## Developing Web Parts

**Lab Time:** 45 minutes

**Lab Folder**: C:\Student\Labs\WebParts

**Lab Overview:** In this lab you will begin using the Visual Studio 2010 SharePoint Tools. This will give you a chance to become familiar with the standard project structure used by the SharePoint Developer Tools. You will go through the steps of creating and testing a project that contains a Feature, a Feature Receiver and a Web Parts. Along the way you will learn to configure the SharePoint Developer Tools deployment options as well debug a SharePoint Developer Tools project by single-stepping through the code in your solution.

Lab Setup Requirements

* Before you begin this lab, you must run the batch file named **SetupLab.bat**. This batch file creates a new blank site collection at the location **http://intranet.wingtip.com/sites/WebParts**. This is the site you will use to test and debug the code you are going to write with the Visual Studio 2010 SharePoint Tools.

### Exercise 1: Creating, Deploying and Testing Web Parts

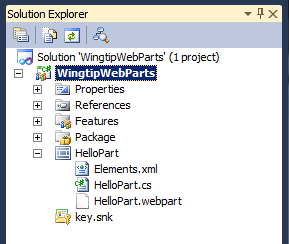
In this exercise you will create a simple Web Part and add it to a custom Web Part group with a custom icon.

1. Create a new Visual Studio project named **WingtipWebParts** based on the SharePoint Tools project of type **Empty SharePoint Project**. You can find this project type using the path **Visual C# » SharePoint » 2010**. Ensure the **.NET Framework 3.5** is selected instead of the default .NET Framework 4.0. Click **OK** to start the **SharePoint Customization Wizard**.
2. Complete the SharePoint Customization wizard that appears using the following information:

**Debugging site:** http://intranet.wingtip.com/sites/Webparts

**Deploy as a farm solution**: selected

1. Now you are going to add a few graphic images into your project so they are deployed inside the SharePoint **Images** folder. The first step to doing this correctly is to add a mapped folder to your project for the SharePoint **Images** folder. Accomplish this by right-clicking the **WingtipWebParts** project in the Solution Explorer, expanding the **Add** menu and selecting the **SharePoint "Images" Mapped Folder** option.
2. After completing the previous step you should be able to see that there is an **Images** directory inside your project. You should also be able to observe that there is a child folder inside the **Images** folder with the same name as the project **WingtipWebParts**. This demonstrates how the SPT helps to ensure best practices with SharePoint development. When adding image files to your project, do not add them directly inside the **Images** folder. Instead, add them to the inner directory in this case named **WingtipWebParts** to avoid file name conflicts with the image files that Microsoft deploys inside the **Images** directory.
3. There are three GIF files inside the folder [[LAB FILES]]\StarterFiles. These files are named FeatureIcon.gif, SiteIcon.gif and WebPartIcon.gif. Right-click on the **WingtipWebParts** folder inside the **Images** folder and select the **Add » Existing Items...** command. When the resulting dialog opens, navigate to the folder location that contains the GIF files. Add all three GIF files to the project.
4. Add a new Web Part to the **WingtipWebParts** project named **HelloPart**.
5. Right-click on the **WingtipWebParts** project in Solution Explorer and select **Add » New Item**.
6. Select the **Web Part** project item template and give it a name of **HelloPart**.
7. Inspect the SPI node for the Web Part named **HelloPart**. You should be able to verify that it contains three SPI files named Elements.xml, HelloPart.cs and HelloPart.webpart.



1. Open the file named HelloPart.webpart and make the following modifications.
2. Find the element for the Title property and change its value to **The "Hello" Web Part**.
3. Find the element for the Description property and change its value to something else.

<?xml version="1.0" encoding="utf-8"?>

<webParts>

<webPart xmlns="http://schemas.microsoft.com/WebPart/v3">

<metaData>

<type name="WingtipWebParts.HelloPart.HelloPart,

$SharePoint.Project.AssemblyFullName$" />

<importErrorMessage>$Resources:core,ImportErrorMessage;</importErrorMessage>

</metaData>

<data>

<properties>

<property name="Title" type="string">The "Hello" Web Part</property>

<property name="Description" type="string">A most compelling Web Part</property>

</properties>

</data>

</webPart>

</webParts>

1. In addition to the property values for Title and Description, add three more additional Web Part properties to HelloPart.webpart using the following name/value pairs:

**ChromeType:** TitleAndBorder

**CatalogIconImageUrl:** \_layouts/images/WingtipWebParts/WebPartIcon.gif

**TitleIconImageUrl:** \_layouts/images/WingtipWebParts/WebPartIcon.gif

<?xml version="1.0" encoding="utf-8"?>

<webParts>

<webPart xmlns="http://schemas.microsoft.com/WebPart/v3">

<metaData>

<type name="WingtipWebParts.HelloPart.HelloPart, $SharePoint.Project.AssemblyFullName$" />

