Lab 03: Developing Custom Application Pages

**Lab Time**: 45 Minutes

**Lab Directory**: C:/Student/Labs/03\_Architecture

**IMPORTANT NOTE:**Because we are building this project piecemeal and testing it as we go, if after you build your project 1 time, the next change and build causes an error when you try to test your addition (i.e. with either SiteInfo.aspx or DocumentInfo.aspx) **YOU WILL NEED TO RE-BUILD YOUR PROJECT AGAIN IN ORDER TO TEST IT OUT**.  During the second build your old files were removed but not replaced correctly so an additional re-build of your project will correctly re-deploy these pages.  (This tends to occur during Exercise 3 in this lab).

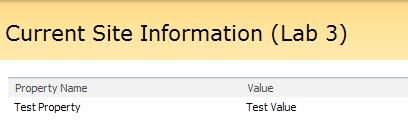
**More Info on Important Note:** This issue tends to be much more prevalent when working with VB.NET as your source language.  As service packs for Visual Studio and .Net are released this issue may become largely mitigated.

**Lab Overview**: In this lab, you will work with a Visual Studio project to write, deploy and test custom application pages that run out of the virtual **\_layouts** directory. This will demonstrate a few ways to use custom application pages and also give you a sense for what it's like to develop them for a WSS solution. At the end of this lab, you will also work with batch files to take several WSS components and compile them into a solution package so that they can be deployed in staging and production environments.

# Exercise 1: Provisioning a new Site Collection for testing

1. Make sure you are logged on as **LITWAREINC\Administrator**. Launch the **WSS Central Administration Web** site by using the command in the Windows Start menu. You can find the command at **Start >> Administrative Tools >> SharePoint 3.0 Central Administration**.
2. Once the home page of the WSS Central Administration Web site appears, click on the **Application Management** tab at the top of the page to get to the Application Management page. Under the **SharePoint Site Management** section, click the link with the caption **Create site collection**. This will take you to the page where you can create a new site collection and a new top-level site. On the **Create Site Collection** page, fill in the required information (see below for instructions) to create a new site collection.
   1. Make sure the Web Application used for site creation is the one named **Litware Public Site** that is accessible through the URL **http://litwareinc.com**.
   2. Enter a title such as **Lab 3 Test Site**.
   3. Create the new site collection so that its URL is **http://litwareinc.com/sites/Lab3**.
   4. Under the **Template Selection** section, look at all the sites templates that are available. Choose **Blank Site** from the **Collaboration** tab as the site template for the new top-level site that is automatically created.
   5. Assign the primary site collection owner as **LITWAREINC\Administrator**.  Be sure to verify this by using Ctrl+ k or by clicking the check person icon checkIcon (your entry should become LitwareInc Administrator)
   6. Leave the **Quota Template** with the default setting of **No Quota**.
   7. Click **OK** to create the new site collection and top-level site. Once you see that page that confirms that everything has been created, navigate to the top-level site using the browser (click the link <http://litwareinc.com/sites/Lab3>.)
3. Once you have created and navigated to this new site collection, you can go on to the next exercise.

