

Microsoft® Virtual Labs

A SharePoint Developer Introduction – C#

Microsoft®

Table of Contents

A SharePoint Developer Introduction	1
Exercise 1 Hello World	2
Exercise 2 Web Part Interaction	10
Exercise 3 Connecting Web Parts.....	19

A SharePoint Developer Introduction – C#

Objectives

Web Parts are one of the core ASP.NET technologies used by SharePoint to present dynamic information to users. They are the most common customization created for SharePoint. A Web Part is a reusable component which exists on a Web Part Page and can present any type of web-based information.

The objective of this lab is to learn about how to build Web Parts for use in the SharePoint system.

- Create a basic SharePoint Web Part which displays information from within the SharePoint site.
- Create a more advanced Web Part utilizing a server postback and the SPGridView control.
- Create an advanced Web Part leveraging Web Part connection that displays budget data.

Scenario

In this exercise, you will develop and deploy your first

- Display text on a label within a Web Part
- Allow the text on the label to be edited and colored.
- Create a Web Part that allows users to navigate to sites and lists within a site collection. This will utilize the **SPGridView** control. The grid view will display two columns – one for sites the other for lists. Each column will contain **ButtonField** controls that perform a postback when clicked. This exercise will also demonstrate how to use existing ASP.NET user controls in SharePoint.
- Build a Dashboard Web Part that summarizes sales of widgets. Monthly sales records will be recorded to a custom list. The Dashboard Web Part will connect to a standard ListView Web Part on the same page.

Estimated Time to Complete This Lab

60 Minutes

Computers used in this Lab



Server1


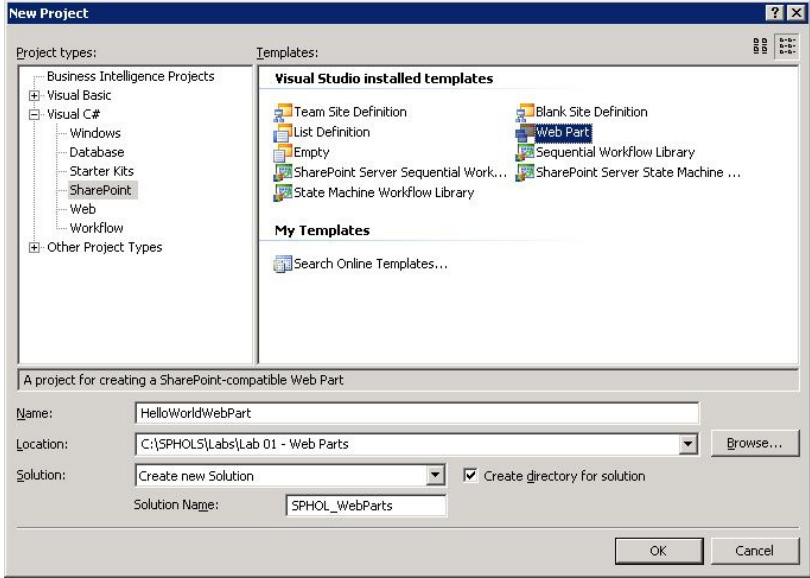
Exercise 1

Hello World


Scenario

In this exercise, you will develop and deploy your first Web Part. This Web Part will:

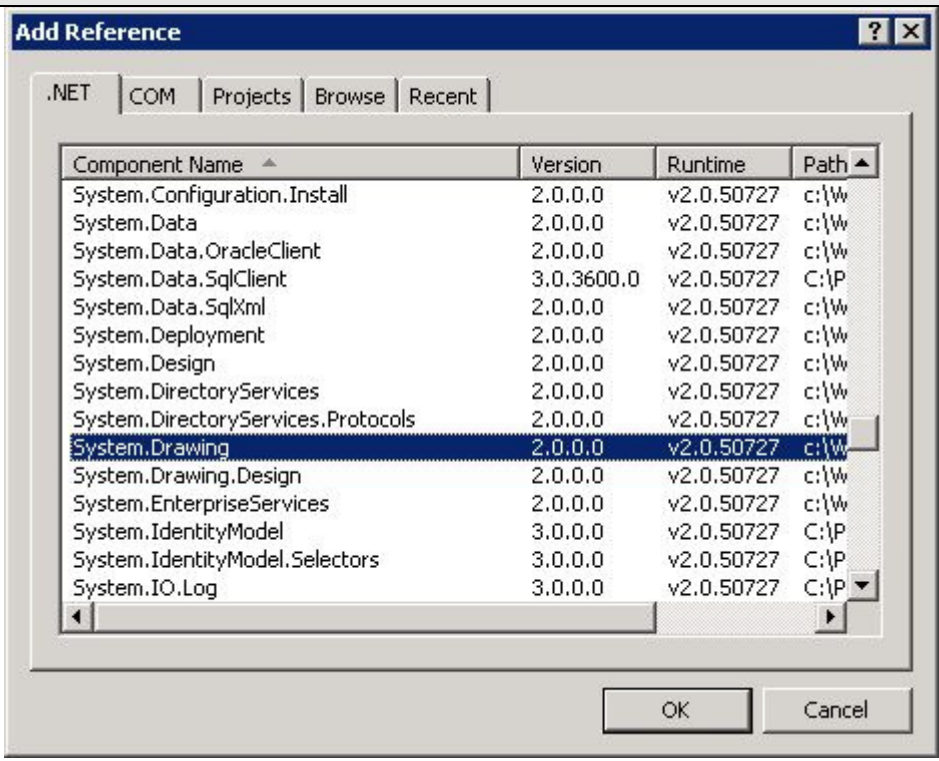
- Display text on a label within a Web Part
- Allow the text on the label to be edited and colored.


Tasks	Detailed Steps
<p>Complete the following tasks on:</p> <p> Server1</p> <p>1. Create a SharePoint Web Part Solution</p>	<p>a. Open Visual Studio 2005 by going to the Start Menu Programs Microsoft Visual Studio 2005 Microsoft Visual Studio 2005.</p> <p>b. From the menu, select File New Project.</p> <p>c. In the New Project dialog box, expand the Visual C# > SharePoint project type and select Web Part.</p> <p>d. Enter “C:\SPHOLS\Labs\Lab 01 - Web Parts” for the Location.</p> <p>e. Enter HelloWorldWebPart for the Name.</p> <p>f. Enter SPHOL_WebParts for the Solution Name.</p>  <p>g. Click Ok.</p> <p>h. The HelloWorldWebPart project can now be seen in the solution folder. Note that a folder called WebPart1 is created and contains a few files.</p>

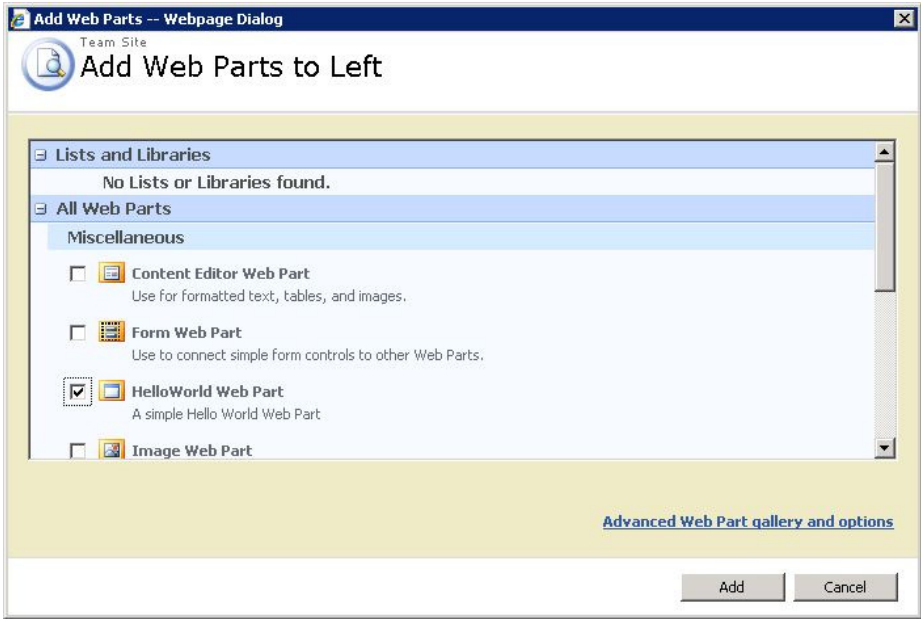
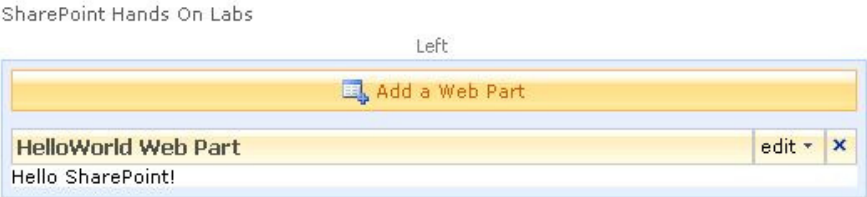
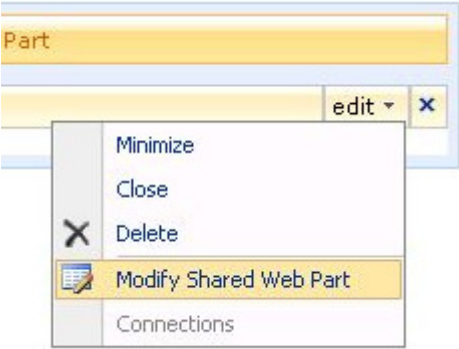
Tasks	Detailed Steps
	<div data-bbox="565 184 1002 443"> </div> <p>i. Delete the WebPart1 folder from the project. You will create a new Web Part with the correct name in the next step.</p> <p><i>Note: While you can rename the WebPart1 folder and the files within it, is quicker and simpler to delete and recreate the Web Part.</i></p> <p><i>Note: When using VSeWSS a number of templates are installed. Most of these are created from the Blank project template but have then had a number of pre-configured project items added by default. The default files could be customized however for the process of learning it is a good idea to know how to add project items. This illustrates the process of adding a Web Part item to a Site Definition for instance.</i></p> <p>j. In the Solution Explorer right click the newly created HelloWorldWebPart project and select Add New Item.</p> <p>k. In the Categories area of the Add New Item dialogue box select SharePoint and in the Templates area select Web Part.</p> <p>l. Name this Web Part HelloWorld and select Add.</p> <div data-bbox="557 1016 1406 1530"> </div> <p><i>Note: The HelloWorld folder that has been added to the project. This folder contains three files with the base name of HelloWorld.</i></p> <p>m. Right click the HelloWorldWebPart project and select Properties. The Project Properties will be displayed.</p> <p>n. Select the Debug tab.</p> <p>o. Set the Start URL to http://spvm. This is used by VSeWSS to determine the location of SharePoint when deploying the solution.</p>