<importErrorMessage>$Resources:core,ImportErrorMessage;</importErrorMessage>

</metaData>

<data>

<properties>

<property name="Title" type="string">The "Hello" Web Part</property>

<property name="Description" type="string">A most compelling Web Part.</property>

<property name="ChromeType" type="string">TitleAndBorder</property>

<property name="CatalogIconImageUrl"

type="string">\_layouts/images/WingtipWebParts/WebPartIcon.gif</property>

<property name="TitleIconImageUrl"

type="string">\_layouts/images/WingtipWebParts/WebPartIcon.gif</property>

</properties>

</data>

</webPart>

</webParts>

1. Open the Web Parts elements.xml and make the following modifications.
2. Modify the URL attribute of File element to change the \*.webpart file name to ensure that it is unique. Do this by appending the text value of "**WingtipWebPart\_**" to the beginning of the URL property value.

<?xml version="1.0" encoding="utf-8"?>

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

<Module Name="HelloPart" List="113" Url="\_catalogs/wp">

<File Path="HelloPart\HelloPart.webpart"

Url="WingtipWebPart\_HelloPart.webpart"

Type="GhostableInLibrary">

<Property Name="Group" Value="Custom" />

</File>

</Module>

</Elements>

1. Change the value for the Group property to **"Wingtip Web Parts"**.

<?xml version="1.0" encoding="utf-8"?>

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

<Module Name="HelloPart" List="113" Url="\_catalogs/wp">

<File Path="HelloPart\HelloPart.webpart"

Url="WingtipWebPart\_HelloPart.webpart"

Type="GhostableInLibrary">

<Property Name="Group" Value="Wingtip Web Parts" />

</File>

</Module>

</Elements>

1. Open the C# source file named HelloPart.cs. Modify the code with this very simple starting point for a Web Part class implementation.

namespace WingtipWebParts.HelloPart {

public class HelloPart : WebPart {

protected Label label;

protected override void CreateChildControls() {

label = new Label();

label.Text = "Hello Web Part";

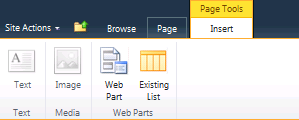
Controls.Add(label);

}

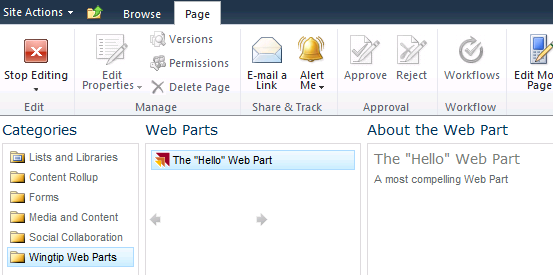
}

}

1. Build the **WingtipWebParts** project and make sure there are no compilation errors. If there are errors when running the **Build** command, find and fix these errors until you can run the **Build** command without experiencing any errors.
2. Run the **Deploy** command which will retract and deploy the solution package for the **WingtipWebParts** project.
3. In a browser, navigate to the test site at **http://intranet.wingtip.com/sites/WebParts** and follow these steps to go through the activation process.
4. Click **Site Actions » Site Settings** to navigate to the **Site Settings** page.
5. Inside the **Site Collection Administration** section of the **Site Settings** page, click on the **Site collection features** link to navigate to the **Site Collection Administration » Features** page.
6. Locate the **WingtipWebParts Feature1** feature and activate it. If it was already active, then deactivate first and then activate it. The key point here is that feature activation is what provisions the \*.webpart file into the Web Part Gallery which will allow you to test your work.
7. Now it is time to add a test instance of the Web Part you have just created to a Web Part Page. Navigate to the home page default.aspx. Click on the **Site Actions** button and choose the **Edit Page** menu item. Select the **Left** Web Part Zone, so that an additional **Page Tools** ribbon tab becomes visible. Click on the **Insert** tab on the ribbon and click on the **Web Part** button which is shown in the following screenshot.



1. At this point you should see the new SharePoint 2010 UI for adding new Web Parts to a page. You should be able to see standard Web Part Categories such as **Lists and Libraries**, **Authoring** and **Content Rollup**. You should also be able to see a new custom category named **Wingtip Web Parts** which you created when you edited the Group property value inside the elements.xml file. When you select the **Wingtip Web Parts** category in the left-hand section, you should be able to see your Web Part in the right-hand section with a title of **The "Hello" Web Part** that you added to HelloPart.webpart. Once you have selected the **The "Hello" Web Part**, make sure the drop-down box on the far right bottom (shown below) has the **Left** Web Part zone selected, then click the **Add** button to add the Web Part instance to the page.



1. At this point, you have gone through all the required steps to create, deploy and test a Web Part. The test instance of your Web Part should look like the following screenshot.

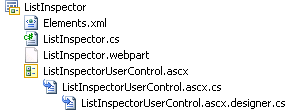


In this exercise you created a Web Part, added it to your project and added it to the sample SharePoint site.