# Exercise 2: Working with a Custom Application Page

1. Launch Visual Studio.
2. Open the project named **Lab3.sln** located inside the **\Student\Labs\03\_Architecture\Lab\VB** or **\Student\Labs\03\_Architecture\Lab\C#** directory.
   1. This project contains two custom application pages named **SiteInfo.aspx** and **DocumentInfo.aspx** (located in the **Solution Explorer TEMPLATE > LAYOUTS > Lab3** directory), as well as various other source files required to deploy and integrate it into a custom business solution.
   2. Use the **Solution Explorer** to get an idea of the project's directory structure as well as what source files are included as part of this project. Make sure to inspect all subdirectories of the **\TEMPLATE** directory.
3. Open **SiteInfo.aspx** and look inside. You should be able to see that this ASP.NET page is defined to inherit from a class defined inside **SiteInfo.cs or SiteInfo.vb** named **Lab3.SiteInfo.** You should notice that this page file contains no code; only HTML mark up and a server control tag to create an instance of the **SPGridView** control named **grdSiteProperties**.
4. Open **SiteInfo.cs or SiteInfo.vb** and find the class named **Lab3.SiteInfo.** Note that this code-behind class defines a control field using the same name and type that was used in the server control tag in **SiteInfo.aspx**. This technique is what allows method implementation inside the code-behind class **SiteInfo.cs** to access the **SPGridView** control named **grdSiteProperties**.
5. Examine the Feature that is defined in **feature.xml** and **elements.xml** (located in **Solution Explorer TEMPLATE > FEATURES > Lab3 directory**).
   1. Inside the **elements.xml** file you should be able to see that this Feature defines a single **CustomAction** element to add a custom menu item to the **Site Actions** menu that will allow a user to navigate **SiteInfo.aspx**.
6. Build the project. This will recompile the project's output assembly **Lab3.dll** and then run **Install.bat** which contains commands to install **Lab3.dll** in the GAC, to copy the Feature files into the WSS **FEATURES directory** and to install (or reinstall) the Feature itself (note: just like in Lab 2 the Install.bat file also runs an IISRESET command).  At this point, the Feature should be installed and ready for activation in any site within the current farm.
7. Navigate to the site you created in the first exercise located at is **http://litwareinc.com/sites/Lab3**.
   1. Click on **Site Actions > Site Settings** to navigate to the **Site Settings page.**
   2. On the **Site Settings page** click the link to go to the **Site Feature** management page (this link is located under the **Site Administration** column).
   3. You should be able to see the Feature for this lab with a title of **Lab 3 - Working with Application Pages**.
   4. Click on **Activate** to enable this Feature.
8. Open the **Site Actions** menu. You should see a menu item with the title of **Get Site Info**.
   1. Select that menu item to navigate to **SiteInfo.aspx**. The page **SiteInfo.aspx** should render and display an **SPGridView** control with one row of test data.  
      
9. Now return to Visual Studio and look at the **OnLoad** event handler method for the page within **SiteInfo.cs or SiteInfo.vb**. You should see that this code works with a utility class named **PropertyCollectionBinder** to populate the **SPGridView** control named **grdSiteProperties**.
10. Inspect the **PropertyCollectionBinder** class defined inside **PropertyCollectionBinder.cs or  PropertyCollectionBinder.vb** in order to get an understanding of how this utility class encapsulates creating a **DataTable** and binding to any **SPGridView** control. Note: you don't have to modify the code inside the **PropertyCollectionBinder** class. You will just leverage the utility class as it already exists.
11. Now, go back to the **OnLoad** method of the **SiteInfo.cs** or **SiteInfo.vb** page class and locate the following line of code.

C#

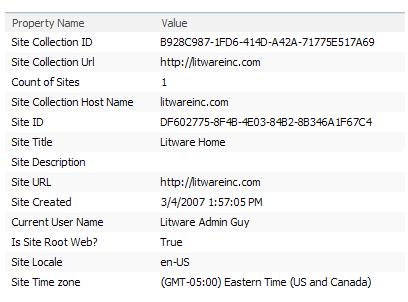
// remove the next line and begin your work here

binder.AddProperty("Test Property", "Test Value");

**VB.NET**

' remove the next line and begin your work here

binder.AddProperty("Test Property", "Test Value")

1. Remove that line of code and replace it with code which programs against the WSS object model to populate the **SPGirdView** control with name value pairs which display information about the current site collection and the current site. In this step you will be forced to use whatever means available to you (IntelliSense, Visual Studio object browser, etc) to find the correct methods you need within the WSS object model to complete the job.
   1. **Hint**: Your **SiteInfo.vb or SiteInfo.cs** file has an **OnLoad()** method that defines two variables **siteCollection** and **site** that may be **EXTREMELY** **useful** in solving this problem.
   2. **Further Hint**: copy the **binder.AddProperty** line and **replace** "Test Property" with the Property names in the table below (see table in step 13).  **Replace** "Test Value" not with a string value but with a property of either **siteCollection** or **site** to expose the requisite information.
2. You should be able to rebuild the project and then refresh the browser to test your work. When you are done, the resulting **SPGridView** control should look approximately like the one below.  
     