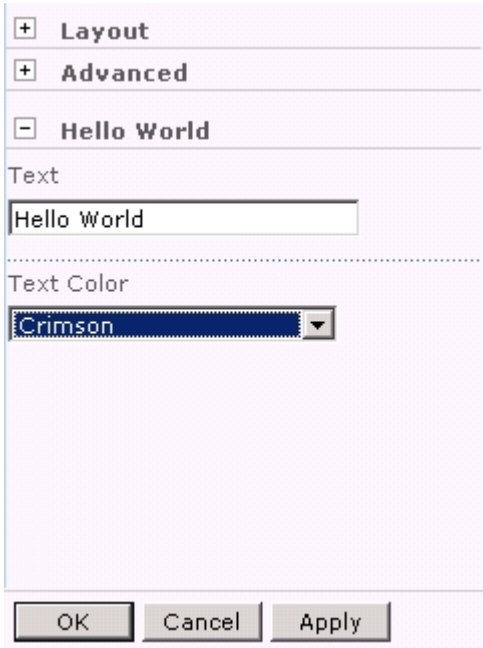
Tasks	Detailed Steps
	
2. Add Web Part Customizations	<p>Note: This task will describe the steps to add code to the Web Part. Properties of the Web Part will be created, including settings which can be changed using the SharePoint browser-based interface.</p> <p>a. Open HelloWorld.webpart. This XML file contains the Web Part definition that will be deployed to SharePoint.</p> <pre> 1 <?xml version="1.0" encoding="utf-8"?> 2 <webParts> 3 <webPart xmlns="http://schemas.microsoft.com/WebPart/v3"> 4 <metadata> 5 <!-- 6 The following Guid is used as a reference to the web part class, 7 and it will be automatically replaced with actual type name at deployment time. 8 --> 9 <type name="961e5345-38c5-4dc5-b421-81d55aef5dc4" /> 10 <importErrorMessage>Cannot import HelloWorld Web Part.</importErrorMessage> 11 </metadata> 12 <data> 13 <properties> 14 <property name="Title" type="string">HelloWorld Web Part</property> 15 <property name="Description" type="string">HelloWorld Description</property> 16 </properties> 17 </data> 18 </webPart> 19 </webParts> </pre> <p>b. Change the Description property to “A simple Hello World Web Part”.</p> <p>c. Open the HelloWorld.cs file in Visual Studio and inspect the code. The CreateChildControls method contains commented code which implements a simple way to add a label control to the Web Part.</p> <p>d. Uncomment the default code in the CreateChildControls method. The method should now resemble the following code.</p> <pre> protected override void CreateChildControls() { base.CreateChildControls(); // TODO: add custom rendering code here. Label label = new Label(); label.Text = "Hello World"; this.Controls.Add(label); } </pre> <p>Note: CreateChildControls is an overridden method. HelloWorldWebPart requires this method to provide a specific implementation in which controls are created.</p> <p>e. The next step is to add properties to the Web Part. Create a new property to allow the user to define the “Hello World” message to display. Paste the following code into the Web Part class.</p> <pre> private string _helloWorldText = "Hello SharePoint!"; public string HelloWorldText </pre>

Tasks	Detailed Steps												
	<pre>{ get { return _helloWorldText; } set { _helloWorldText = value; } }</pre> <p>f. The Web Part properties must be decorated with a few attributes. Add the following using statement to the top of <i>HelloWorld.cs</i>.</p> <pre>using System.ComponentModel;</pre> <p>g. In order for SharePoint to display the HelloWorldText property for user modification, you must add the following attributes to the property.</p> <pre>[WebBrowsable(true), Personalizable(PersonalizationScope.User), WebDescription("Hello World Text"), Category("Hello World"), WebDisplayName("Text")] public string HelloWorldText { get { return helloWorldText; } set { helloWorldText = value; } }</pre> <table border="1"> <thead> <tr> <th colspan="2">Table 1 – HelloWorldText property explanations</th></tr> </thead> <tbody> <tr> <td>WebDisplayName("...")</td><td>This attribute defines the text that is used to label the property in the Web Part task pane.</td></tr> <tr> <td>WebBrowsable(true)</td><td>This is used to allow Editing of the Web Part property. Without this the property will not be displayed in the Web Part task pane.</td></tr> <tr> <td>Personalizable(PersonalizationScope.User)</td><td>This attribute should be coupled with WebBrowsable as it allows saving of modified property values.</td></tr> <tr> <td>WebDescription("...")</td><td>This is an optional attribute and can contain anything you define. The description is displayed as a tooltip when hovering over the property in the Web Part task pane.</td></tr> <tr> <td>Category("...")</td><td>This optional attribute is an organizing mechanism, defining where the property should reside in the Web Part task pane of SharePoint and also providing a grouping strategy for logically related properties. The default category is <i>Miscellaneous</i>.</td></tr> </tbody> </table> <p>h. Next, a property should be added to allow the user to select a color for the message. This will follow a similar process as the steps above.</p> <p>First, add a reference to System.Drawing. Right click the HelloWorldWebPart project and select Add Reference.</p> <p>i. Select System.Drawing from the .NET tab.</p>	Table 1 – HelloWorldText property explanations		WebDisplayName("...")	This attribute defines the text that is used to label the property in the Web Part task pane.	WebBrowsable(true)	This is used to allow Editing of the Web Part property. Without this the property will not be displayed in the Web Part task pane.	Personalizable(PersonalizationScope.User)	This attribute should be coupled with WebBrowsable as it allows saving of modified property values.	WebDescription("...")	This is an optional attribute and can contain anything you define. The description is displayed as a tooltip when hovering over the property in the Web Part task pane.	Category("...")	This optional attribute is an organizing mechanism, defining where the property should reside in the Web Part task pane of SharePoint and also providing a grouping strategy for logically related properties. The default category is <i>Miscellaneous</i> .
Table 1 – HelloWorldText property explanations													
WebDisplayName("...")	This attribute defines the text that is used to label the property in the Web Part task pane.												
WebBrowsable(true)	This is used to allow Editing of the Web Part property. Without this the property will not be displayed in the Web Part task pane.												
Personalizable(PersonalizationScope.User)	This attribute should be coupled with WebBrowsable as it allows saving of modified property values.												
WebDescription("...")	This is an optional attribute and can contain anything you define. The description is displayed as a tooltip when hovering over the property in the Web Part task pane.												
Category("...")	This optional attribute is an organizing mechanism, defining where the property should reside in the Web Part task pane of SharePoint and also providing a grouping strategy for logically related properties. The default category is <i>Miscellaneous</i> .												

Tasks	Detailed Steps
	<div data-bbox="557 184 1490 940">  <p>The screenshot shows the 'Add Reference' dialog box with the 'Component Name' tab active. The list contains various .NET assemblies. 'System.Drawing' is selected, showing version 2.0.0.0 and runtime v2.0.50727.</p> </div> <p>j. Click OK.</p> <p>k. Add the following using statement to the top of HelloWorld.cs.</p> <pre>using System.Drawing;</pre> <p>l. Insert the TextColor property using the following code.</p> <pre>private KnownColor _textColor = KnownColor.Black; [WebBrowsable(true), Personalizable(PersonalizationScope.User), WebDescription("Hello World Text Color"), Category("Hello World"), WebDisplayName("Text Color")] public KnownColor TextColor { get { return _textColor; } set { _textColor = value; } }</pre> <p>Note: KnownColor is an enumeration that contains all the colors of the .NET rainbow. An enumeration will provide a set of choices in the form of a drop down list for editing the Web Part in the SharePoint Web Part task pane.</p> <p>m. The next step is to edit the code within the CreateChildControls method. Before setting the label.Text you should ensure the property contains some text. Add the following code before setting the label.Text value.</p> <pre>if (string.IsNullOrEmpty(HelloWorldText)) {</pre>

Tasks	Detailed Steps
	<pre> HelloWorldText = "Hello SharePoint!"; } </pre> <p>n. Delete the default code that sets the Label Text.</p> <pre> label.Text = "Hello World"; </pre> <p>o. Now set the label variables Text property to <i>HelloWorldText</i> and set the label's ForeColor property to the <i>TextColor</i> property.</p> <pre> label.Text = HelloWorldText; label.ForeColor = Color.FromKnownColor(TextColor); </pre> <p>Note: You need to convert the KnownColor enum value to a <i>Color</i> value.</p> <p>p. The final CreateChildControls method should look like the following.</p> <pre> protected override void CreateChildControls() { base.CreateChildControls(); Label label = new Label(); if (string.IsNullOrEmpty(HelloWorldText)) { HelloWorldText = "Hello SharePoint!"; } label.Text = HelloWorldText; label.ForeColor = Color.FromKnownColor(TextColor); this.Controls.Add(label); } </pre>
3. Deploy and Test	<p>Note: With the Web Part coding complete, this section will deploy it to the SharePoint site. Then you will test it to ensure that it behaves correctly.</p> <ol style="list-style-type: none"> Build and deploy the project to SharePoint by right clicking the HelloWorldWebPart project and selecting Deploy. The Visual Studio status bar should display Deploy succeeded. Open Internet Explorer and navigate to the SharePoint home page, http://spvm. Select Site Actions Edit Page located at the top right corner of the page.  <ol style="list-style-type: none"> Click Add a Web Part at the top of the Left Web Part zone.