### Exercise 2: Creating a Visual Web Part with AJAX Behavior

Now you will add a second Web Part using the Visual Web Part template. This makes it possible to create the UI for a Web Part using an ASP.NET User Control and the Visual Studio User Control Designer. You will also use the UpdatePanel control from ASP.NET AJAX to give your Web Part a Web 2.0 user experience eliminating postbacks.

1. Add a new Web Part to the **WingtipWebParts** project named **ListInspector**.
2. Right-click on the **WingtipWebParts** project in Solution Explorer and select **Add » New Item**.
3. Select the **Visual Web Part** project item template and give it a name of **ListInspector**.
4. Inspect the SPI node for the Web Part named **ListInspector**. Verify that it contains the SPI files named Elements.xml, ListInspector.cs, ListInspector.webpart, ListInspectorUserControl.ascx, ListInspectorUserControl.ascx.cs and ListInspectorUserControl.ascx.designercs.



1. Open the Web Part description file named ListInspector.webPart and examine the XML content inside. You should see the XML already contains two property elements for the Title property and the Description property. Update this Web Part description file so that it contains a property element and an associated value for each of the following name/value pairs:

**ChromeType**: TitleAndBorder

**Title**: List Inspector Web Part

**Description**: A Web Part which shows all the lists in the current site and allows you to get several of its property values.

**CatalogIconImageUrl**: \_layouts/images/WingtipWebParts/WebPartIcon.gif

**TitleIconImageUrl**: \_layouts/images/WingtipWebParts/WebPartIcon.gif

1. Open the Web Parts elements.xml and make the following modifications.
2. Modify the URL attribute of File element to change the \*.webpart file name to ensure that it is unique. Do this by appending the text value of "**WingtipWebPart\_**" to the beginning of the URL property.

<?xml version="1.0" encoding="utf-8"?>

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

<Module Name="ListInspector" List="113" Url="\_catalogs/wp">

<File Path="ListInspector\ListInspector.webpart"

Url="WingtipWebPart\_ListInspector.webpart"

Type="GhostableInLibrary" >

<Property Name="Group" Value="Custom" />

</File>

</Module>

</Elements>

1. Change the value for the Group property to "**Wingtip Web Part**".

<?xml version="1.0" encoding="utf-8"?>

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

<Module Name="ListInspector" List="113" Url="\_catalogs/wp">

<File Path="ListInspector\ListInspector.webpart"

Url="WingtipWebPart\_ListInspector.webpart"

Type="GhostableInLibrary" >

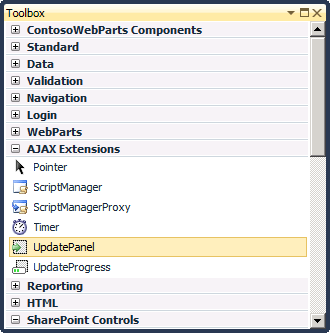
<Property Name="Group" Value="Wingtip Web Parts" />

</File>

</Module>

</Elements>

1. Open the User Control file for the Visual Web Part named ListInspectorUserControl.ascx. You should be able to experiment by moving back and forth between **Design** view and **Source** view. When you are in Design view, the designer should be empty at first as no controls or HTML text has been added.
2. Make sure the User Control is in Design view. Also ensure that the Visual Studio toolbox is visible. Look inside the **AJAX Extensions** section of the toolbox and find the **UpdatePanel** control as shown in the following screenshot. Drag and drop the **UpdatePanel** control onto the user control designer.



1. Switch the User Control back to **Source** view. Note that the User Control designer added a reference to the System.Web.Extensions assembly when you added the UpdatePanel control. Your code view should look like what is shown in the following code block.

<%@ Assembly Name="$SharePoint.Project.AssemblyFullName$" %>

<%@ Assembly Name="Microsoft.Web.CommandUI, [4-part assembly name]" %>

<%@ Register Tagprefix="SharePoint" Namespace="Microsoft.SharePoint.WebControls"

Assembly="Microsoft.SharePoint, [4-part assembly name]" %>

<%@ Register Tagprefix="Utilities" Namespace="Microsoft.SharePoint.Utilities"

Assembly="Microsoft.SharePoint, [4-part assembly name]" %>

<%@ Import Namespace="Microsoft.SharePoint" %>

<%@ Register Tagprefix="WebPartPages" Namespace="Microsoft.SharePoint.WebPartPages"

Assembly="Microsoft.SharePoint, [4-part assembly name]" %>

<%@ Control Language="C#" AutoEventWireup="true"

CodeBehind="ListInspector2UserControl.ascx.cs"

