data as you don't have to write any XML XPath queries to fetch the data from the XML.

6. However, first you need to create a few class-scoped private fields. Add the following just before the method what was generated by Visual Studio:

```
private string _instructions = default(string);
private string _alphaApprover = default(string);
private string _betaApprover = default(string);
private string _charlieApprover = default(string);
```

7. Now, add the following code to the generated handler:

```
private void onWorkflowActivated_Invoked (object sender, ExternalDataEventArgs e) {
    // load the data from the IP form 'InitForm' into a local object
    XmlSerializer serializer = new XmlSerializer(typeof(InitForm));
    XmlTextReader xrInitForm = new XmlTextReader(new
    System.IO.StringReader(workflowProperties.InitiationData));
    InitForm frmInit = serializer.Deserialize(xrInitForm) as InitForm;

    // get approvers submitted
    this._alphaApprover = @"LITWAREINC\" + frmInit.alphaApprover;
    this.EnsureUserIsRecognized(this._alphaApprover);
    this._betaApprover = @"LITWAREINC\" + frmInit.betaApprover;
    this.EnsureUserIsRecognized(this._betaApprover);
    this._charlieApprover = @"LITWAREINC\" + frmInit.charlieApprover;
    this.EnsureUserIsRecognized(this._charlieApprover);

    // get instructions
    this._instructions = frmInit.instructions;
}
```

8. You may notice a method called **EnsureUserIsRecognized()** in the code above that hasn't been defined. What this method will do is check to see if the provided user has rights into the current SharePoint site where the workflow is running. If not, the user is automatically granted access to the site. This is done using the two following methods which you need to add to the **MajorityApprovers.cs** file:

```
private void EnsureUserIsRecognized (string userName) {
    // if the user isn't found in the site, add them
    if (this.GetUserIdFromUserName(workflowProperties.Web, userName) == -1)
        workflowProperties.Web.SiteUsers.Add(userName, "", "", "");
}

private int GetUserIdFromUserName (SPWeb site, string userName) {
    // get the user's info
    Microsoft.SharePoint.Utilities.SPPrincipalInfo principalInfo =
        Microsoft.SharePoint.Utilities.SPUtility.ResolvePrincipal(site,
            userName,
            SPPrincipalType.All,
            SPPrincipalSource.All,
            null,
            false);

    // if the user was found, return their ID, else -1
    if (principalInfo != null)
        return principalInfo.PrincipalId;
    else
        return -1;
}
```

9. Now you need to log a message indicating the workflow started. Right-click the **logToHistoryListActivity** at the top of the workflow and select **Generate Handlers**. Add the following code that will report the data collected from the initialization form and save it to the two local fields created when you created this activity:

```
private void logWorkflowStarted_MethodInvoking (object sender, EventArgs e) {
    // get reference to the item this workflow is associated with
    SListItem page = workflowProperties.Item;

    // create log message & outcome
    this.HistoryOutcome = "Workflow successfully initiated...";
    this.HistoryDescription = String.Format("Collected three approvers: {0}, {1} & {2}. All
received the following instructions: {3}",
        this._alphaApprover,
        this._betaApprover,
        this._charlieApprover,
        this._instructions);
}
```

10. The next step is to create the event handlers for the task creation activities and each associated log activity. Right-click the two activities in **sequenceActivity1** within the **createTasksParallel** parallel activity (**createAlphaApprovalTask** & **logAlphaTaskCreated**) and add the following code to each:

```
private void createAlphaApprovalTask_MethodInvoking (object sender, EventArgs e) {
    this.AlphaTaskId = Guid.NewGuid();

    this.AlphaTaskProperties.Title = "Approval requested for " +
workflowProperties.Item.Title;
    this.AlphaTaskProperties.Description = "Please review the item, then approve or reject
it.";
    this.AlphaTaskProperties.AssignedTo = this._alphaApprover;
    this.AlphaTaskProperties.PercentComplete = 0;
    this.AlphaTaskProperties.StartDate = DateTime.Today;
    this.AlphaTaskProperties.DueDate = DateTime.Today.AddDays(7);
    this.AlphaTaskProperties.ExtendedProperties["instructions"] = this._instructions;
}

private void logAlphaTaskCreated_MethodInvoking (object sender, EventArgs e) {
    this.HistoryOutcome = String.Format("Task created and assigned to first approver: {0}",
this._alphaApprover);
    this.HistoryDescription = String.Format("Approval task 'alpha' created and assigned to
{0}", this._alphaApprover);
}
```