Tasks	Detailed Steps
	<p>f. Scroll to All Web Parts Miscellaneous HelloWorld Web Part and tick the check box.</p>  <p>g. Click Add.</p> <p>h. The HelloWorld Web Part will be added to the page displaying the default message Hello SharePoint!</p>  <p>i. To edit the Web Part select the edit button on the newly added Web Part. Select Modify Shared Web Part.</p>  <p>j. In the Web Part task pane which appears, expand the Hello World section.</p> <p>k. In the Text box enter the desired text, and select a desired color in Text Color.</p> <p><i>Note: The Hello World editing section was defined previously when the Category attribute was added to the properties.</i></p>

Tasks	Detailed Steps
	<div data-bbox="557 226 1036 869">  </div> <p data-bbox="500 900 1211 936">I. Click OK. The Web Part will now display the modifications.</p>


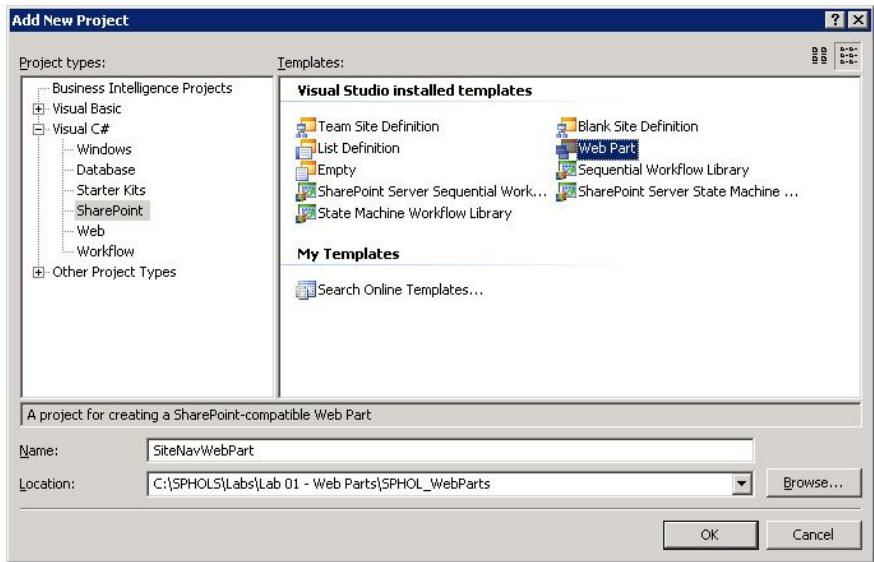

Exercise 2

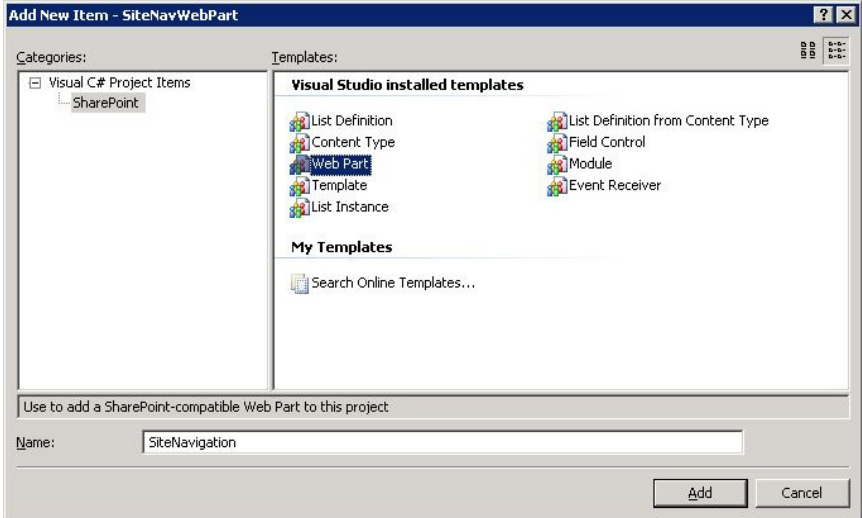
Web Part Interaction

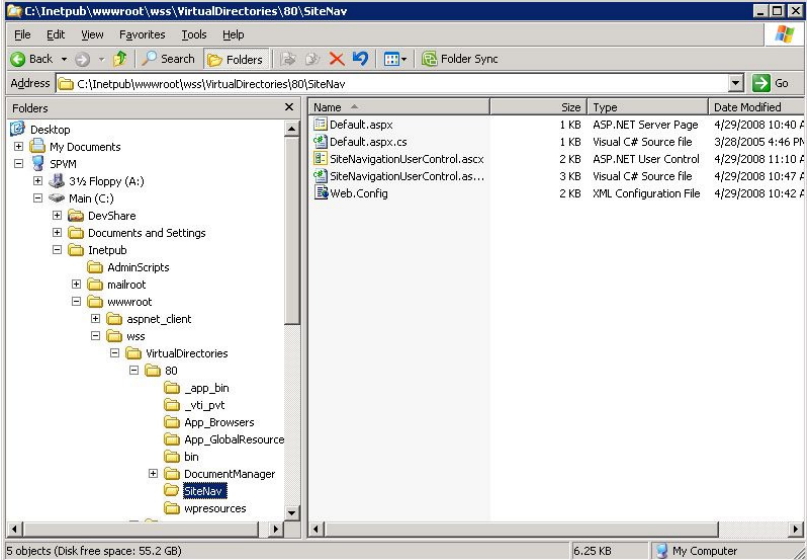
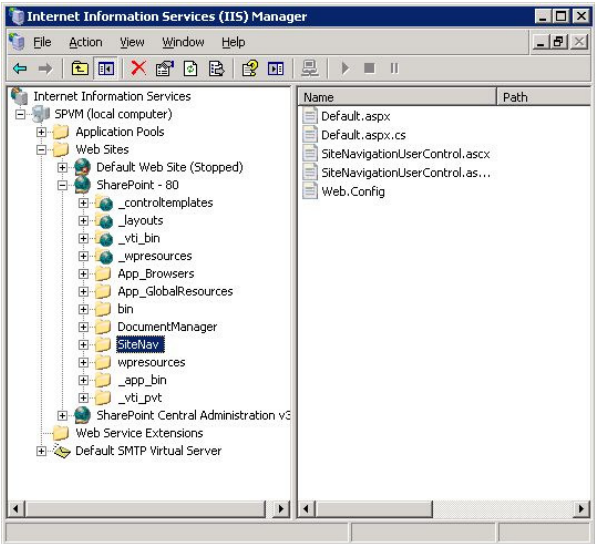
Scenario

Simple Web Parts that display data are very easy to create. However, it is more likely that you will want to enable some interaction with users. This usually requires a postback to your Web Part.

For this exercise you will create a Web Part that allows users to navigate to sites and lists within a site collection. This will utilize the **SPGridView** control. The grid view will display two columns – one for sites the other for lists. Each column will contain **ButtonField** controls that perform a postback when clicked. This exercise will also demonstrate how to use existing ASP.NET user controls in SharePoint.

Tasks	Detailed Steps
<p>Complete the following tasks on:</p> <p> Server1</p> <p>1. Set up the Project</p>	<p>a. In Visual Studio 2005 open the SPHOL_WebPart solution you created in Exercise 1 above.</p> <p>b. In the Solution Explorer right click the solution and select Add New Project.</p> <p>c. From the SharePoint project type, select Web Part.</p> <p>d. Name the Web Part project SiteNavWebPart.</p>  <p>e. Click OK.</p> <p>f. Right click the SiteNavWebPart project and select Properties. The Project Properties will be displayed.</p> <p>g. Select the Debug tab.</p> <p>h. Set the Start URL to http://spvm/. This is used by VSeWSS to determine the location of SharePoint when deploying the solution.</p> 