   

# Exercise 3: Writing an Application Page to display document information

1. Now it's time to work on a second application page. This one will display information to the user about a specific document. The project already includes the source files for a custom application page **DocumentInfo.aspx** in the same location as **SiteInfo.aspx**. Also there is a code-behind class file named **DocumentInfo.cs or DocumentInfo.vb** in the same location as **SiteInfo.cs or SiteInfo.vb**.
2. Open **DocumentInfo.aspx** and **DocumentInfo.cs or DocumentInfo.vb** in code view and take a minute to review all the code inside these files.
3. The new page named **DocumentInfo.aspx** will be used to display information about a specific document. Therefore, it makes sense to add a new custom ECB (Edit Control Box) menu item for all documents within all document libraries. The purpose of this new ECB menu is to provide users with a command so that they can redirect to **DocumentInfo.aspx** and get more information about a specific document upon request.
   1. Go to the **elements.xml** file and add the following element.

<CustomAction

Id="DocumentInfo"

RegistrationType="List"

RegistrationId="101"

ImageUrl="/\_layouts/images/GORTL.GIF"

Location="EditControlBlock"

Sequence="10"

Title="Get Document Info" >

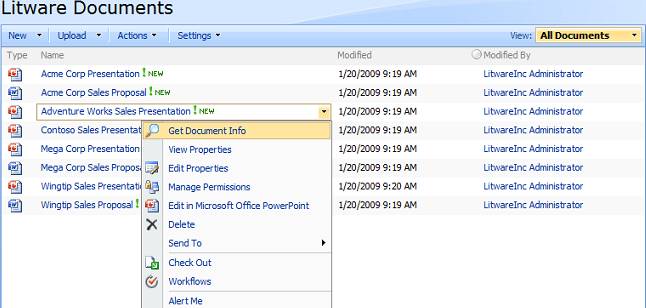
<UrlAction Url="~site/\_layouts/Lab3/DocumentInfo.aspx"/>

</CustomAction>

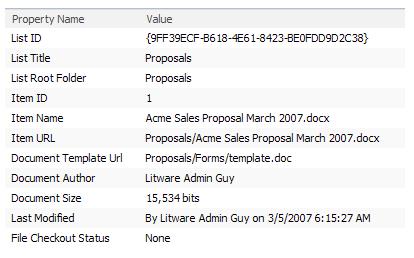
* 1. The CAML you have just added to **elements.xml** is not yet complete. When a user redirects from a particular document to **DocumentInfo.aspx**, the code behind this custom application page will require query string parameters to get information about the GUID ID for the document library (i.e. SPList object) and the integer ID for the specific document (i.e. SPListItem object). Modify the **Url** attribute of the **UrlAction** element with the following value.

~site/\_layouts/Lab3/DocumentInfo.aspx?ItemId={ItemId}&amp;ListId={ListId}

1. Now rebuild the project. This should copy the updated version of the Feature files into the **\FEATURES** directory, reinstall the new updated Feature definition and then run an IISRESET operation.
2. Now it's time to check your work and make sure the ECB menu appears as it should.
   1. Return to the test site and create a new document library. When you create the new document library, you can name it anything you would like.
   2. Once you have created the document library, add a new document or upload an existing document.
   3. Once you have added a document to the document library, drop down the document's Edit menu (i.e. in the document library when you point to your newly created\uploaded document a drop down arrow appears inside of a box that appears around the document.).  When you click on the drop down arrow or the Edit Control Box, you should see the ECB menu.