Inherits="WingtipWebParts.ListInspector2.ListInspector2UserControl" %>

<%@ Register assembly="System.Web.Extensions, [4-part assembly name]"

namespace="System.Web.UI" tagprefix="asp" %>

<asp:UpdatePanel ID="UpdatePanel1" runat="server">

</asp:UpdatePanel>

1. Add a ContentTemplate element inside the UpdatePanel element as shown in the following code block. Next, you will copy and paste a pre-provided table layout which contains ASP.NET controls from a text file in your lab folder. Using the Windows Explorer, navigate to [[LAB FILES]]\StarterFiles. Locate the file named ListInspectorTable.txt. Open the file with NotePad and copy all the text into inside the ContentTemplate element:

<asp:UpdatePanel ID="UpdatePanel1" runat="server">

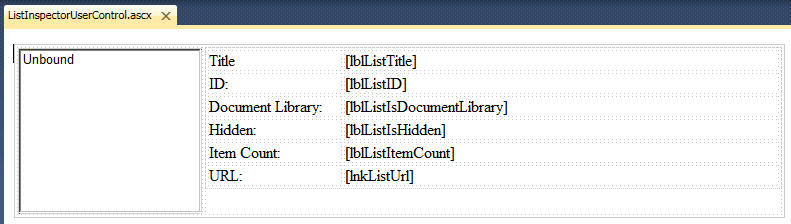
<ContentTemplate>

<!-- paste text from ListInspectorTable.txt here to create HTML table layout -->

</ContentTemplate>

</asp:UpdatePanel>

1. Switch the User Control back into **Design** view and verify that you can see the table layout.



1. Now it's time to write some code to bring this Web Part to life. Open the source file named ListInspectorUserControl.ascx.cs. Follow these steps to begin the implementation:
2. Add a using statement to import the Microsoft.SharePoint namespace.
3. Delete any pre-existing members (properties & methods from the class definition).
4. Add a protected property named SelectedListID based on the GUID type.
5. Add a protected property named UpdateListProperties based on the bool type.
6. Add a method named lstLists\_SelectedIndexChanged() using the parameter list shown in the following code block. Note that the **WingtipWebParts** project will not compile until you add this method implementation because the pre-provided table layout text you pasted in contains a list box named lstLists containing an attribute which attaches it to an event handler named lstLists\_SelectedIndexChanged().
7. Add an overridden implementation of the OnPreRender() method using the parameter list shown in the following code block.

using System;

using System.Web.UI;

using System.Web.UI.WebControls;

using System.Web.UI.WebControls.WebParts;

using Microsoft.SharePoint;

namespace WingtipWebParts.ListInspector {

public partial class ListInspectorUserControl : UserControl {

protected Guid SelectedListID = Guid.Empty;

protected bool UpdateListProperties = false;

protected void lstLists\_SelectedIndexChanged(object sender, EventArgs e) {

}

protected override void OnPreRender(EventArgs e) {

}

}

}

1. Provide an implementation for lstLists\_SelectedIndexChanged() as follows:

protected void lstLists\_SelectedIndexChanged(object sender, EventArgs e) {

SelectedListID = new Guid(lstLists.SelectedValue);

UpdateListProperties = true;

}

1. Provide an implementation for OnPreRender() do the following.
2. Clear the items from the lstLists control and then add a new list item for each SPList item in the current site. When adding ListItem instances to the ListBox control, use list title as the ListItem Text and use list's identifying GUID as the ListItem Value.

protected void OnPreRender(EventArgs e){

lstLists.Items.Clear();

SPWeb site = SPContext.Current.Web;

foreach (SPList list in site.Lists) {

ListItem listItem = new ListItem(list.Title, list.ID.ToString());

lstLists.Items.Add(listItem);

}

}

1. Since you have cleared the list box and re-added all the list items, you must reapply the selection made by the user. Do this using the following code.

protected override void OnPreRender(EventArgs e) {

lstLists.Items.Clear();

SPWeb site = SPContext.Current.Web;

foreach (SPList list in site.Lists) {

ListItem listItem = new ListItem(list.Title, list.ID.ToString());

lstLists.Items.Add(listItem);

}

if (SelectedListID != Guid.Empty) {

lstLists.Items.FindByValue(SelectedListID.ToString()).Selected = true;

}

}

1. If the UpdateListProperties field contains a value of true, then update all the Label controls on the right-hand side of the table layout to display properties of the selected list. Use the following code block to see the names of the Label controls that need to be updated as well as the SharePoint object model code required.

protected override void OnPreRender(EventArgs e) {

lstLists.Items.Clear();

SPWeb site = SPContext.Current.Web;

foreach (SPList list in site.Lists) {

ListItem listItem = new ListItem(list.Title, list.ID.ToString());

lstLists.Items.Add(listItem);

}

if (SelectedListID != Guid.Empty) {

lstLists.Items.FindByValue(SelectedListID.ToString()).Selected = true;