Tasks	Detailed Steps
	<p>i. Once again, Visual Studio has created a default Web Part in the new project called WebPart1. Delete the WebPart1 folder from the new project.</p> <p>j. Add a correctly named Web Part to the project by right clicking on SiteNavWebPart and selecting Add New Item.</p> <p>k. From the categories area select SharePoint and then from the Templates section select Web Part.</p> <p>l. Enter the name SiteNavigation.</p>  <p>m. Click Add.</p> <p>n. Open SiteNavigation.webpart and change the Description property to “A Site navigation Web Part that displays sites and lists”.</p> <p>o. Build the solution but don’t deploy it yet.</p>
<p>2. Setting up the user control</p>	<p>Note: In this task you will use an existing ASP.NET user control to display the sites and lists within a Web Part. User controls have some advantages over Web Part:</p> <ul style="list-style-type: none"> Visual Studio provides a visual designer for user controls, but not Web Parts. User Control .ASCX files can be altered without the need to recompile the .NET code behind. User Controls can be dynamically loaded to a Controls collection (as can Server controls and Web Parts). Creation of a user control for use in SharePoint is possible in two main ways: Create the user control in a Web Site and migrate the .ASCX file and code behind to the SharePoint solution. Create the user control and code behind in the SharePoint solution. <p>The second option has the drawback of not being able to test the user control without deploying it to a SharePoint site.</p> <p>This task will use a pre-created Web Site project containing a single user control. This simple site contains a single page, default.aspx hosting a single user control SiteNavigationUserControl.ascx that uses an SPGridView control.</p> <p>a. Copy the SiteNav folder from the resources folder “C:\SPHOLS\Labs\Lab 01 – Web Parts\Resources\CS” to the base folder for the local SharePoint site (http://spvm)</p>

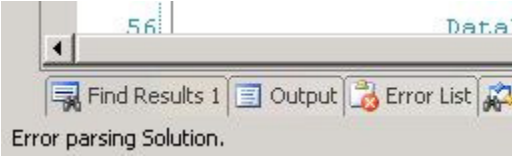
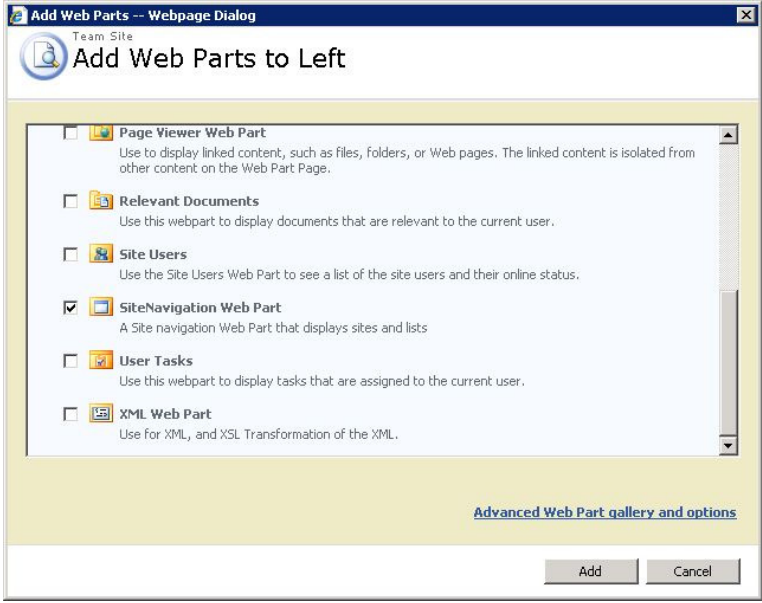
Tasks	Detailed Steps
	<p data-bbox="553 191 1122 222">“C:\inetpub\wwwroot\wss\VirtualDirectories\80”.</p>  <p data-bbox="506 852 1458 915">b. The site will need to function as an application in IIS. Select Start Administrative Tools Internet Information Services (IIS) Manager</p> <p data-bbox="506 930 1354 961">c. Navigate to Web Sites SharePoint – 80 and locate the SiteNav directory.</p>  <p data-bbox="506 1579 1013 1610">d. Right-Click SiteNav and select Properties</p> <p data-bbox="506 1625 1510 1688">e. In the Directory tab select Create for the Application Name. The outcome looks like the screenshot below.</p>


Tasks	Detailed Steps
	<div data-bbox="557 189 1177 766"> </div> <p>f. Click Ok.</p> <p>g. Close IIS Manager.</p> <p>Note: You should now be able to test the SiteNav Web site.</p> <p>h. Using Internet Explorer, navigate to http://spvm/SiteNav/default.aspx. You should see something similar to the following.</p> <div data-bbox="537 991 1507 1776"> </div> <p>Note: This simple site is able to display SharePoint content because it is located in a folder within the SharePoint site. The SPGridView control that is embedded in the user control,</p>

Tasks	Detailed Steps
	<p><i>SiteNavigationUserControl.ascx, is aware of the SharePoint context. Sites running outside of the SharePoint context cannot use SharePoint server controls such as SPGridView.</i></p> <ol style="list-style-type: none"> i. Return to Visual Studio. In the Solution Explorer right-click the Solution Folder and select Add Existing Web Site. j. Select SiteNav from C:\InetPub\wwwroot\wss\VirtualDirectories\80. <div data-bbox="555 426 1469 1150" data-label="Image"> </div> <ol style="list-style-type: none"> k. Click Open. l. Right-click the SiteNavWebPart project and select Add New Folder. m. Name the new folder Templates. <div data-bbox="643 1354 1404 1635" data-label="Text" style="border: 1px solid #ccc; padding: 10px; background-color: #e6f2ff;"> <p>Templates</p> <p>Templates are a simple way of copying files to the SharePoint 12 folder, unlike modules which are SharePoint Features that can be activated and deactivated. As such you should only use Templates to copy files that need to be globally visible to all sites. User Controls and Themes fall into this category.</p> <p>In VSeWSS the Templates folder maps directory to the TEMPLATE folder in the SharePoint 12 folder.</p> </div> n. Within the Templates folder create another folder called CONTROLTEMPLATES. o. Copy the SiteNavigationUserControl.ascx file from the newly added SiteNav Web site folder and paste it into the CONTROLTEMPLATES folder. <p>Visual Studio will copy the .ASCX and .CS files.</p> <ol style="list-style-type: none"> p. Move the SiteNavigationUserControl.ascx.cs file from the CONTROLTEMPLATE folder to the SiteNavigation folder. The outcome should look like as follows.

Tasks	Detailed Steps
	<div data-bbox="560 226 1068 583"> </div> <p>q. Open the SiteNavigationUserControl.ascx file. There are attributes within the @ Control directive that need to be modified.</p> <p>Remove the attribute value <code>CodeFile="SiteNavigationUserControl.ascx.cs"</code>.</p> <p>r. Replace the Inherits attribute with the following code – <u>keeping it all on a single line</u>.</p> <pre>Inherits="SiteNavWebPart.SiteNavigationUserControl, SiteNavWebPart, Version=1.0.0.0, Culture=neutral, PublicKeyToken=9f4da00116c38ec5"</pre> <p><i>Note: This specifies the code-behind class that the control inherits. SharePoint requires all code behind to be installed to the Global Assembly Cache (GAC). As the SiteNavWebPart project assembly is already being installed to the GAC this is a logical location for the code. However, there is nothing preventing you from creating a new separate Strongly Names assembly for the code behind.</i></p> <p>s. The PublicKeyToken value must be updated.</p> <p>To retrieve the PublicKeyToken of your assembly follow these sub-steps:</p> <ul style="list-style-type: none"> • Build the project. • Open an instance of the Visual Studio Command Prompt. This is found under Start All Programs Visual Studio 2005 Visual Studio Tools menu option or use the shortcut in the Links folder on the Desktop • Change to the bin\debug directory of your project, "C:\SPHOLS\Labs\Lab 01 - Web Parts\SPHOL_WebParts\SiteNavWebPart\bin\Debug". • Type SN.exe -Tp SiteNavWebPart.dll. • Copy the Public key token value – not the long Public key value. <p>Use the correct value to replace the PublicKeyToken in the ASCX file.</p> <div data-bbox="646 1528 1404 1732" style="border: 1px solid black; padding: 5px;"> <p>PublicKeyTokens from VSeWSS Project Templates</p> <p>When using VSeWSS to create projects, the Strong Name Key file (.SNK) that is used is part of the project template distributed by Microsoft. Therefore, the PublicKeyToken in this SNK file is the same for all projects and all users of VSeWSS. You should <u>always</u> replace this SNK file with one of your own.</p> </div> <p>t. Open the SiteNavigationUserControl.ascx.cs code file that is located in the SiteNavigation folder.</p> <p>u. Surround the SiteNavigationUserControl class with a namespace SiteNavWebPart. This makes the user control code more consistent with the rest of the Web Part code.</p>

Tasks	Detailed Steps
	<p>namespace SiteNavWebPart</p> <pre>{ public partial class SiteNavigationUserControl : System.Web.UI.UserControl { ... } }</pre> <p>v. Inside the SiteNavigationUserControl class, add a declaration for the SPGridView control that is embedded in the ASCX file.</p> <p>This is necessary because the control was created in a file based web site where Visual Studio does not create a designer file containing these declarations.</p> <pre>public partial class SiteNavigationUserControl : System.Web.UI.UserControl { protected SPGridView SiteNavGridView; ... }</pre> <p>w. Open SiteNavigation.cs and add the following code to the CreateChildControls method. This will dynamically load the user control to the Web Parts Controls collection.</p> <pre>protected override void CreateChildControls() { base.CreateChildControls(); UserControl uc = Page.LoadControl("~/_controltemplates/SiteNavigationUserCont rol.ascx") as UserControl; Controls.Add(uc); }</pre> <p>x. The user control uses a DataTable so you must add a project reference for System.Data.</p> <p>Right-click the SiteNavigationWebPart project and select Add Reference.</p> <p>y.  ab.</p> <p>z. </p> <p>aa.  Point solution by selecting View Other Windows </p> <p>bb.  View toolbar. You should see the following.</p> <p></p> <p>Note: If the WSP View fails to refresh, you may notice an error displayed in the Visual Studio</p>

Tasks	Detailed Steps
	<p><i>status bar.</i></p>  <p>Note: Unfortunately there is a small defect in VSeWSS that prevents it from generating the manifest if a file based web site is included in the solution.</p> <ul style="list-style-type: none"> cc. Remove the SiteNav website from the solution. dd. Switch to the WSP View and Refresh again. ee. Open and examine the SiteNavWebPart manifest.xml file. Note that the manifest now includes a TemplateFiles element to copy the .ASCX file to the CONTROLTEMPLATES folder. ff. Save and Build your project.
<p>3. Deploy and Test</p>	<ul style="list-style-type: none"> a. In Solution Explorer right click the SiteNavWebPart project and select Deploy. b. Open Internet Explorer and type in http://spvm. c. Select Site Actions Edit Page from the top right corner of the page. d. Click Add a Web Part buttons in the Left Web Part zone. e. Scroll to All Web Parts Miscellaneous SiteNavigationWebPart Web Part and tick the check box.  <ul style="list-style-type: none"> f. Click Add. g. Click Exit Edit Mode and refresh the Home page. h. Check that the final display is rendered similar to the Image below.