11. Repeat the previous step but for the activities within **sequenceActivity2** within the **createTasksParallel** parallel activity (**createBetaApprovalTask** & **logBetaTaskCreated**), using the following code:

```
private void createBetaApprovalTask_MethodInvoking (object sender, EventArgs e) {
    this.BetaTaskId = Guid.NewGuid();

    this.BetaTaskProperties.Title = "Approval requested for " +
workflowProperties.Item.Title;
    this.BetaTaskProperties.Description = "Please review the item, then approve or reject
it.";
    this.BetaTaskProperties.AssignedTo = this._betaApprover;
    this.BetaTaskProperties.PercentComplete = 0;
    this.BetaTaskProperties.StartDate = DateTime.Today;
    this.BetaTaskProperties.DueDate = DateTime.Today.AddDays(7);
}
```

```

        this.BetaTaskProperties.ExtendedProperties["instructions"] = this._instructions;
    }

    private void logBetaTaskCreated_MethodInvoking (object sender, EventArgs e) {
        this.HistoryOutcome = String.Format("Task created and assigned to second approver: {0}", this._betaApprover);
        this.HistoryDescription = String.Format("Approval task 'beta' created and assigned to {0}", this._betaApprover);
    }
}

```

12. Repeat the previous step but for the activities within **sequenceActivity3** within the **createTasksParallel** parallel activity (**createCharlieApprovalTask** & **logCharlieTaskCreated**), using the following code:

```

private void createCharlieApprovalTask_MethodInvoking (object sender, EventArgs e) {
    this.CharlieTaskId = Guid.NewGuid();

    this.CharlieTaskProperties.Title = "Approval requested for " +
    workflowProperties.Item.Title;
    this.CharlieTaskProperties.Description = "Please review the item, then approve or reject it.";
    this.CharlieTaskProperties.AssignedTo = this._charlieApprover;
    this.CharlieTaskProperties.PercentComplete = 0;
    this.CharlieTaskProperties.StartDate = DateTime.Today;
    this.CharlieTaskProperties.DueDate = DateTime.Today.AddDays(7);
    this.CharlieTaskProperties.ExtendedProperties["instructions"] = this._instructions;
}

private void logCharlieTaskCreated_MethodInvoking (object sender, EventArgs e) {
    this.HistoryOutcome = String.Format("Task created and assigned to third approver: {0}", this._charlieApprover);
    this.HistoryDescription = String.Format("Approval task 'charlie' created and assigned to {0}", this._charlieApprover);
}

```

With all the tasks created you can now move onto the section where the workflow will go to sleep waiting for some activity on the tasks. However, first there are a few things you need to create before moving on.

13. First, you are going to need an enumeration called **ApproverDecision** that will be used instead of strings (much safer!). Create a new **C#** class file named **ApproverDecision.cs** and add the following code to it:

```

using System;

namespace Lab10
{
    public enum ApprovalDecision
    {
        Approved,
        Rejected,
        NoAnswer
    }
}

```

14. Second, there are a few new class-scoped fields that are needed. Some will keep track of the different tasks answers, others keep track of how many approvals & rejections there have been, and one is used to determine if the majority of approvers have responded one way or another. Add the following field declarations to the **MajorityApprovers** class:

```
public ApprovalDecision AlphaTaskAnswer = ApprovalDecision.NoAnswer;
public ApprovalDecision BetaTaskAnswer = ApprovalDecision.NoAnswer;
public ApprovalDecision CharlieTaskAnswer = ApprovalDecision.NoAnswer;
public int ApproveAnswerCount = 0;
public int RejectAnswerCount = 0;
public bool MajorityHasAnswered = false;
```

15. Third, when you add code to the event handlers for the **OnTaskChanged** activities within the while activity, you are going to add a call to a method called **UpdateMajorityDecision**. This method is called to total up the approve/reject score into a single value that will be used to determine if the while loop should continue or not. Add the following method to the **MajorityApprovers.cs** file:

```
private void UpdateMajorityDecision () {
    int countApproved = 0;
    int countRejected = 0;

    // check alpha approver
    if (this.AlphaTaskAnswer == ApprovalDecision.Approved)
        countApproved++;
    else if (this.AlphaTaskAnswer == ApprovalDecision.Rejected)
        countRejected++;

    // check beta approver
    if (this.BetaTaskAnswer == ApprovalDecision.Approved)
        countApproved++;
    else if (this.BetaTaskAnswer == ApprovalDecision.Rejected)
        countRejected++;

    // check charlie approver
    if (this.CharlieTaskAnswer == ApprovalDecision.Approved)
        countApproved++;
    else if (this.CharlieTaskAnswer == ApprovalDecision.Rejected)
        countRejected++;

    // set results
    this.ApproveAnswerCount = countApproved;
    this.RejectAnswerCount = countRejected;

    // if three respondants, no need to check futher
    if (countApproved + countRejected == 3) {
        this.MajorityHasAnswered = true;
    } // else if two of EITHER answer have responded, majority rules
    else if (countApproved == 2 || countRejected == 2)
        this.MajorityHasAnswered = true;
}
```

Now you can to work on creating the event handlers that are waiting for the tasks to update...