* 1. When you drop down the ECB menu for this document, you should see the new custom ECB menu item that was added by our Feature (i.e. the **Get Document Info** choice).
  2. When you select this custom ECB menu item, you should be redirected to **DocumentInfo.aspx**. However, the code behind this application has not yet been updated to show any information about the current document (therefore, we will receive our test data on this page when it tries to load).

1. Update the **OnLoad** event handler method inside **DocumentInfo.cs** to fetch the query string parameters named **ItemId** and **ListId**. Use these values to create an **SPList** object for the document library and an **SPListItem** object for the document. Now, program against the WSS object model to get the information required to make the **SPGridView** control on **DocumentInfo.aspx** look like this.  
     
   
   1. Once again, we have left it up to you to use IntelliSense and the Visual Studio object browser to find the correct methods in the WSS object model. If you get stuck and you want some help, you can look at the **DocumentInfo.cs** file in the solution directory for the lab at **\Student\Labs\03\_Architecture\Solution**. When you are done, rebuild the project and refresh the browser to test your work.
   2. **Hint**: In order to facilitate working with the object model you should create a couple of variables to help you traverse the appropriate properties.  Using these variables and the structure outlined below you should be able to add the appropriate properties to your **DocumentInfo** code file:

C#

// get list information

string ListId = Request.QueryString["ListId"];

SPList list = site.Lists[new Guid(ListId)];

// get list item information

string ItemId = Request.QueryString["ItemId"];

SPListItem item = list.Items.GetItemById(Convert.ToInt32(ItemId));

// provided the list is a Document Library, get the fine info for the list item

if (list is SPDocumentLibrary) {

SPDocumentLibrary documentLibrary = (SPDocumentLibrary)list;

SPFile file = site.GetFile(item.Url);

}

VB.NET

' get list information

Dim ListId As String = Request.QueryString("ListId")

Dim list As SPList = site.Lists(New Guid(ListId))

' get list item information

Dim ItemId As String = Request.QueryString("ItemId")

Dim item As SPListItem = list.Items.GetItemById(Convert.ToInt32(ItemId))

' provided the list is a Document Library, get the fine info for the list item

If TypeOf list Is SPDocumentLibrary Then

Dim documentLibrary As SPDocumentLibrary = CType(list, SPDocumentLibrary)

Dim file As SPFile = site.GetFile(item.Url)

End If

* 1. **Further Hint:** Open the **DocumentInfo.cs** or **DocumentInfo.vb** file from the solution to see the relevant properties.

# Exercise 4: Working with a Solution Package to deploy this WSS solution

1. Now it is time to use a solution package to deploy this entire solution of WSS components from your development machine into a staging environment or a production environment.
   1. The first thing you need to do is to uninstall and remove all the components from your development machine. Start up a command prompt and run the following batch file named **Uninstall.bat** to reverse everything that was done by **Install.bat**. You can find this batch file at the following path.

****C#****

****\Student\Labs\03\_Architecture\Lab\C#****\Uninstall.bat

****VB.NET****

****\Student\Labs\03\_Architecture\Lab\VB\Uninstall.bat****

1. Once you have run **Uninstall.bat**, you should be able to verify that the **Lab3** feature is no longer activated or installed. You can do this by going to the **Site Feature** management page (**/\_layouts/ManageFeatures.aspx**or **Site Actions > Site Settings > Site Features**) and verifying that the **Lab3** feature is not available for activation.
2. Now within the current project, use the Visual Studio Solution Explorer to look at the files inside the **Solution** folder. Locate the two files named **manifest.xml** and **cab.ddf**. You will not be modifying these files in this exercise, but just be looking at them. Once you have seen what is inside, you will then run batch files to build and deploy the solution package.
3. First, inspect **manifest.xml**. This file represents the top-level metadata file for the solution package and will be read by the WSS installer whenever the solution package is installed or deployed.

