## SharePoint Developer Tools in Visual Studio 2010

**Lab Time**: 45 minutes

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

**Lab Overview**: In this lab you will begin using the SharePoint Developer Tools in Visual Studio 2010. This will give you a chance to become familiar with the standard project structure used in 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 SharePoint Developer Tools deployment options as well debug a SharePoint Developer Tools project by single-stepping through the code in your solution.

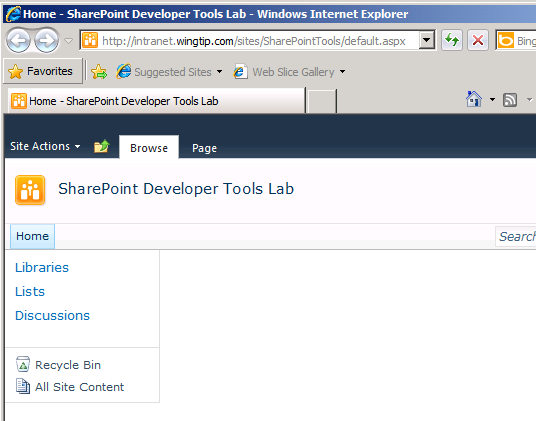
Lab Setup Requirements

* 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/SharePointTools**. 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.

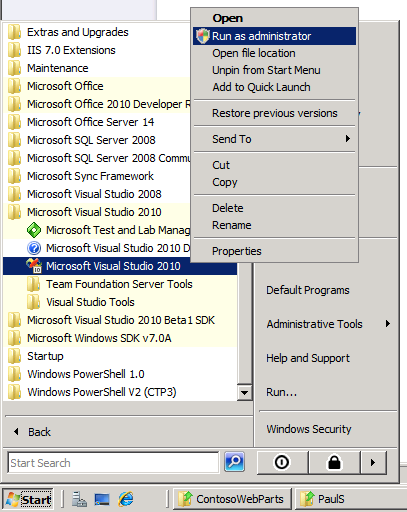
### Exercise 1: Creating a SharePoint 2010 Project

In the first exercise, you will create an empty project and focus on the aspects of the Visual Studio 2010 SharePoint Tools that are common across all projects created with this toolset. Most of your work will involve changing the properties of the project and properties of the project's main feature.

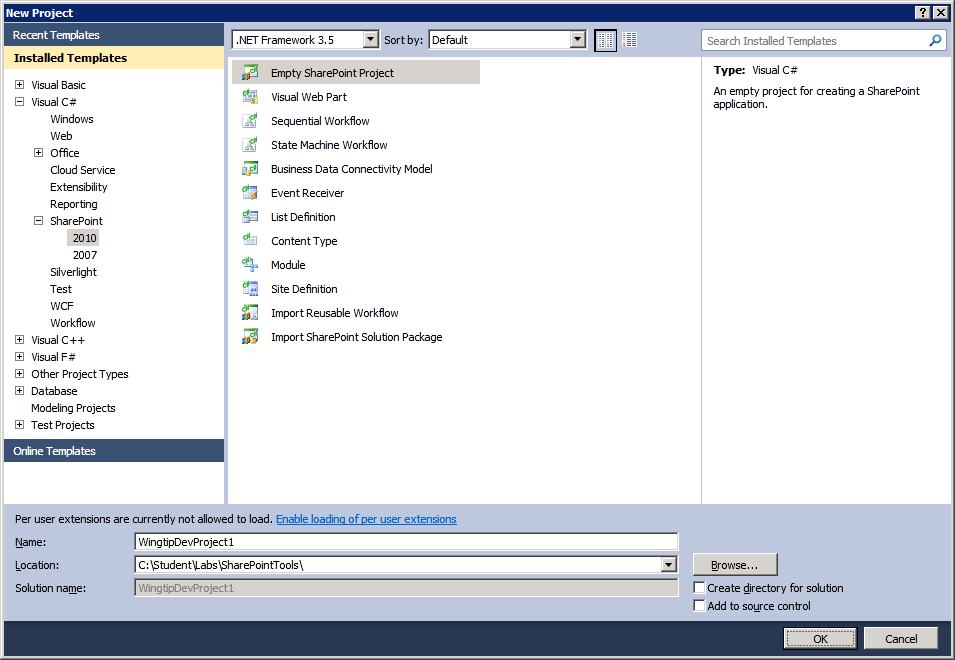
1. Launch the Internet Explorer and navigate to the top-level site at **http://intranet.wingtip.com/sites/SharePointTools**. You should observe that the newly-created site is a Blank site. This is the site you will use to test and debug the project you will develop throughout this entire lab exercise.



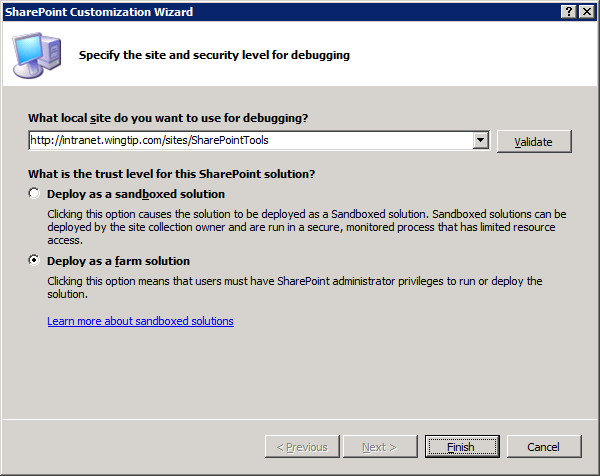
1. Now launch Microsoft Visual Studio 2010 with administrator privileges by right-clicking on its **Start** menu shortcut and clicking the **Run as Administrator** option. Starting Visual Studio in this manner (i.e. with administrative permissions) makes Visual Studio debugger work correctly when attaching to the SharePoint worker process.