16. There is some work to do on the **whileNoMajorityDecision** while activity, but you will address that at the end. For now, just generate handlers for the two of the three activities in the eventDrivenActivity1 (**onAlphaTaskChanged** & **logAlphaTaskChanged**... no need to create a handler for the **completeAlphaTask** activity) and add the following code to the created handlers:

```
private void onAlphaTaskChanged_Invoked (object sender, ExternalDataEventArgs e) {
    // check if the task was approved
    string taskResult =
    this.AlphaTaskAfterProperties.ExtendedProperties["decision"].ToString();
    if (taskResult.ToLower() == "approve")
```

```

        this.AlphaTaskAnswer = ApprovalDecision.Approved;
    else if (taskResult.ToLower() == "reject")
        this.AlphaTaskAnswer = ApprovalDecision.Rejected;

    // now, need to check all the other tasks to see 2/3 are approved
    // b/c if so, need to mark whole thing as complete
    UpdateMajorityDecision();
}

private void logAlphaTaskChanged_MethodInvoking (object sender, EventArgs e) {
    this.HistoryOutcome = ("First approver task answered.");
    this.HistoryDescription = String.Format("Approval task 'alpha' status is now: {0}.
Current score: {1}-{2}. Approver entered the following comments: '{3}'.",
        this.AlphaTaskAnswer.ToString(),
        this.ApproveAnswerCount.ToString(),
        this.RejectAnswerCount.ToString(),
        this.AlphaTaskAfterProperties.ExtendedProperties["comments"].ToString());
}

```

17. Repeat the previous step but for the activities within **evenDrivenActivity2** within the **taskChangedListenActivity** listen activity (**onBetaTaskChanged** & **logBetaTaskChanged**), using the following code:

```

private void onBetaTaskChanged_Invoked (object sender, ExternalDataEventArgs e) {
    // check if the task was approved
    string taskResult =
this.BetaTaskAfterProperties.ExtendedProperties["decision"].ToString();
    if (taskResult.ToLower() == "approve")
        this.BetaTaskAnswer = ApprovalDecision.Approved;
    else if (taskResult.ToLower() == "reject")
        this.BetaTaskAnswer = ApprovalDecision.Rejected;

    // now, need to check all the other tasks to see 2/3 are approved
    // b/c if so, need to mark whole thing as complete
    UpdateMajorityDecision();
}

private void logBetaTaskChanged_MethodInvoking (object sender, EventArgs e) {
    this.HistoryOutcome = ("Second approver task answered.");
    this.HistoryDescription = String.Format("Approval task 'beta' status is now: {0}.
Current score: {1}-{2}. Approver entered the following comments: '{3}'.",
        this.BetaTaskAnswer.ToString(),
        this.ApproveAnswerCount.ToString(),
        this.RejectAnswerCount.ToString(),
        this.BetaTaskAfterProperties.ExtendedProperties["comments"].ToString());
}

```

18. Repeat the previous step but for the activities within **evenDrivenActivity3** within the **taskChangedListenActivity** listen activity (**onCharlieTaskChanged** & **logCharlieTaskChanged**), using the following code:

```

private void onCharlieTaskChanged_Invoked (object sender, ExternalDataEventArgs e) {
    // check if the task was approved
    string taskResult =
this.CharlieTaskAfterProperties.ExtendedProperties["decision"].ToString();
    if (taskResult.ToLower() == "approve")
        this.CharlieTaskAnswer = ApprovalDecision.Approved;
    else if (taskResult.ToLower() == "reject")
        this.CharlieTaskAnswer = ApprovalDecision.Rejected;

    // now, need to check all the other tasks to see 2/3 are approved
    // b/c if so, need to mark whole thing as complete

```

```

        UpdateMajorityDecision();
    }

    private void logCharlieTaskChanged_MethodInvoking (object sender, EventArgs e) {
        this.HistoryOutcome = ("Third approver task answered.");
        this.HistoryDescription = String.Format("Approval task 'charlie' status is now: {0}. Current score: {1}-{2}. Approver entered the following comments: '{3}'.",
            this.CharlieTaskAnswer.ToString(),
            this.ApproveAnswerCount.ToString(),
            this.RejectAnswerCount.ToString(),
            this.CharlieTaskAfterProperties.ExtendedProperties["comments"].ToString());
    }

```

19. Before you move on, let's go back and configure the while activity's condition for when it will terminate. You want the while loop to continue until a majority has responded one way or another. For example, you want it to stop when two approve or reject the task, but if the score is 1-1 after two responses, it waits for the third response.

Select the **whileNoMajorityDecision** while activity and take a look at the **Properties** tool window. Notice there is an error for the **Condition** property. Change the **Condition** to a **Declarative Rule Condition**, click the expando icon and then click the ellipses [...] button in the **ConditionName** property.

Create a new condition by clicking **New....** Then, in the **Rule Condition Editor**, enter the following code and click **OK**:

```
!this.MajorityHasAnswered
```

After the condition has been created, rename it from **Condition1** to **Majority has not answered**. Select it and click **OK**.