<Solution

SolutionId="42262980-404A-4e1c-916A-FD72B031AEDF"

xmlns="http://schemas.microsoft.com/sharepoint/" >

<FeatureManifests>

<FeatureManifest Location="Lab3\feature.xml" />

</FeatureManifests>

<TemplateFiles>  
 <TemplateFile Location="LAYOUTS\Lab3\DocumentInfo.aspx"/>  
 <TemplateFile Location="LAYOUTS\Lab3\SiteInfo.aspx"/>

<TemplateFile Location="IMAGES\TPG\PithHelmet.gif"/>

</TemplateFiles>

<Assemblies>

<Assembly Location="Lab3.dll" DeploymentTarget="GlobalAssemblyCache" />

</Assemblies>

</Solution>

1. Next, look at what is inside the **cab.ddf** file. These are the instructions used to generate the solution package named **Lab3.wsp**. **Note** that there is an **individual line for each file that needs to be added to the solution package**. As you can imagine, maintaining a **.ddf** file and keeping it in sync with the **manifest.xml** file is a very tedious and error-prone undertaking. **Unfortunately**, **Microsoft provides no developer tools to help create or maintain these files.**

.OPTION EXPLICIT ; Generate errors

.Set CabinetNameTemplate=Lab3.wsp

.set DiskDirectoryTemplate=CDROM ; All cabinets go in a single directory

.Set CompressionType=MSZIP;\*\* All files are compressed in cabinet files

.Set UniqueFiles="ON"

.Set Cabinet=on

.Set DiskDirectory1=Package

Solution\manifest.xml manifest.xml

TEMPLATE\FEATURES\Lab3\feature.xml Lab3\feature.xml

TEMPLATE\FEATURES\Lab3\elements.xml Lab3\elements.xml

TEMPLATE\LAYOUTS\Lab3\SiteInfo.aspx LAYOUTS\Lab3\SiteInfo.aspx

TEMPLATE\LAYOUTS\Lab3\DocumentInfo.aspx LAYOUTS\Lab3\DocumentInfo.aspx

TEMPLATE\IMAGES\TPG\PithHelmet.gif IMAGES\TPG\PithHelmet.gif

bin\Debug\Lab3.dll Lab3.dll

;\*\*\* <the end>

1. Now you have seen the two main files used to build a solution package. As you may know, the **cab.ddf** file is used as input to the **MAKECAB.EXE** utility to create a solution package named **Lab3.wsp**. The **manifest.xml** file is included inside the solution package and this file acts as the main manifest used by the WSS installer during solution package deployment.
2. Now within the current project, use the Visual Studio Solution Explorer to look at the files inside the **Package** folder. You should see the following four batch files which have been created to assist you with creating and managing the solution package:
   1. **CreateSolutionPackage.cmd**
   2. **DeleteSolutionPackage.cmd**
   3. **DeploySolutionPackage.cmd**
   4. **InstallSolutionPackage.cmd**
3. Inspect (**but do not run**) the batch file named **CreateSolutionPackage.cmd** which has been written to build **Lab3.wsp**. The contents of this batch file look like this:

@echo off

if EXIST Lab3.wsp del Lab3.wsp

cd ..

makecab /f Solution\cab.ddf

cd Package

1. Inspect (**but do not run**) the batch file named **InstallSolutionPackage.cmd** to install **Lab3.wsp** into the farm-scoped solution package store. The contents of this batch file look like this:

@SET SPDIR="c:\program files\common files\microsoft shared\web server extensions\12"

%SPDIR%\bin\stsadm -o addsolution -filename Lab3.wsp

1. Finally, inspect (**but do not run**) the batch file named **DeploySolutionPackage.cmd**. This batch file accomplishes the same steps as you have seen with **CreateSolutionPackage.cmd** and **InstallSolutionPackage.cmd** and then it deploys the **Lab3.wsp** package in the current farm. You can also see that this batch **file begins by retracting and removing any previous version of this solution package before reinstalling and redeploying it**. Now, take a detailed look through the batch file named **DeploySolutionPackage.cmd**.

