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****.*

1. 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

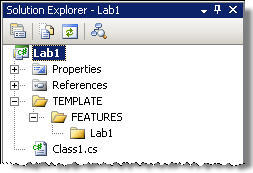
In this exercise you will create a new WSS 3.0 Feature that will be deployed using the WSS solution package framework. Once the Feature has been installed, upon activation, it will create a new page within the site collection created in Exercise 1 based off a template that will be included with the Feature.

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

1. 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:



1. 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.

1. 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.

1. 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]](#footnote-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.

1. 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****.*
2. With the project unloaded, right-click the project again in the **Solution Explorer** tool window and select **Edit Lab1.csproj**.
3. 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" />

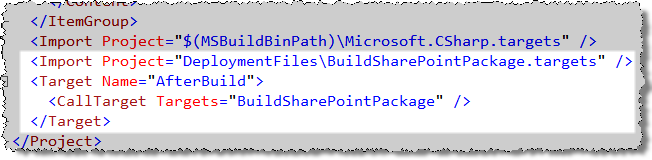
Once the custom MSBuild targets file has been imported, you need to tell MSBuild that after the project has been built, it needs to call your custom target (again, defined in the \*.targets file added to our project). Add the following XML, replacing the XML comment:

<Target Name="AfterBuild">

<CallTarget Targets="BuildSharePointPackage" />

</Target>

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



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!

1. 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>

1. 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

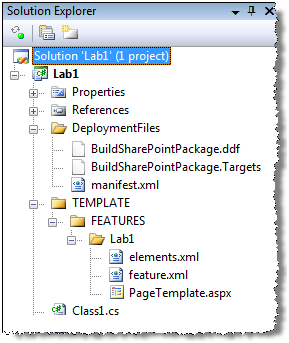
.Set DestinationDir=Lab1

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:



1. 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)

{

}

1. 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.*

1. 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" />

1. Make sure you code behind compiles by building the project. If not, go back and fix any errors.
2. 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.

1. 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.
2. 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;

1. 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) {

}

}

1. 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);

1. 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;

}

}

1. 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.

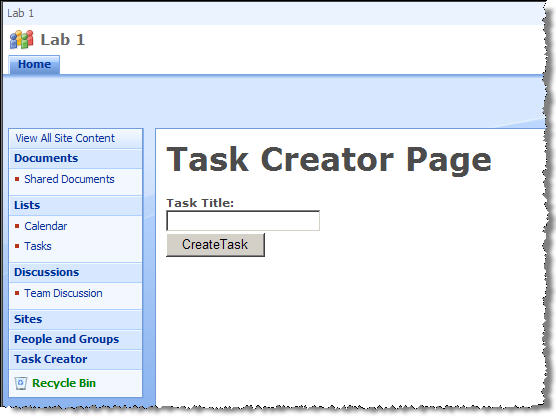
1. First the WSS solution package must be deployed. Open a command prompt and navigate to the following directory:

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

1. 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

1. Launch Central Administration by selecting **Start » All Programs » Microsoft Office Server » SharePoint 3.0 Central Administration**.
2. From the **Central Administration** site, select the **Operations** tab and then select **Solution management** under the **Global Configuration** section.
3. On the **Solution Management** page, click the link on **lab1.wsp**.
4. On the **Solution Properties** page, select **Deploy Solution**.
5. On the **Deploy Solution** page, specify **Now** in the **Deploy When?** section, **http://litwareinc.com** in the **Deploy To?** section and click **OK**.
6. 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/lab**1 site and select **Site Actions » Site Settings**.
7. On the **Site Settings** page, select **Site features** under the **Site Administration** section.
8. 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.
9. 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:



1. 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!

1. <http://support.microsoft.com/default.aspx/kb/310618> [↑](#footnote-ref-1)