20. With the workflow tasks answered and a result determined, you now need to configure the code activities that will either approve or reject the item that kicked off the workflow.

Select the **ifElseBranchActivity1**, the one on the left that contains the **approveItem** code activity. Change the **Condition to a Declarative Rule Condition**, click the expando icon and then click the ellipses [...] button in the **ConditionName** property.

Create a new condition by clicking **New....** Then enter the following code in the **Rule Condition Editor** and click **OK**:

```
this.ApproveAnswerCount >=2
```

After the condition has been created, rename it from **Condition1** to **Majority approved**.

21. Right-click the **approveItem** code activity, select **Generate Handlers** and add the following code to the handler:

```

private void approveItem_ExecuteCode (object sender, EventArgs e) {
    // get the item...
    SPlistItem item = workflowProperties.Item;

    // approve the item
    item.File.Approve("Approved by a majority.");
}

```

```
}
```

22. Right-click the **logItemApproved** activity, select **Generate Handlers** and add the following code to the handler:

```
private void logItemApproved_MethodInvoking (object sender, EventArgs e) {
    this.HistoryOutcome = "Item approved by a majority.";
    this.HistoryDescription = String.Format("The final score was {0} approvers & {1} rejecters. Once a majority of three approvers is reached, the workflow terminates.",
        this.ApproveAnswerCount.ToString(),
        this.RejectAnswerCount.ToString());
}
```

23. Select the **ifElseBranchActivity2**, the one on the right that contains the **rejectItem** code activity. Change the **Condition** to a **Declarative Rule Condition**, click the expando icon and then click the ellipses [...] button in the **ConditionName** property.

Create a new condition by clicking **New....** Then enter the following code in the **Rule Condition Editor** and click **OK**:

```
this.RejectAnswerCount >=2
```

After the condition has been created, rename it from **Condition1** to **Majority rejected**.

24. Right-click the **rejectItem** code activity, select **Generate Handlers** and add the following code to the handler:

```
private void rejectItem_ExecuteCode (object sender, EventArgs e) {
    SPListItem item = workflowProperties.Item;
    item.File.Deny("Rejected by a majority.");
}
```

25. Right-click the **logItemRejected** activity, select **Generate Handlers** and add the following code to the handler:

```
private void logItemRejected_MethodInvoking (object sender, EventArgs e) {
    this.HistoryOutcome = "Item rejected by a majority.";
    this.HistoryDescription = String.Format("The final score was {0} approvers & {1} rejecters. Once a majority of three approvers is reached, the workflow terminates.",
        this.ApproveAnswerCount.ToString(),
        this.RejectAnswerCount.ToString());
}
```

Finally... right-click the **logWorkflowComplete** activity, select the **Generate Handlers** and add the following code to the handler:

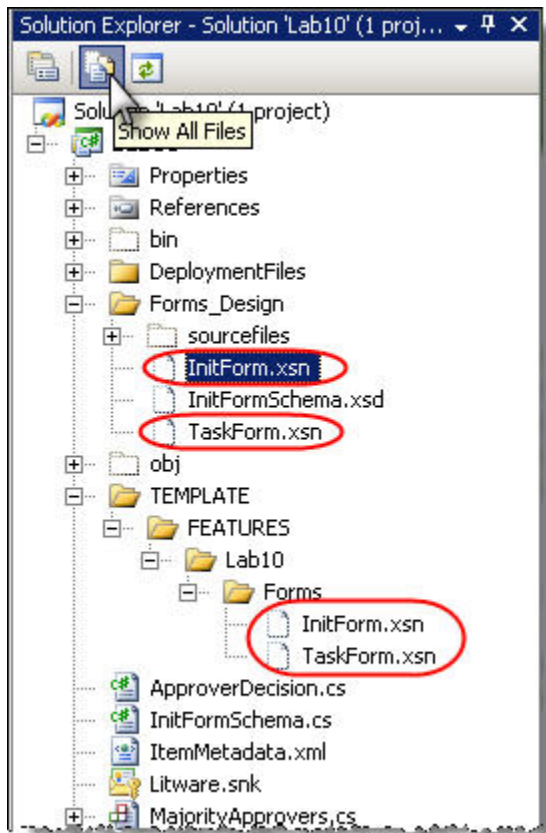
```
private void logWorkflowComplete_MethodInvoking (object sender, EventArgs e) {
    this.HistoryOutcome = "Workflow complete.";
    this.HistoryDescription = "Workflow finished... majority always wins (except in hand grenades & horseshoes).";
}
```

That's it! You made it through all that code!!!! In this exercise, you added logic and integrated the InfoPath forms into the workflow.

Exercise 5: Package the workflow up for deployment

In this exercise you will configure the Feature that will be used to install the workflow as well as the WSS solution package that will be used for deploying the Feature and associated files.