@SET STSADM="c:\program files\common files\microsoft shared\web server extensions\12\bin\stsadm.exe"

Echo Retracting Solution Package Lab3.wsp

%STSADM% -o retractsolution -name Lab3.wsp -immediate

%STSADM% -o execadmsvcjobs

Echo Deleting Solution Package Lab3.wsp

%STSADM% -o deletesolution -name Lab3.wsp -override

%STSADM% -o execadmsvcjobs

Echo Generating Solution Package Lab3.wsp

if EXIST Lab3.wsp del Lab3.wsp

cd ..

makecab /f Solution\cab.ddf

cd package

Echo Installing Lab3.wsp in WSS Solution Package Store

%STSADM% -o addsolution -filename Lab3.wsp

%STSADM% -o execadmsvcjobs

Echo Deploying Solution Package Lab3.wsp

%STSADM% -o deploysolution -name Lab3.wsp -immediate -allowGacDeployment -force

%STSADM% -o execadmsvcjobs

1. Now it's time to update the post-build event of the Visual Studio project named Lab3. Currently, the post-build event instructions for the project are configured to run **Install.bat**.

cd $(ProjectDir)

Install.bat

* 1. Modify the post-build event instructions to run **DeploySolutionPackage.cmd** instead of **Install.bat**. Note that you must add a line to change the current directory to the **Package** directory before you run **DeploySolutionPackage.cmd**.

cd $(ProjectDir)Package

DeploySolutionPackage.cmd

* 1. In **Solution Explorer** right-click the project and choose **Properties** to view the **Project Properties** for the **Lab3** project.  If using VB go to step i.  If using C# go to step ii.

Visual Basic .Net directions:

* + - 1. Navigate to the **Compile** tab
      2. Click on the **Build Events...** button (scroll down if necessary to see this button)
      3. Edit  the Post-build event command line instructions so that they match the following.

cd $(ProjectDir)Package

DeploySolutionPackage.cmd

* + - * 1. Note that the first line with **cd $(ProjectDir)Package** is required to change the current directory to that of the project directory. The second line runs the command file **DeploySolutionPackage.cmd** to deploy this package.

C# directions:

1. Navigate to the **Build Events** tab.
2. Add the following Post-build event command line instructions.

cd $(ProjectDir)Package

DeploySolutionPackage.cmd

* + - * 1. Note that the first line with **cd $(ProjectDir)Package** is required to change the current directory to that of the project directory. The second line runs the command file **DeploySolutionPackage.cmd** to deploy this package.

1. Now, rebuild the project. Monitor the progress of the run **DeploySolutionPackage.cmd** by inspecting the **Output** window for Visual Studio.
   1. From the Visual Studio **Build** menu select **Build Lab 3.**
2. Now, it’s time to make sure everything has been installed correctly and that your solution works after the solution package has been deployed.
   1. Navigate to **Site Feature** management page (**/\_layouts/ManageFeatures.aspx**or **Site Actions > Site Settings > Site Features**).
   2. Ensure the **Lab3** feature is there and can be activated.
   3. Once you have activated it, go to the Document Library you created and test out the ECB menu item and custom application pages to make sure they work.
3. If you are interested, go through the steps of retracting the solution package as well.
   1. Start by deactivating the **Lab3** feature in the current site.
   2. Once you have deactivated the **Lab3** feature, bring up a command prompt and run the batch file named **DeleteSolutionPackage.cmd** which contains the follow command-line instructions.

@SET STSADM="c:\program files\common files\microsoft shared\web server extensions\12\bin\stsadm.exe"

%STSADM% -o retractsolution -name Lab3.wsp -immediate

%STSADM% -o execadmsvcjobs

%STSADM% -o deletesolution -name Lab3.wsp