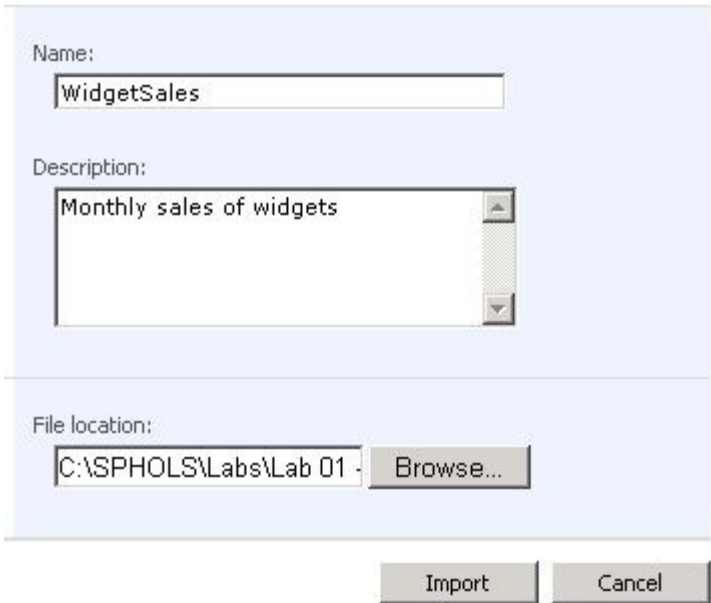
Tasks	Detailed Steps
	<p>SharePoint Hands On Labs</p> <p>SiteNavigation Web Part ▼</p>  <p>i. Optionally you can create one or more sites under the top level site (http://spvm) by selecting Site Actions Create, then Web Pages Sites and Workspaces. These will appear in the Site list of the Web Part.</p> <p>j. Test that clicking a link navigates to the correct location in the site.</p>

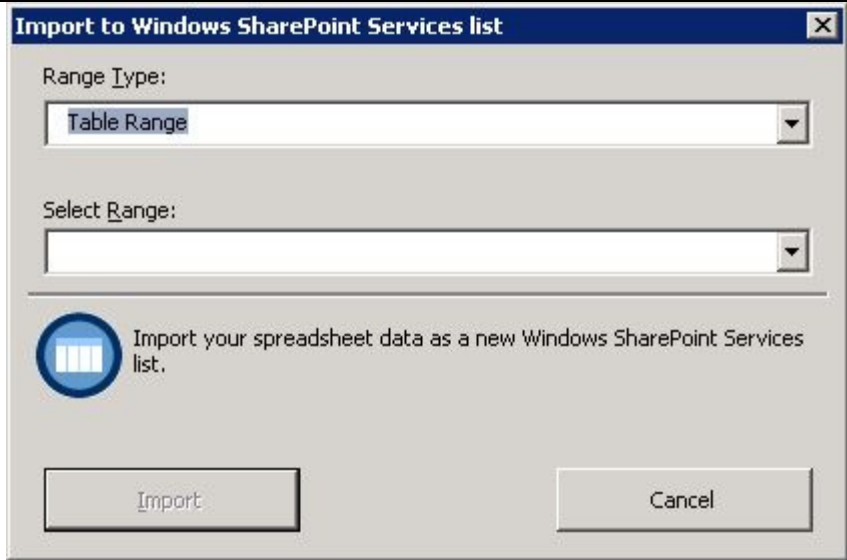

Exercise 3

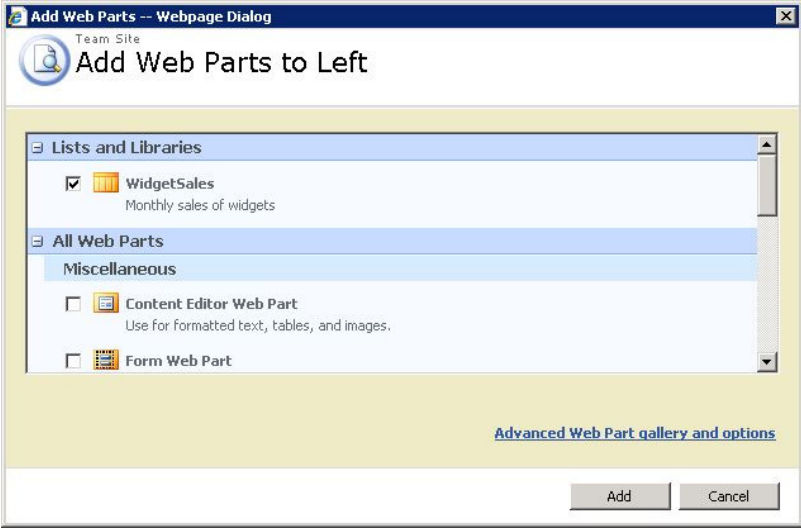
Connecting Web Parts

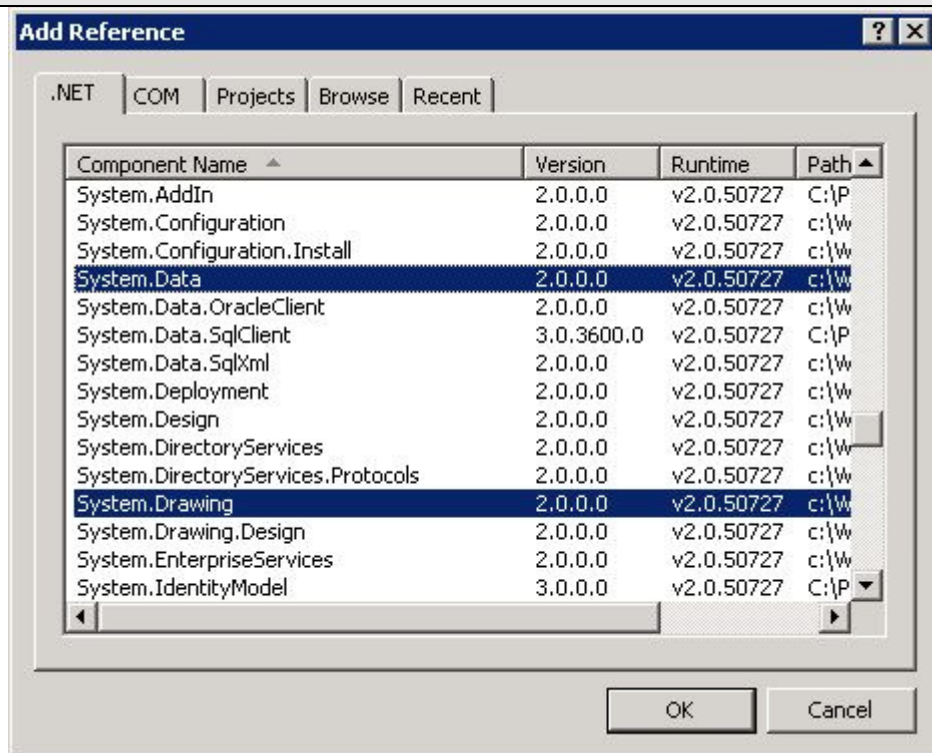
Scenario

In this exercise you will build a Dashboard Web Part that summarizes sales of widgets. Monthly sales records will be recorded to a custom list. The Dashboard Web Part will connect to a standard ListView Web Part on the same page.

Tasks	Detailed Steps
<p>Complete the following tasks on:</p> <p> Server1</p> <p>1. Creating the Data List</p>	<p><i>Note: While it is possible to create a list definition in code using VSeWSS, this is a very complex task and is not something that you would normally do. Creating lists and views in the SharePoint browser-based interface is much simpler. The completed list can be exported as a list template which can be easily provisioned into a site as part of a Feature or manually by a site administrator.</i></p> <ol style="list-style-type: none"> Start Internet Explorer and navigate to the local SharePoint site, http://spvm. From the Site Actions menu select Create. From the Custom Lists column select Import Spreadsheet. Enter the Name WidgetSales. Enter a Description Monthly sales of widgets and the filename C:\SPHOLS\Labs\Lab 01 - Web Parts\Resources\WidgetSales.xlsx.  <ol style="list-style-type: none"> Click Import. After a few moments Microsoft Excel will start and will display the spreadsheet with the following dialog.