1. There are a few files that you've added to the folder structure of the workflow project in Visual Studio, but you've yet to add them to the project itself... let's take care of that now. If hidden items aren't being shown currently, click the **Show all files** icon at the top of the **Solution Explorer** tool window. Take note of two InfoPath published form files you need to add to the project (within the **Feature** folder) and the two InfoPath design form files saved within the **Forms_Design** folder. Right-click each of these *.XSN files and select **Include In Project**:



2. Now, the first thing you need to do is create the necessary Feature files. First, create a new XML file named **feature.xml** in the **Lab10** folder in the project and add the following XML markup:

```
<?xml version="1.0" encoding="utf-8"?>
<Feature xmlns="http://schemas.microsoft.com/sharepoint/"
  Id="39513EA7-A8A3-4D92-9028-14DF5690B532"
  Title="Lab 10 - Majority Rules Workflow for Publishing Sites"
  Description="Deploys a workflow template that will assign one task to three people, and
  then wait for a majority to approve/reject before completing the workflow."
  Scope="Site"
  Hidden="False"
  ReceiverAssembly="Microsoft.Office.Workflow.Feature, Version=12.0.0.0, Culture=neutral,
  PublicKeyToken=71e9bce111e9429c"
  ReceiverClass="Microsoft.Office.Workflow.Feature.WorkflowFeatureReceiver"
```

```

Version="1.0.0.0">

<ElementManifests>
  <ElementManifest Location="workflow.xml" />
  <ElementFile Location="Forms\InitForm.xsn" />
  <ElementFile Location="Forms\TaskForm.xsn" />
</ElementManifests>

<Properties>
  <Property Key="GloballyAvailable" Value="true" />
  <Property Key="RegisterForms" Value="Forms\*.xsn" />
</Properties>
</Feature>

```

Parts of this file deserve a bit of explanation:

- **<Feature Site="">**: all Features that deploy workflow templates to SharePoint must be scoped at the site collection level.
- **<Feature ReceiverAssembly="" & ReceiverClass="">**: this is boilerplate code... when your workflow leverages InfoPath forms, you need to use this Feature receiver as it will do the work of uploading the InfoPath forms to SharePoint.
- **<Property Key="GloballyAvailable">**: this property tells the Feature receiver that the InfoPath forms should be uploaded to Central Administration for all sites in the farm. This is so the same workflow added to site collections across your organization doesn't create hundreds if not thousands of instances of your forms.
- **<Property Key="RegisterForms">**: this property tells the Feature receiver which InfoPath forms to upload & register as well as where it will find these forms.

3. Now, create a new XML file named **workflow.xml** in the **Lab10** folder in the project and add the following XML markup:

```

<?xml version="1.0" encoding="utf-8" ?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">

  <Workflow Name="Majority Approvers"
    Description="Assigns three different people approval tasks. When a majority
    approves/rejects their tasks, the workflow approves/rejects the item."
    Id="D5613F31-FA46-4E45-A264-BDB5CC8061EC"
    CodeBesideAssembly="Lab10, Version=1.0.0.0, Culture=neutral,
    PublicKeyToken=d4e5777b16a5749f"
    CodeBesideClass="Lab10.MajorityApprovers"
    TaskListContentTypeId="0x01080100C9C9515DE4E24001905074F980F93160"
    AssociationUrl="_layouts/CstWrkflIP.aspx"
    InstantiationUrl="_layouts/IniWrkflIP.aspx"
    ModificationUrl="_layouts/ModWrkflIP.aspx"
    StatusUrl="_layouts/WrkStat.aspx">

    <Categories />

    <MetaData>
      <Instantiation_FormURN>[[ your form's URN ]]</Instantiation_FormURN>
      <Task0_FormURN>[[ your form's URN ]]</Task0_FormURN>
    </MetaData>
  </Workflow>
</Elements>

```

```
</Workflow>
</Elements>
```

Parts of this file deserve a bit of explanation:

- **<Workflow Name="" & Description="">**: these values are seen by site owners/admins who associate this workflow template with a list or content type.
 - **<Workflow CodeBesideAssembly="" & CodeBesideClass="">**: these tell SharePoint where the class that contains the workflow, and then assembly containing the class, can be found. Note: the assembly will be added to the GAC which you'll see in a moment.
 - **<Workflow TaskListContentTypeId="">**: your workflow used the OOTB task list so this is the content type ID for the task content type, but if you created a custom content type used by your workflow, you would specify its content type ID here.
 - **<Workflow [Association|Instantiation|Modification|Status]Url="">**: these attributes tell SharePoint what pages should be used for the different forms; the nice thing here is this is typically boilerplate code as each of these provided pages contains a Web Part which will load the appropriate InfoPath form based on what has been defined in the <MetaData> section.
 - **<MetaData>**: this is where you need to specify the InfoPath form's URN that you copied to a text file; put your form's URN's in the appropriate XML node.
4. With the Feature created, you now need to package everything up into a WSS solution package. Open the **BuildSharePointPackage.ddf** file and add the following code between the comments:

```
DeploymentFiles\manifest.xml

bin\debug\Lab10.dll

.Set DestinationDir=Lab10
TEMPLATE\FEATURES\Lab10\feature.xml
TEMPLATE\FEATURES\Lab10\workflow.xml

.Set DestinationDir=Lab10\Forms
TEMPLATE\FEATURES\Lab10\Forms\InitForm.xsn
TEMPLATE\FEATURES\Lab10\Forms\TaskForm.xsn
```