}

if (UpdateListProperties) {

SPList list = SPContext.Current.Web.Lists[SelectedListID];

lblListTitle.Text = list.Title;

lblListID.Text = list.ID.ToString().ToUpper();

lblListIsDocumentLibrary.Text = (list is SPDocumentLibrary).ToString();

lblListIsHidden.Text = list.Hidden.ToString();

lblListItemCount.Text = list.ItemCount.ToString();

lnkListUrl.Text = list.DefaultViewUrl;

lnkListUrl.NavigateUrl = list.DefaultViewUrl;

}

}

1. Now, you need to add one final piece of code at the very beginning of the OnPreRender() implementation to assign a value to the SelectedListID field in cases where the lstLists controls has a selected item but did not generate a postback resulting from a user selecting a different list.

protected override void OnPreRender(EventArgs e) {

if ((lstLists.SelectedIndex > -1) & (!UpdateListProperties)) {

SelectedListID = new Guid(lstLists.SelectedValue);

}

lstLists.Items.Clear();

SPWeb site = SPContext.Current.Web;

foreach (SPList list in site.Lists) {

ListItem listItem = new ListItem(list.Title, list.ID.ToString());

lstLists.Items.Add(listItem);

}

if (SelectedListID != Guid.Empty) {

lstLists.Items.FindByValue(SelectedListID.ToString()).Selected = true;

}

if (UpdateListProperties) {

SPList list = SPContext.Current.Web.Lists[SelectedListID];

lblListTitle.Text = list.Title;

lblListID.Text = list.ID.ToString().ToUpper();

lblListIsDocumentLibrary.Text = (list is SPDocumentLibrary).ToString();

lblListIsHidden.Text = list.Hidden.ToString();

lblListItemCount.Text = list.ItemCount.ToString();

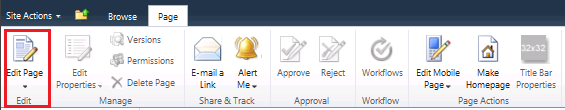
lnkListUrl.Text = list.DefaultViewUrl;

lnkListUrl.NavigateUrl = list.DefaultViewUrl;

}

}

1. Build the **WingtipWebParts** project and make sure there are no compilation errors. If there are errors when running the **Build** command, find and fix these errors until you can run the **Build** command without errors.
2. Now deploy the **ListInspector** Web Part by right-clicking on the **WingtipWebParts** project and running the Deploy command which will retract the previous version of the project's solution package and deploy the new one.
3. In the browser, navigate to the test site at http://intranet.wingtip.com/sites/WebParts and follow these steps to go through the activation process:
4. Click **Site Actions » Site Settings** to navigate to the **Site Settings** page.
5. Inside the **Site Collection Administration** section of the **Site Settings** page, click on the **Site collection features** link to navigate to the **Site Collection Administration » Features** page.
6. Locate the **Wingtip Web Parts** and activate it. If it was already active, then deactivate first and then activate it. The key point here is that feature activation is what provisions the Web Part Description file with the \*.webpart extension into the Web Part Gallery. This is required to complete the next step of this exercise.
7. Now it is time to add a test instance of the **ListInspector** Web Part. Navigate to the home page default.aspx. Activate the **Page** ribbon tab and choose **Edit Page**.



1. Select the **Left** web part zone, to make the additional **Page Tools** ribbon tab visible. Click on the **Insert** tab on the ribbon and click on the **Web Part** button. Select the **Wingtip Web Parts** category in the left-hand section which should then allow you to see the Web Part in the right-hand section with a title of **List Inspector Web Part** that you added to ListInspector.webpart. Select the List Inspector Web Part and then click the **Add** button to add the Web Part instance to the Left Zone of the page. Take the page out of edit mode again by activating the **Page** ribbon tab and clicking the **Stop Editing** button. The Web Part should look the one shown in the following screenshot.



Once you have tested the ListInspector Web Part and verified it can allow the user to see the properties of each list in the current site, you have completed all the exercises in this lab.

### Exercise 3: Creating Custom Editor Parts

In this exercise you will create a custom editor part and integrate it into an existing Web Part created in a previous exercise.

1. Add a new Web Part to the **WingtipWebParts** project named **FooWebPart**:
2. Right-click on the **WingtipWebParts** project in Solution Explorer and select **Add » New Item**.
3. Select the **Web Part** project item template and give it a name of **FooWebPart**.
4. Open the Web Part description file named FooWebPart.webPart and examine the XML content inside. You should see the XML already contains two property elements for the Title property and the Description property. Update this Web Part description file this so that it contains a property element and an associated value for each of the following name/value pairs:

**ChromeType**: TitleAndBorder

**Title**: Foo Web Part

**Description**: A sample Web Part that displays a value modified by an Editor Part.

**CatalogIconImageUrl**: \_layouts/images/WingtipWebParts/WebPartIcon.gif

**TitleIconImageUrl**: \_layouts/images/WingtipWebParts/WebPartIcon.gif

1. Open the Web Parts elements.xml and make the following modifications:
2. Modify the URL attribute of File element to change the \*.webpart file name to ensure that it is unique. Do this by appending the text value of "**WingtipWebPart\_**" to the beginning of the URL property value.