Tasks	Detailed Steps																														
	<div></div> <div><p>h. Select the only range from the Select Range drop down list.</p><p>i. Click Import.</p><p>j. After a few seconds SharePoint will display the new list.</p></div> <div><table><thead><tr><th>WidgetCode</th><th>UnitsSold</th><th>DateSold</th></tr></thead><tbody><tr><td>GREENLEFT <small>NEW</small></td><td>624</td><td>1/1/2007</td></tr><tr><td>GREENLEFT <small>NEW</small></td><td>968</td><td>2/1/2007</td></tr><tr><td>GREENLEFT <small>NEW</small></td><td>846</td><td>3/1/2007</td></tr><tr><td>GREENLEFT <small>NEW</small></td><td>890</td><td>4/1/2007</td></tr><tr><td>GREENLEFT <small>NEW</small></td><td>853</td><td>5/1/2007</td></tr><tr><td>GREENLEFT <small>NEW</small></td><td>739</td><td>6/1/2007</td></tr><tr><td>GREENLEFT <small>NEW</small></td><td>675</td><td>7/1/2007</td></tr><tr><td>GREENLEFT <small>NEW</small></td><td>640</td><td>8/1/2007</td></tr><tr><td>GREENLEFT <small>NEW</small></td><td>568</td><td>9/1/2007</td></tr></tbody></table></div> <div><p>k. Return to the home page for the site.</p><p>l. From the Site Actions menu select Edit Page.</p><p>m. Click Add Web Part on the Left Web Part zone.</p><p>n. Select WidgetSales from the Libraries and Lists section of the Add Web Parts selection dialog.</p></div>	WidgetCode	UnitsSold	DateSold	GREENLEFT <small>NEW</small>	624	1/1/2007	GREENLEFT <small>NEW</small>	968	2/1/2007	GREENLEFT <small>NEW</small>	846	3/1/2007	GREENLEFT <small>NEW</small>	890	4/1/2007	GREENLEFT <small>NEW</small>	853	5/1/2007	GREENLEFT <small>NEW</small>	739	6/1/2007	GREENLEFT <small>NEW</small>	675	7/1/2007	GREENLEFT <small>NEW</small>	640	8/1/2007	GREENLEFT <small>NEW</small>	568	9/1/2007
WidgetCode	UnitsSold	DateSold																													
GREENLEFT <small>NEW</small>	624	1/1/2007																													
GREENLEFT <small>NEW</small>	968	2/1/2007																													
GREENLEFT <small>NEW</small>	846	3/1/2007																													
GREENLEFT <small>NEW</small>	890	4/1/2007																													
GREENLEFT <small>NEW</small>	853	5/1/2007																													
GREENLEFT <small>NEW</small>	739	6/1/2007																													
GREENLEFT <small>NEW</small>	675	7/1/2007																													
GREENLEFT <small>NEW</small>	640	8/1/2007																													
GREENLEFT <small>NEW</small>	568	9/1/2007																													

Tasks	Detailed Steps
	 <p>o. Click Add. You will now see the list of Widget Sales displayed on the home page.</p>
<p>2. Create the Dashboard</p>	<p><i>Note: Now that you have some data to summarize you can go ahead and create a dashboard Web Part. The Web Part will total sales of each type of widget for a given month. It will then display the total sales colored red if the sales are less than budgeted or green if they exceed the budget.</i></p> <p><i>This Web Part will utilize a DropDownList to display a list of months and a SPGridView to display the sales data.</i></p> <p><i>In Visual Studio 2005 open the SPHOL_WebParts solution you created in previous exercises.</i></p> <p>a. Right click the solution in the Solution Explorer and select the Add New Project option. Visual Studio will display the New Project dialog window.</p> <p>b. Expand the Visual C# item and select SharePoint. From the Templates section select Empty.</p> <p>c. Name the new Web Part project DashboardWebPart.</p> <p>d. Click OK.</p> <p><i>Note: You will now have an empty project into which you can add any type of project item. The Empty project template provided by VSeWSS is exactly that – empty! You can add Web Parts and other types of features to this project and VSeWSS will generate the correct manifest and feature elements that are required.</i></p> <p>e. Right click the new project and select Add New Item.</p> <p>f. From the SharePoint category select Web Part.</p> <p>g. Enter the name Dashboard and click Add.</p> <p>h. In the Solution Explorer, under the DashboardWebPart project, right click References and select Add Reference.</p> <p>i. Select System.Data and System.Drawing from the options available from the .NET tab.</p>

Tasks	Detailed Steps
	 <p>The screenshot shows the 'Add Reference' dialog box with the 'Component Name' tab selected. The list of components includes System.AddIn, System.Configuration, System.Configuration.Install, System.Data (highlighted), System.Data.OracleClient, System.Data.SqlClient, System.Data.SqlXml, System.Deployment, System.Design, System.DirectoryServices, System.DirectoryServices.Protocols, System.Drawing (highlighted), System.Drawing.Design, System.EnterpriseServices, and System.IdentityModel. The 'Path' column shows various paths like C:\P and c:\W.</p> <p>j. Click OK.</p> <p>k. Under the new Dashboard folder select and open the Dashboard.webpart file.</p> <p>l. Change the Description property to “Summarize monthly widget sales”.</p> <pre><?xml version="1.0" encoding="utf-8"?> <webParts> <webPart xmlns="http://schemas.microsoft.com/WebPart/v3"> <metadata> <!-- The following Guid is used as a reference to the web part class, and it will be automatically replaced with actual type name at deployment time. --> <type name="c7fec753-a901-416c-89c4-865491ebe53f" /> <importErrorMessage>Cannot import Dashboard Web Part.</importErrorMessage> </metadata> <data> <properties> <property name="Title" type="string">Dashboard Web Part</property> <property name="Description" type="string">Summarize monthly widget sales</property> </properties> </data> </webPart> </webParts></pre> <p>m. Open Dashboard.cs.</p> <p>n. Add the following namespace references to the Dashboard.cs file.</p> <pre>using System.Collections; using System.ComponentModel; using System.Data; using System.Drawing;</pre> <p>o. Create properties to specify the maximum and minimum budgeted sales. Add the following code to the Dashboard.cs class file.</p> <pre>private int _budgetedMinimum;</pre>

Tasks	Detailed Steps
	<pre data-bbox="553 222 1446 968">[WebDisplayName("Budgeted Sales Minimum"), WebBrowsable(true), Personalizable(PersonalizationScope.User), WebDescription("Budgeted Sales Minimum Value for Widget Sales"), Category("Budget Threshold")] public int BudgetedMinimum { get { return _budgetedMinimum; } set { _budgetedMinimum = value; } } private int _budgetedMaximum; [WebDisplayName("Budgeted Sales Maximum"), WebBrowsable(true), Personalizable(PersonalizationScope.User), WebDescription("Budgeted Sales Maximum Value for Widget Sales"), Category("Budget Threshold")] public int BudgetedMaximum { get { return _budgetedMaximum; } set { _budgetedMaximum = value; } }</pre> <p data-bbox="508 1016 1495 1073">p. The next step is to implement the methods required for the connection to work. Add the following fields to the Dashboard.cs source file.</p> <pre data-bbox="553 1094 1060 1150">private IWebPartTable _provider; private ICollection _tableData;</pre> <p data-bbox="508 1199 1203 1226">q. Paste the following code into your Dashboard.cs source file.</p> <pre data-bbox="553 1247 1490 1598">[ConnectionConsumer("Widget Sales Data")] public void GetConnectionInterface(IWebPartTable provider) { TableCallback callback = new TableCallback(ReceiveTable); _provider = provider; provider.GetTableData(callback); } public void ReceiveTable(object providerTable) { _tableData = providerTable as ICollection; }</pre> <p data-bbox="508 1650 1523 1833">Note: <i>GetConnectionInterface</i> is marked by an attribute ConnectionConsumer. This attribute is necessary for Web Parts to consume data from Web Parts. The single parameter to this method determines the type of provider that is expected. SharePoint's Web Part Manager looks for ConnectionCosumers and determines if there are any providers that can supply the required interface. The TableCallback delegate defines the method that is invoked transparently by the ConnectionProvider – in this case ReceiveTable.</p> <p data-bbox="508 1881 1523 1908">r. Add the following protected fields to the source code that will reference the dropdown list</p>

Tasks	Detailed Steps
	<p>and SPGridView controls.</p> <pre>protected DropDownList dateList = null; protected SPGridView gridView = null;</pre> <p>s. Update the CreateChildControls method by replacing it with the code below. This code creates the instances of the DropDownList and SPGridView controls, sets some control properties, sets the grid's data binding and adds the two controls to the Controls collection of the Web Part.</p> <pre>protected override void CreateChildControls() { base.CreateChildControls(); //Setup DropDownList settings dateList = new DropDownList(); dateList.AutoPostBack = true; //Statically add List Items dateList.Items.Add(new ListItem("Select Date")); dateList.Items.Add(new ListItem("January")); dateList.Items.Add(new ListItem("February")); dateList.Items.Add(new ListItem("March")); dateList.Items.Add(new ListItem("April")); dateList.Items.Add(new ListItem("May")); dateList.Items.Add(new ListItem("June")); dateList.Items.Add(new ListItem("July")); dateList.Items.Add(new ListItem("August")); dateList.Items.Add(new ListItem("September")); dateList.Items.Add(new ListItem("October")); dateList.Items.Add(new ListItem("November")); dateList.Items.Add(new ListItem("December")); //Set up SPGridView settings gridView = new SPGridView(); gridView.AutoGenerateColumns = false; //Add the columns to the grid SPBoundField fld = new SPBoundField(); fld.HeaderText = "Widget Code"; fld.DataField = "Widget Code"; gridView.Columns.Add(fld); fld = new SPBoundField(); fld.HeaderText = "Units Sold"; fld.DataField = "Units Sold"; fld.ItemStyle.HorizontalAlign = HorizontalAlign.Right; fld.ItemStyle.Width = new Unit(80); gridView.Columns.Add(fld); //Add EventHandlers gridView.RowDataBound += new GridViewRowEventHandler(gridView_RowDataBound); Controls.Add(dateList); Controls.Add(gridView); }</pre>