5. Finally, the last thing to do is to create the WSS solution package manifest. Add a new XML file named **manifest.xml** to the **DeploymentFiles** folder in the project and add the following XML code to the file:

```
<?xml version="1.0" encoding="utf-8" ?>
<Solution xmlns="http://schemas.microsoft.com/sharepoint/"
  SolutionId="85EF6F81-4B4A-44EB-A71E-12605A24B7D9"
  DeploymentServerType="WebFrontEnd"
  ResetWebServer="FALSE">

  <Assemblies>
    <Assembly DeploymentTarget="GlobalAssemblyCache" Location="Lab10.dll" />
  </Assemblies>
```

```
<FeatureManifests>
  <FeatureManifest Location="Lab10\feature.xml" />
</FeatureManifests>
</Solution>
```

6. Build the project to create the WSP file... now it's time to deploy and see this guy in action!

In this exercise you finished creating the Feature that is used to deploy the workflow and custom InfoPath 2007 forms to SharePoint as well as the final configuration tasks necessary to package everything up in a WSS solution package.

Exercise 6: Implement the workflow

At last... we can finally see if this thing works! In this exercise you will install the WSS solution package, deploy the workflow to a site collection, and associate it with an existing Publishing site's Pages list.

1. First the WSS solution package must be deployed. Open a command prompt and navigate to the following directory:

```
c:\Program Files\Common Files\Microsoft Shared\web server extensions\12\BIN
```

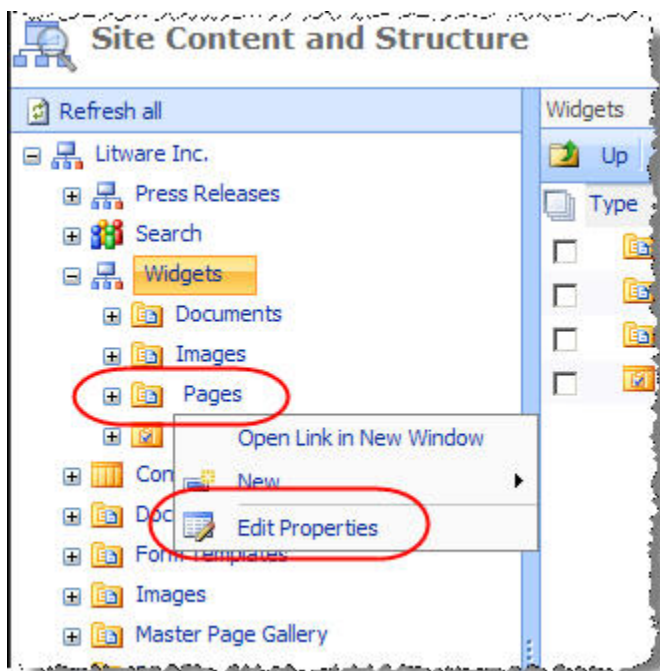
2. Enter the following command into the command line window and hit Enter:

```
stsadm -o addsolution -filename c:\Student\Labs\10_Workflow\Lab\wsp\Debug\Lab10.wsp
```

3. Launch Central Administration by selecting **Start » All Programs » Microsoft Office Server » SharePoint 3.0 Central Administration**.
4. From the **Central Administration** site, select the **Operations** tab and then select **Solution management** under the **Global Configuration** section.
5. On the **Solution Management** page, click the link on **lab10.wsp**.
6. On the **Solution Properties** page, select **Deploy Solution**.
7. On the **Deploy Solution** page, specify **Now** in the **Deploy When?** section and click **OK**.
8. Browse to the **http://wcm.litwareinc.com/** site and select **Site Actions » Site Settings » Modify All Site Settings**.
9. Under the **Site Collection Administration** section, select **Site collection features**.
10. On the **Site Collection Features** page, click the **Activate** button for the **Lab 10 - ... Feature**.
11. Now you need to associate the workflow with a Pages list to see it work. You will do this to the Pages list within the Widgets sample site that was created using the dummy content creator tool in the Web Parts.

If you didn't run the tool to create the dummy content, refer back to the Web Parts lab for instructions on how to create the dummy content using the provided utility.