<?xml version="1.0" encoding="utf-8"?>

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

<Module Name="FootWebPart" List="113" Url="\_catalogs/wp">

<File Path="FootWebPart\FootWebPart.webpart"

Url="WingtipWebPart\_FooWebPart.webpart"

Type="GhostableInLibrary" >

<Property Name="Group" Value="Custom" />

</File>

</Module>

</Elements>

1. Change the value for the Group property to "**Wingtip Web Part**".

<?xml version="1.0" encoding="utf-8"?>

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

<Module Name="FootWebPart" List="113" Url="\_catalogs/wp">

<File Path="FootWebPart\FootWebPart.webpart"

Url="WingtipWebPart\_FooWebPart.webpart"

Type="GhostableInLibrary" >

<Property Name="Group" Value="Wingtip Web Parts" />

</File>

</Module>

</Elements>

1. Now, add the core logic necessary to store a string message in the Web Part and display it when the page is rendered. Add the following code to the FooWebPart.cs class that will declare a private field with an associated public property to modify the string as well as a Label control to display the string results:

public class FooWebPart : WebPart

{

private string \_displayMessage;

public string DisplayMessage

{

get { return \_displayMessage; }

set { \_displayMessage = value; }

}

protected override void CreateChildControls()

{

Label label = new Label();

label.Text = \_displayMessage;

Controls.Add(label);

}

}

1. The next step is to create the Editor Part. Right-click the **FooWebPart** SPI in the Solution Explorer tool window & select **Add » New Item…**. When prompted, create a new class named FooEditorPart.cs.
2. First, add a few extra using statements to the FooEditorPart.cs class:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Web.UI.WebControls.WebParts;

using System.Web.UI;

using System.Web.UI.WebControls;

1. To make this an Editor Part, change the accessibility to public for the FooEditorPart class and set it to inherit from the EditorPart. In addition add a TextBox field that will be used to display the message in the Web Part task pane and add the CreateChildControls() method to render the display message TextBox out:

public class FooEditorPart : EditorPart

{

protected TextBox \_displayMessageTextBox;

protected override void CreateChildControls()

{

base.CreateChildControls();

// create the textbox for the display string

Controls.Add(new LiteralControl("Message to display:<br />"));

\_displayMessageTextBox = new TextBox();

Controls.Add(\_displayMessageTextBox);

}

}

1. All editor parts must implement two methods. The first method, ApplyChanges(), is called when the user clicks the **OK** or **Apply** button in the Web Part’s task pane. Its job is to take the values out of the controls in the Editor Part and use them to update properties on the owning Web Part. Add this method to the FooEditorPart.cs class:

public override bool ApplyChanges()

{

EnsureChildControls();

// get reference to web part this editor part is associated with

FooWebPart webPart = this.WebPartToEdit as FooWebPart;

// set value in web part to that in the textbox

webPart.DisplayMessage = \_displayMessageTextBox.Text;

return true;

}

1. The second method, SyncChanges(), is called when the user selects **Modify This Web Part** from the Web Part’s verb’s menu. Its job is to get the values from the owning Web Part and use these values to update the Editor Part’s controls. Add this method to the FooEditorPart.cs class:

public override void SyncChanges()

{

EnsureChildControls();

// get reference to web part this editor part is associated with

FooWebPart webPart = this.WebPartToEdit as FooWebPart;

// set value in textbox to that of the display message

\_displayMessageTextBox.Text = webPart.DisplayMessage;

}

1. At this point the Editor Part is complete. The next step is to modify the Web Part so that it will create a new instance of the editor part and add it to the Editor Part collection for the Web Part. Go back to the FooWebPart.cs class and add the method CreateEditorParts() which will create a new instance of the Editor Part and return a new collection of editor parts to the Web Part framework that includes all the default Editor Parts as well as the custom one you just created:

public override EditorPartCollection CreateEditorParts()

{

// create container for new editor parts

List<EditorPart> editorParts = new List<EditorPart>(1);

// create an instance of the custom editor part

EditorPart editorPart = new FooEditorPart();

editorPart.ID = this.ID + "\_EditorPart";

editorParts.Add(editorPart);