Tasks	Detailed Steps
	<p>t. Next you will need to implement the RowDataBound event handler. This event handler will be called for each row displayed in the SPGridView. It will set the color of the Units Sold column based on the BudgetedMinimum and BudgetedMaximum property values.</p> <p>Add the following new method into the Dashboard.cs class file.</p> <pre>void gridView_RowDataBound(object sender, GridViewRowEventArgs e) { DataRowView view = e.Row.DataItem as DataRowView; if (view != null) { int sales = Convert.ToInt32(view.Row["Units Sold"].ToString()); if (e.Row.RowType == DataControlRowType.DataRow) { if (sales > BudgetedMaximum) e.Row.Cells[1].BackColor = Color.LawnGreen; else if (sales < BudgetedMinimum) e.Row.Cells[1].BackColor = Color.Tomato; else e.Row.Cells[1].BackColor = Color.White; } } }</pre> <p>u. During the life cycle of the Web Part, the data connection is not established until the [ConsumerConnection] method is passed the provider and the provider has provided the data via the callback method ReceiveTable. In this instance this means CreateDataView cannot be called until ReceiveDataTable is invoked.</p> <p>Update the ReceiveDataTable method to add a call to CreateDataView.</p> <pre>public void ReceiveTable(object providerTable) { _tableData = providerTable as ICollection; CreateDataView(); }</pre> <p>v. Copy and paste the following code for the new CreateDataView method.</p> <p>CreateDataView filters the data by the selected month and binds it to the SPGridView.</p> <pre>private void CreateDataView() { if (_tableData != null && dateList != null && dateList.SelectedIndex > 0) { DataColumn column; DataRow row; DataTable data = new DataTable(); //Add Columns to the data table if (_tableData != null) { data.Columns.Clear(); column = new DataColumn(); column.DataType = typeof(string); column.ColumnName = "Widget Code";</pre>

Tasks	Detailed Steps
	<pre> column.Caption = "Widget Code"; data.Columns.Add(column); data.PrimaryKey = new DataColumn[] { column }; column = new DataColumn(); column.DataType = typeof(int); column.ColumnName = "Units Sold"; column.Caption = "Units Sold"; data.Columns.Add(column); } foreach (DataRowView rowView in _tableData) { // during import of the data the LinkTitle column is // used // for the first column of data in the spreadsheet // so you can't use WidgetCode to find the column string widgetCode = rowView.Row["LinkTitle"].ToString(); int unitsSold = Convert.ToInt32(rowView.Row["UnitsSold"].ToString()); DateTime dateSold = Convert.ToDateTime(rowView.Row["DateSold"].ToString()); if (dateSold.Month == dateList.SelectedIndex) { row = data.Rows.Find(widgetCode); if (row == null) { row = data.NewRow(); row["Widget Code"] = widgetCode; data.Rows.Add(row); } else { unitsSold += (int)row["Units Sold"]; } row["Units Sold"] = unitsSold; } } gridView.DataSource = data; gridView.DataBind(); } </pre> <p>w. Build the project by pressing Ctrl + Shift + B.</p> <p>x. The project should display Build Successful in the status bar.</p>
3. Deploy and Test	<p>a. Right click the DashboardWebPart project folder and select Deploy.</p> <p>b. Open Internet Explorer and browse to the home page, http://spvwm.</p> <p>c. Select Site Actions Edit Page.</p> <p>d. Click Add Web Part in the Left Web Part zone.</p> <p>e. Scroll to the All Web Parts Miscellaneous section and check DashBoard Web Part.</p>

Tasks	Detailed Steps
	<div data-bbox="552 226 1463 963" data-label="Image"> </div> <p>f. Click Add.</p> <p>g. Click Exit Edit Mode</p> <p>h. The Dashboard Web Part will be displayed.</p> <div data-bbox="548 1167 860 1197" data-label="Text"> <p>SharePoint Hands On Labs</p> </div> <div data-bbox="548 1209 834 1243" data-label="Section-Header"> <h3>Dashboard Web Part</h3> </div> <div data-bbox="552 1249 735 1285" data-label="Text"> <p>Select Date ▼</p> </div> <div data-bbox="496 1369 1505 1434" data-label="Text"> <p><i>Note: If you select a month nothing will be displayed as you have not yet connected the Web Part to a provider.</i></p> </div> <p>i. From the Web Part Edit Menu select Modify Shared Web Part.</p> <p>j. Again, from the Web Part Edit Menu select Connections Get Widget Sales Data From WidgetSales.</p> <div data-bbox="552 1562 1430 1728" data-label="Image"> </div> <p>k. In the Web Part task pane scroll down to the Budget Threshold section.</p> <p>l. Set the Budgeted Sales Minimum to 400 and Budgeted Sales Maximum to 700.</p>

Tasks	Detailed Steps										
	<div data-bbox="558 226 964 569"> </div> <p>m. Click Ok.</p> <p>n. Select a month from the list and the dashboard should display summarized data.</p> <div data-bbox="526 701 1414 1031"> <table border="1"> <thead> <tr> <th>Widget Code</th><th>Units Sold</th></tr> </thead> <tbody> <tr> <td>GREENLEFT</td><td>114</td></tr> <tr> <td>REDTOP</td><td>991</td></tr> <tr> <td>BLUEUPSIDE</td><td>168</td></tr> <tr> <td>BROWN17</td><td>110</td></tr> </tbody> </table> </div>	Widget Code	Units Sold	GREENLEFT	114	REDTOP	991	BLUEUPSIDE	168	BROWN17	110
Widget Code	Units Sold										
GREENLEFT	114										
REDTOP	991										
BLUEUPSIDE	168										
BROWN17	110										
<p>4. Creating a Custom Editor</p>	<p><i>Note: SharePoint makes it very simple to expose Web Part properties to a user for editing via the use of attributes. However, for full control of the behavior and appearance of property editing you can create a Custom Editor.</i></p> <p><i>A Custom Editor is a special Web Part that inherits from the base EditorPart class. Once associated with a Web Part, the editor appears in the Web Part task pane and can replace or suppliment the existing automatic Web Part property controls.</i></p> <p><i>In this task you will create a Custom Editor that will utilize JavaScript slider controls to replace the BudgetMinimum and BudgetMaximum text boxes.</i></p> <p>a. Right click the Dashboard folder in the DashboardWebPart project.</p> <p>b. Select Add Class ...</p> <p>c. Select Visual C# from the Categrory and Class from the Item Templates</p> <p>d. Enter DashboardEditor.cs for the file name.</p>										

Tasks	Detailed Steps
	<div data-bbox="557 191 1377 684"> </div> <p>e. Click Add.</p> <p>f. The new class file will be displayed. Visual Studio uses the folder name when creating new class files; this does not match the namespace for the other code files in the Web Part.</p> <p>Change the default namespace to DashboardWebPart.</p> <p>g. Add the following using statements to the top of the new class file. These are required for the controls used by the Web Part.</p> <pre>using System.Drawing; using System.Text; using System.Web.UI; using System.Web.UI.HtmlControls; using System.Web.UI.WebControls; using System.Web.UI.WebControls.WebParts; using Microsoft.SharePoint;</pre> <p>h. The new editor Web Part must inherit from EditorPart. Change the class declaration to the following.</p> <pre>public class DashboardEditor : EditorPart</pre> <p>i. The Dashboard Web Part has two properties – BudgetedMinimum and BudgetedMaximum. These will be represented on the Editor Web Part by slider controls that are implemented with HTML, some Javascript and hidden HTML Input controls. A Label control is also needed to display any validation error message.</p> <p>Add the following declarations to the top of the DashboardEditor class.</p> <pre>protected HtmlInputHidden budgetedMinimum; protected HtmlInputHidden budgetedMaximum; protected Label errorLabel;</pre> <p>j. As with other Web Parts, you must override CreateChildControls to update the base Controls collection.</p> <p>Add the following CreateChildControls method. The comments within the code explain the various steps.</p> <pre>protected override void CreateChildControls() { base.CreateChildControls();</pre>

Tasks	Detailed Steps
	<pre> // create the 2 controls - must set the id as it's needed later budgetedMinimum = new HtmlInputHidden(); budgetedMinimum.ID = "budMin"; budgetedMaximum = new HtmlInputHidden(); budgetedMaximum.ID = "budMax"; // lets have a label to display any errors errorLabel = new Label(); errorLabel.ForeColor = Color.Red; errorLabel.Text = string.Empty; // add the hidden controls to the part // have to add the controls to the collection // so we get the correct clientid which is used // in the slider javascript Controls.Add(budgetedMinimum); Controls.Add(budgetedMaximum); // add the style sheet reference to the header addStyleSheet(); // add the common javascript for the slider controls Page.ClientScript.RegisterClientScriptInclude("slider", "~/_layouts/slider/sliders.js"); // add some script to initialise the 2 sliders on the page Page.ClientScript.RegisterClientScriptBlock(typeof(Dashboard Editor), "initsliders", getSliderInit(new Control[] { budgetedMinimum, budgetedMaximum }, 0, 1000, 50), true); // add the slider control elements to the page Controls.Add(new LiteralControl(createSlider("Budget Minimum", budgetedMinimum.ClientID, 0))); Controls.Add(new LiteralControl(createSlider("Budget Maximum", budgetedMaximum.ClientID, 1))); // add the error label Controls.Add(errorLabel); // give the editor a title in the task pane Title = "Dashboard Editor"; } </pre> <p>k. There are a number of helper methods used by CreateChildControls. A description of these methods follows the code below. Paste all three methods to the DashboardEditor class file.</p> <pre> /// <summary> /// Add a stylesheet link to the page header </pre>