The easiest way to do this is to jump to the **Site Content and Structure** page by selecting **Site Actions » Manage Content and Structure**. Then, open the **Widgets** subsite, followed by selecting **Edit Properties** from the **Pages** list's ECB menu to jump straight to the Pages' settings page:



12. On the **Customize Pages** page, select **Workflow settings** under the **Permissions and Management** section.
13. Now, what you want to do is not only add your custom workflow, but also remove or deactivate the Parallel Approval workflow so it doesn't get in the way. Click **Remove a workflow**, then set the **Parallel Approval** workflow to **No New Instances** and click **OK**.
14. Next, click **Add a workflow** and use the following values to complete the **Add a Workflow: Pages** page... leave all default settings where not specified below and click **Next**:
 - **Workflow:** Majority Approvers
 - **Name:** Majority of Three Approvers

15. Now we're ready to test!

In this exercise you deployed and associated the workflow with an existing list.

Exercise 7: Test the workflow

Now the part you've worked so hard to get to... to see this guy in action! In this exercise you will test your workflow on some existing pages.

Note that the buttons in the Page Editing Toolbar's Quick Access Button area are configured to work with the Parallel Approval workflow, so you'll have to manually fire it off and check the status. This is a good thing so you can see all parts of this workflow.

1. Navigate to **Widget Product 1** page
(<http://wcm.litwareinc.com/Widgets/Pages/WidgetProduct1.aspx>)... the page should not be checked out, it should be published. If it is, pick another page that is published.
2. Select **Site Actions » Edit Page**. No need to make any changes, just click **Check In To Share Draft** on the Quick Access Button area. Then from the **Page Actions** area, select **Workflow » Start a Workflow....**
3. On the **Workflows: WidgetProduct1** page, select **Majority of Three Approvers** as shown in the following image:



4. On the **Start "Majority of Three Approvers": WidgetProduct1** page, use the following information in the initiation form... notice this is your **InitForm.xsn** InfoPath 2007 form!
 - **Approver1:** LITWAREINC\brianc
 - **Approver2:** LITWAREINC\angelab
 - **Approver3:** LITWAREINC\jayh
 - **Instructions:** Please review this page and either approve or reject it.
5. After the workflow starts, from the **Page Actions** area, select **Workflow » View Workflow Status....** Notice the tasks have been created and assigned and there is a lot of information that's been logged to the history as shown in the following image:

Workflow Status: Majority of Three Approvers

Workflow Information

Initiator: Litware Admin Guy Document: WidgetProduct1
 Started: 7/7/2007 8:32 PM Status: In Progress
 Last run: 7/7/2007 8:32 PM

If an error occurs, you can terminate the workflow. Terminating the workflow will set its status to Canceled and will delete all tasks created by the workflow.

Tasks

The following tasks have been assigned to participants in this workflow. Click a task to edit it. You can also view these tasks in the list Workflow Tasks.

Assigned To	Due Date	Status	Outcome
Brian Cox	7/14/2007	Not Started	
Angela Barbariol	7/14/2007	Not Started	
Jay Henningsen	7/14/2007	Not Started	

Workflow History

The following events have occurred in this workflow.

Date Occurred	Event Type	User ID	Description
7/7/2007 8:32 PM	Comment	System Account	Collected three approvers: , & . All received the following instructions:
7/7/2007 8:32 PM	Comment	System Account	Approval task 'alpha' created and assigned to LITWAREINC\brianc
7/7/2007 8:32 PM	Comment	System Account	Approval task 'beta' created and assigned to LITWAREINC\angelab
7/7/2007 8:32 PM	Comment	System Account	Approval task 'charlie' created and assigned to LITWAREINC\jayh

Workflow is currently running...

Three tasks have been created, each assigned to a different approver you specified.

The MajorityApprovers workflow has created the tasks and notice it has logged quite a few messages... its working!!!

- Now, open each of the tasks and try different combinations of selecting Approve or Reject and watch how the workflow reacts!

In this exercise you have successfully tested your workflow.

Lab 12: Implementing Multilingual Sites Using Variations

Lab Time: 30 minutes

Lab Overview: In this lab you will practice setting up and configuring variations which will be used within a multilingual site. Variations can be used for multilingual solutions as well as multidevice solutions and even used for delivering content in different vehicles.

Exercise 1: Configuring Variations for a Multilingual Site

In this exercise you will create a new variation container and a handful of labels. These labels will then be used to create a multilingual site.