// return a new collection of all the editor parts for this web part

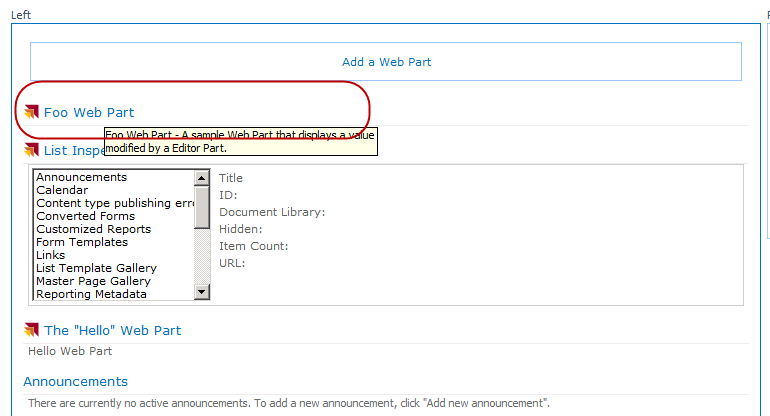
// which includes the default ones + custom ones

return new EditorPartCollection(base.CreateEditorParts(), editorParts);

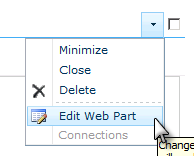
}

**Note:** you may need to add a using statement for the System.Collections.Generics, namespace for the List<T> object added in this step.

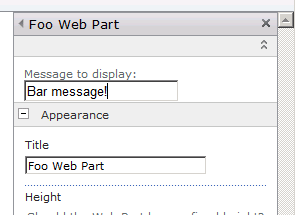
1. Build the **WingtipWebParts** project and make sure there are no compilation errors. If there are errors when running the **Build** command, find and fix these errors until you can run the **Build** command without errors.
2. Now deploy the **Foo Web Part** by right-clicking on the **WingtipWebParts** project and running the **Deploy** command which will retract the previous version of the project's solution package and deploy the new one.
3. In browser, navigate to the test site at http://intranet.wingtip.com/sites/WebParts and follow these steps to go through the activation process.
4. Click **Site Actions » Site Settings** to navigate to the **Site Settings** page.
5. Inside the **Site Collection Administration** section of the **Site Settings** page, click on the **Site collection features** link to navigate to the **Site Collection Administration » Features** page.
6. Locate the **Wingtip Web Parts** and activate it. If it was already active, then deactivate first and then activate it. The key point here is that feature activation is what provisions the Web Part description file with the \*.webpart extension into the Web Part Gallery. This is required to complete the next step of this exercise.
7. Now it is time to add a test instance of the **Foo Web Part**. Navigate to the home page default.aspx. Activate the **Page** ribbon tab and choose **Edit Page**.
8. Find the **Foo Web Part** and add it to the page. From the figure below, you can see that the Web Part is not displaying any message:



1. To open the Web Part task pane, hover the mouse over the title bar on the Web Part and look at the right-hand side of the Web Part zone. You’ll see a down arrow next to a checkbox appear. Click the down arrow to bring up the Web Part’s verbs menu & select **Edit Web Part**. This will bring up the task pane.



1. Add some text to the **Display Message** textbox in the custom Editor Part and click **OK** or **Apply**. Notice how the message is now appearing in the Web Part rendering!





In this exercise you created a custom editor part and integrated it into an existing Web Part created in a previous exercise.

### Exercise 4: Customizing the Verbs Menu

In this exercise you will learn how to customize the Web Part verb menu. There are three types of verbs you can add to the verbs menu: (1) one that calls a client-side script when clicked, (2) one that issues a postback and raises a server-side event when clicked and (3) one that raises a client-side script when clicked and then issues the postback to raise a server-side event.

1. Using the **Foo Web Part** that you created in the previous exercise, open the FooWebPart.cs file. Adding custom verbs allows you customize the interaction options available to users. This is done by overriding the Verbs property on a Web Part and returning a collection of verbs to render in the Web Part page. Within the FooWebPart.cs code file, add a new property:

public override WebPartVerbCollection Verbs

{

get

{

}

}

1. First create the client side verb. This is done by first creating the verb and within the constructor, passing in the ID of the verb in the menu as well as the client-side script to run. Next, set the Title property (which is what will show in the verb menu) and the Description property (which is the tooltip when hovering over the verb).

public override WebPartVerbCollection Verbs

{

get

{

// create client-side verb

string clientScript = "alert('Client-side event clicked');";

WebPartVerb clientSideVerb = new WebPartVerb("ClientSideVerb", clientScript);

clientSideVerb.Text = "Client-side script verb";

clientSideVerb.Description = "This is the message that shows in the tool tip.";

}

}

1. Next, create the verb that will execute a server side callback. This is done in the exact same way as creating a client verb except you’ll use a different overloaded constructor in the WebPartVerb. This one will allow you to name the server-side event handler to call upon clicking the verb. Add the following markup to the overridden Verbs property as well as the server side event handler:

public override WebPartVerbCollection Verbs

{

get

{

// create client-side verb

string clientScript = "alert('Client-side event clicked');";

WebPartVerb clientSideVerb = new WebPartVerb("ClientSideVerb", clientScript);

clientSideVerb.Text = "Client-side script verb";

clientSideVerb.Description = "This is the message that shows in the tool tip.";