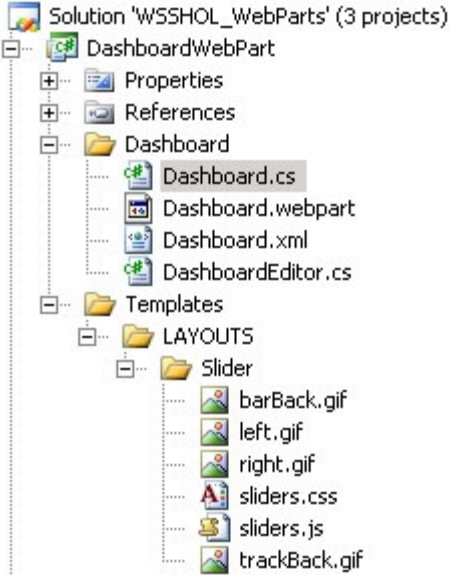


Tasks	Detailed Steps
	<pre> /// </summary> private void addStyleSheet() { HtmlHead header; header = Page.Header as HtmlHead; if (header != null) { HtmlLink link = new HtmlLink(); link.Attributes.Add("href", "~/_layouts/slider/sliders.css"); link.Attributes.Add("media", "screen"); link.Attributes.Add("rel", "stylesheet"); link.Attributes.Add("type", "text/css"); header.Controls.Add(link); } } /// <summary> /// Create some inline javascript to initialise the 2 sliders on the page /// </summary> /// <param name="controls"></param> /// <param name="min"></param> /// <param name="max"></param> /// <param name="step"></param> /// <returns></returns> private string getSliderInit(Control[] controls, int min, int max, int step) { StringBuilder initBlock = new StringBuilder(); initBlock.Append("var sliders = new Array();"); initBlock.Append("function initsliders() {"); for (int i = 0; i < controls.Length; i++) { initBlock.Append("sliders["); initBlock.Append(i); initBlock.Append("] = new Slider("); initBlock.Append(min); initBlock.Append(", "); initBlock.Append(max); initBlock.Append(", "); initBlock.Append(step); initBlock.Append(", document.getElementById('"); initBlock.Append(controls[i].ClientID); initBlock.Append("'), document.getElementById('innerslider"); initBlock.Append(i); initBlock.Append("'), document.getElementById('label"); initBlock.Append(i); initBlock.Append("'))"); } initBlock.Append("}"); initBlock.Append("window.onload = initsliders;"); return initBlock.ToString(); } </pre>

Tasks	Detailed Steps
	<pre> /// <summary> /// Create the DOM elements for each slider control /// </summary> /// <param name="label"></param> /// <param name="inputID"></param> /// <param name="sliderNumber"></param> /// <returns></returns> private string createSlider(string label, string inputID, int sliderNumber) { StringBuilder slider = new StringBuilder(); slider.Append(@"<div class='heading'>"); slider.Append(label + ":"); slider.Append(@"</div>"); slider.Append(@"<div class='progress'>"); slider.Append(@""); slider.Append(@""); slider.Append(@""); slider.Append(@""); slider.Append(@""); slider.Append(@""); slider.Append(@""); slider.Append(@""); slider.Append(@""); slider.Append(@""); slider.Append(@""); slider.Append(@"</div>"); return slider.ToString(); } </pre> <p>addStyleSheet adds a <link> tag to the page header. The stylesheet, <i>sliders.css</i>, will be deployed to the _layouts fodler in a step further below.</p> <p>getSliderInit generates a string that contains some inline javascript for the page to initialize the sliders and connect them to the hidden HTML elements. The resulting javascript will look as follows.</p> <pre> var sliders = new Array(); function initsliders() { sliders[0] = new Slider(0, 1000, 50, </pre>

Tasks	Detailed Steps
	<pre> document.getElementById('ele1'), document.getElementById('innerslider1'), document.getElementById('label1')); sliders[1] = new Slider(0, 1000, 50, document.getElementById('ele2'), document.getElementById('innerslider2'), document.getElementById('label2')); } window.onload = initsliders; </pre> <p>createSlider generates a string containing the DOM elements that represent the slider control. Each slider control will look similar to the following.</p> <pre> <div class='heading'>Budget Minimum:</div> <div class='progress'> </div> </pre> <p>I. EditorPart Web Parts must override two methods to read and update the properties for the Web Part they are editing – SyncChanges is called when the EditorPart is displayed and ApplyChanges is called when the user clicks Ok or Apply in the task pane.</p> <p>Add the following two methods.</p> <pre> /// <summary> /// Update the editor part with values from the associated WebPart /// </summary> public override void SyncChanges() { EnsureChildControls(); // get reference to Web Part being edited Dashboard webPart = WebPartToEdit as Dashboard; // pull properties from Web Part budgetedMaximum.Value = webPart.BudgetedMaximum.ToString(); budgetedMinimum.Value = webPart.BudgetedMinimum.ToString(); } /// <summary> </pre>

Tasks	Detailed Steps
	<pre> /// Saves the values in an EditorPart control to the corresponding properties /// in the associated WebPart control. /// </summary> public override bool ApplyChanges() { int min, max; EnsureChildControls(); // get reference to Web Part being edited Dashboard webPart = WebPartToEdit as Dashboard; // validate and save the changes from the Toolpane min = Convert.ToInt32(budgetedMinimum.Value); max = Convert.ToInt32(budgetedMaximum.Value); if (min > max) { errorLabel.Text = "Minimum must be less than maximum!"; return false; } else { webPart.BudgetedMinimum = min; webPart.BudgetedMaximum = max; errorLabel.Text = string.Empty; } return true; } </pre> <p>m. Build the <i>DashboardWebPart</i> project to ensure there are no errors.</p>
<p>5. Connecting and Completing the Custom Editor</p>	<p>a. Open <i>Dashboard.cs</i> in the DashboardWebPart project.</p> <p>b. Add the following overridden CreateEditorParts method to create and connect and instace of the DashboardEditor to the Dashboard Web Part.</p> <pre> /// <summary> /// Create a DashboardEditor to allow editing of the budget min/max properties /// </summary> /// <returns></returns> public override EditorPartCollection CreateEditorParts() { List<EditorPart> editorParts = new List<EditorPart>(1); // create instsance of custom editor part & add to collection EditorPart editorPart = new DashboardEditor(); editorPart.ID = ID + "_EditorPart"; editorParts.Add(editorPart); return new EditorPartCollection(base.CreateEditorParts(), editorParts); } </pre> <p>c. A generic list is used in the new method so add the following to the using statements at the top of the class file.</p>

Tasks	Detailed Steps
	<pre>using System.Collections.Generic;</pre> <p>d. The DashboardEditor Web Part will replace the standard property editor in the task pane for the Budget Minimum and Budget Maximum values. Remove all the attributes from those two properties.</p> <pre>#[WebDisplayName("Budgeted Sales Minimum"), #WebBrowsable(true), #Personalizable(PersonalizationScope.User), #WebDescription("Budgeted Sales Minimum Value for Widget Sales"), #Category("Budget Threshold")] public int BudgetedMinimum { get { return _budgetedMinimum; } set { _budgetedMinimum = value; } }</pre> <pre>#[WebDisplayName("Budgeted Sales Maximum"), #WebBrowsable(true), #Personalizable(PersonalizationScope.User), #WebDescription("Budgeted Sales Maximum Value for Widget Sales"), #Category("Budget Threshold")] public int BudgetedMaximum { get { return _budgetedMaximum; } set { _budgetedMaximum = value; } }</pre> <p>e. Finally, you need to deploy a few files for the slider control in the DashboardEditor. Right click the DashboardWebPart project and select Add New Folder. Name the new folder Templates.</p> <p>f. Create a LAYOUTS folder in the templates folder.</p> <p>g. Open Windows Explorer and navigate to the lab Resources folder “C:\SPHOLS\Labs\ Lab 01 – Web Parts\Resources”.</p> <p>h. Select and Copy the Slider folder – copy the whole folder, NOT just it’s contents.</p> <p>i. Return to Visual Studio. Right click the new LAYOUTS folder and paste the copied Slider folder. Your project should now look as follows.</p>

Tasks	Detailed Steps
	
<p>6. Deploy and Test</p>	<ol style="list-style-type: none"> Right click the DashboardWebPart project folder and select Deploy. Open Internet Explorer and browse to the home page, http://spvm. Select Modify Share Web Part from the Dashboard Web Part Edit menu. The task pane will now display the new DashboardEditor Web Part.  <ol style="list-style-type: none"> Test the editor by setting new invalid Budget value using the slider's left and right arrows.  <ol style="list-style-type: none"> Click Apply. <p><i>Note: An error message will display at the top of the task pane in addition to the error label of the DashboardEditor Web Part.</i></p>

Tasks	Detailed Steps
	<div data-bbox="558 197 1036 527"><p>Cannot save the property settings for this Web Part. An error has occurred.</p><p>Dashboard Editor ⌵</p><p>Budget Minimum:</p><div data-bbox="570 359 898 405">< 800 ></div><p>Budget Maximum:</p><div data-bbox="570 449 898 495">< 400 ></div><p>Minimum must be less than maximum!</p></div> <p>f. Correct the setting and click Ok.</p>