1. Before you begin, create a new subsite within the **http://wcm.litwareinc.com** site collection that will be used as the variation container. Open a browser and navigate to **http://wcm.litwareinc.com**, then select **Site Actions » Create Site**.
2. On the **New SharePoint Site** page, use the following information to complete the page and then click **Create**:
 - **Title:** Variation Root
 - **Web Site Address:** http://wcm.litwareinc.com/VariationRoot
 - **Template Selection:** Publishing site with Workflow
 - **Permissions:** Use same permissions as parent site
 - **Navigation Inheritance:** Yes
3. In order to implement variations on a Publishing site, you need to first configure the site's variation settings. Open a browser and navigate to **http://wcm.litwareinc.com**, then select **Site Actions » Site Settings » Modify All site Settings** and then select **Variations** under the **Site Collection Administration** section.
4. On the **Variations Settings** page, use the following information to configure variations for the **http://wcm.litwareinc.com** site collection:
 - **Variation Home:** http://wcm.litwareinc.com/VariationRoot
 - **Automatic Creation:** Automatically create site and page variations
 - **Recreate Deleted Target Page:** Recreate a new target page when the source page is republished
 - **Update Target Page Web Parts:** Do not update Web Part changes to target pages when variation source page is propagated
 - **Notification:** Uncheck Send e-mail notification when a new site or page is created or a page is updated by the variations system.
 - **Resources:** Copy Resources

- At this point, you have now configured variations for the **http://wcm1.litwareinc.com** site collection. The next step is to create labels for each variation flavor.

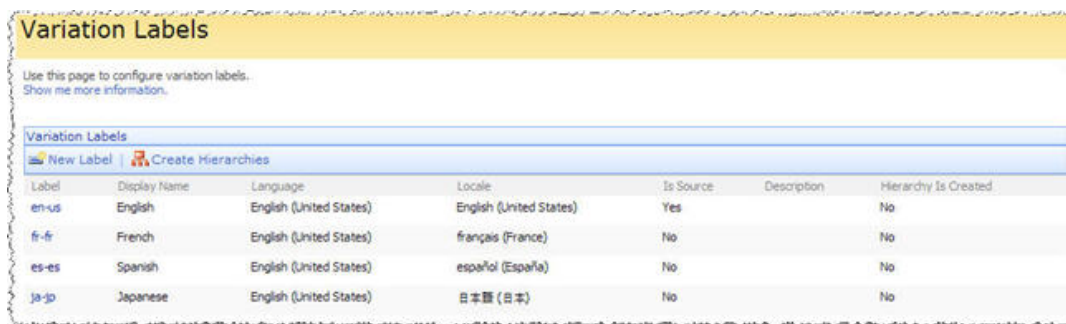
Exercise 2: Creating Variation Labels

In this exercise you will create a few variation labels that will represent a few different languages within your site collection for the subsite **Variations Root**.

- Navigate to the root of the **http://wcm.litwareinc.com** site collection and select **Site Actions » Site Settings » Modify All Site Settings** and then select **Variation Labels** under the **Site Collection Administration** section.
- You will now create three new variations for three different languages. On the **Variation Labels** page, select **New Label**.
- On the **Create Variation Label** page, use the following information to create a new label and click **OK**:
 - **Label Name:** en-us
 - **Display Name:** English
 - **Locale:** English (United States)
 - **Hierarchy Creation:** Publishing Sites and All Pages
 - **Source Variation:** Check **Set this variation to be the source variation**.
 - **Select the Publishing site template you want to use:** Publishing Site with Workflow
- Now, using the same process in the previous step, create three more labels using the following information:

	Label #1	Label #2	Label #3
Label Name:	fr-fr	es-es	ja-jp
Display Name:	French	Spanish	Japanese
Locale:	French (France)	Spanish (Spain)	Japanese

- The **Variation Labels** page should now look like the following image:



6. Now you need to create the site hierarchy with all the labels configured. Select the link **Create Hierarchies** from the **Variation Labels** page toolbar. The variation system will now create subsites for each of the labels and establish the necessary links between each.
7. When the hierarchy creation process is complete, select the **Variation Root** menu item from the top navigation bar. Unless your desktop is configured with a locale of fr-fr, es-es, or ja-jp, you will be automatically redirected to **<http://wcm.litwareinc.com/VariationRoot/en-us/Pages/default.aspx>**. This is because the variation system adds a special control to the default page within the **Variation Root** site that automatically redirects users to the label that is configured with the same locale configured on their desktops.

In this exercise you created four variation labels and the necessary hierarchies to make this work.

Exercise 3: Creating Content in a Multilingual Site

In this exercise you will create content in the root variation label and watch it propagate to the other labels when published.

1. Navigate to the root variation, **<http://wcm.litwareinc.com/VariationRoot/en-us/Pages/default.aspx>**.
2. Now, create two new content pages using the **Article Page** content type... use any of the page layouts associated with the Article Page. Do not take time entering too much content into the pages, but enter just enough. **Do not submit the pages for approval... simply check them in.**
3. Once you've created the two pages, navigate to the other labels and notice how the pages do not exist.
4. Go back to the pages you created and move them all the way through the approval process to publish them to the site.
5. Once the pages have been published, navigate to one of the other labels. Notice how the pages have been copied over (if they haven't, give it a few minutes as the propagation procedure is not always immediate), however they aren't published. At this point, content owners responsible for the different language variation labels would perform translation; optionally change the master page (as long as it was associated to the same **Article Page** content type).

In this exercise you created some content and moved them through the publishing approval process to monitor them making its way through the other labels.