// create server-side verb

WebPartVerb serverSideVerb = new WebPartVerb("ServerSideVerb", OnClickServerSideVerb);

serverSideVerb.Text = "Server-side script verb";

serverSideVerb.Description = "This is the message that shows in the tool tip.";

}

}

protected void OnClickServerSideVerb(object sender, EventArgs e)

{

\_displayMessage = "Server-side verb clicked.";

}

1. The third type of verb to create is one that will do both the client-side and server-side behavior. This is done just like the other verbs by using yet another overloaded constructor on the WebPartVerb object. Notice in this one, you’ll specify both the client-side and server-side handlers:

public override WebPartVerbCollection Verbs

{

get

{

// create client-side verb

string clientScript = "alert('Client-side event clicked');";

WebPartVerb clientSideVerb = new WebPartVerb("ClientSideVerb", clientScript);

clientSideVerb.Text = "Client-side script verb";

clientSideVerb.Description = "This is the message that shows in the tool tip.";

// create server-side verb

WebPartVerb serverSideVerb = new WebPartVerb("ServerSideVerb", OnClickServerSideVerb);

serverSideVerb.Text = "Server-side script verb";

serverSideVerb.Description = "This is the message that shows in the tool tip.";

// create client & server-side verb

WebPartVerb comboVerb = new WebPartVerb("ComboVerb", OnClickComboVerb, clientScript);

comboVerb.Text = "Client-side + server side verb";

comboVerb.Description = "This is the message that shows in the tool tip.";

}

}

protected void OnClickServerSideVerb(object sender, EventArgs e)

{

\_displayMessage = "Server-side verb clicked.";

}

protected void OnClickComboVerb(object sender, EventArgs e)

{

\_displayMessage = "Client+Server-side verb clicked.";

}

1. The last step is to take all the verbs that were created, add them to the collection of verbs and return that collection to the Web Part framework:

public override WebPartVerbCollection Verbs

{

get

{

// create client-side verb

string clientScript = "alert('Client-side event clicked');";

WebPartVerb clientSideVerb = new WebPartVerb("ClientSideVerb", clientScript);

clientSideVerb.Text = "Client-side script verb";

clientSideVerb.Description = "This is the message that shows in the tool tip.";

// create server-side verb

WebPartVerb serverSideVerb = new WebPartVerb("ServerSideVerb", OnClickServerSideVerb);

serverSideVerb.Text = "Server-side script verb";

serverSideVerb.Description = "This is the message that shows in the tool tip.";

// create client & server-side verb

WebPartVerb comboVerb = new WebPartVerb("ComboVerb", OnClickComboVerb, clientScript);

comboVerb.Text = "Client-side + server side verb";

comboVerb.Description = "This is the message that shows in the tool tip.";

// create a verbs collection and return it back to the Web Part framework

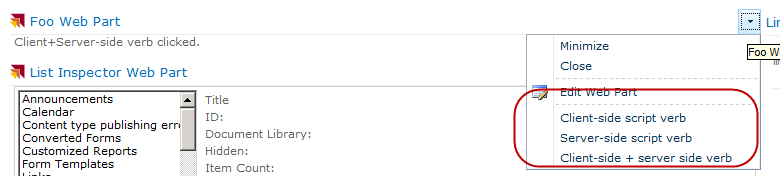
WebPartVerb[] verbs = new WebPartVerb[] { clientSideVerb, serverSideVerb, comboVerb };

return new WebPartVerbCollection(base.Verbs, verbs);

}

}

1. Build the **WingtipWebParts** project and make sure there are no compilation errors. If there are errors when running the **Build** command, find and fix these errors until you can run the **Build** command without errors.
2. Now deploy the **Foo Web Part** by right-clicking on the **WingtipWebParts** project and running the **Deploy** command which will retract the previous version of the project's solution package and deploy the new one.
3. In browser, navigate to the test site at http://intranet.wingtip.com/sites/WebParts and follow these steps to go through the activation process:
4. Click **Site Actions » Site Settings** to navigate to the **Site Settings** page.
5. Inside the **Site Collection Administration** section of the **Site Settings** page, click on the **Site collection features** link to navigate to the **Site Collection Administration » Features** page.
6. Locate the **Wingtip Web Parts** and activate it. If it was already active, then deactivate first and then activate it. The key point here is that feature activation is what provisions the Web Part description file with the \*.webpart extension into the Web Part Gallery. This is required to complete the next step of this exercise.
7. Now it is time to add a test instance of the **Foo Web Part**. Navigate to the home page default.aspx. Activate the **Page** ribbon tab and choose **Edit Page**. From the figure below, you can now see the three new verbs!



1. Clicking on each verb will have a different behavior.

In this exercise you will learned how to customize the Web Part verb menu. There are three types of verbs you can add to the verbs menu: (1) one that calls a client-side script when clicked, (2) one that issues a postback and raises a server-side event when clicked and (3) one that raises a client-side script when clicked and then issues the postback to raise a server-side event.