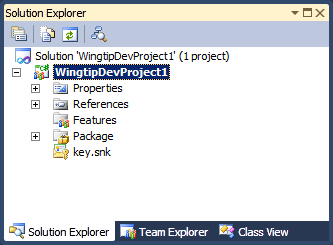
1. Create a new Visual Studio project named **WingtipDevProject1** based on the SharePoint Tools project of type **Empty SharePoint Project**. You can find this project type using the path **Visual C# » SharePoint » 2010**. Ensure the **.NET Framework 3.5** is selected instead of the default .NET Framework 4.0. Click **OK** to start the **SharePoint Customization Wizard**.



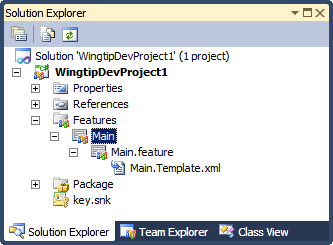
1. When the **SharePoint Customization Wizard** prompts you for a local debugging site, enter the URL of **http://intranet.wingtip.com/sites/SharePointTools**. Change the trust level to **Deploy as a farm solution** and then click **Finish**.



1. Once the new project has been created, you should be able to observe it has four nodes: **Properties**, **References**, **Features** and **Package**. While the first two nodes are standard on all Visual Studio projects, the **Features** node and the **Package** node are unique to projects create with the Visual Studio 2010 SharePoint Tools.



1. Right-click the **Features** node and choose **Add Feature**. A feature with the name **Feature1** is added. Right-click on this node and click **Rename**. Rename this feature to **Main**.

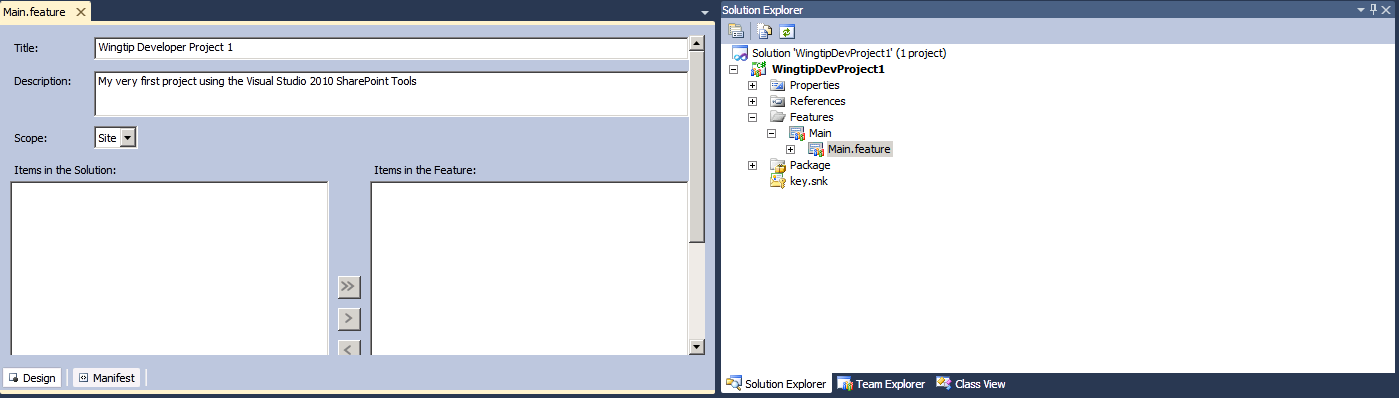


1. Double-click the **Main** feature to open the feature in the feature designer. Use the feature designer to change the feature’s **Title** and **Description** using the following values:

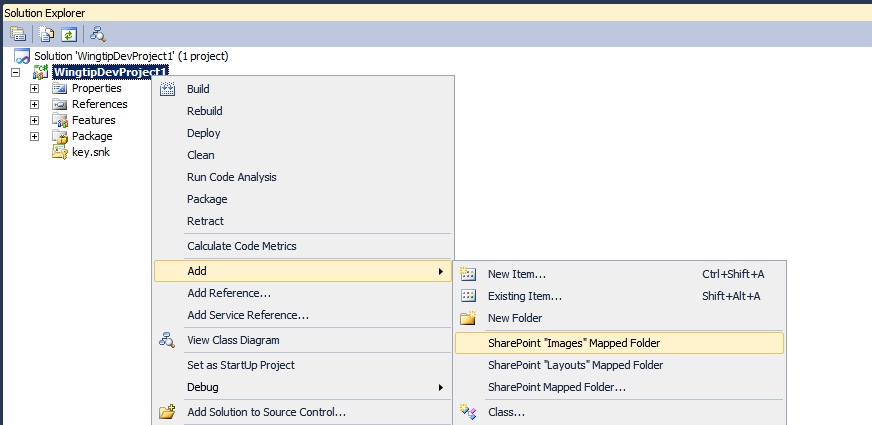
**Title:** Wingtip Developer Project 1

**Description:** My very first project using the Visual Studio 2010 SharePoint Tools

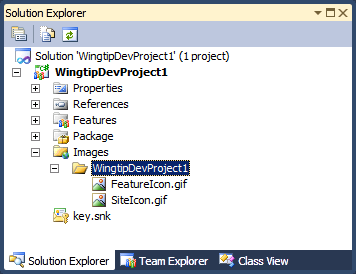
1. Change the **Scope** setting of the **Main** feature from the default value of **Web** to a value of **Site**. Don't be confused here. Remember a **Scope** setting of **Web** results in site-level activation of the feature while a **Scope** setting of **Site** results in activation at the site collection level. This feature will now be available for activation at the level of the site collection.



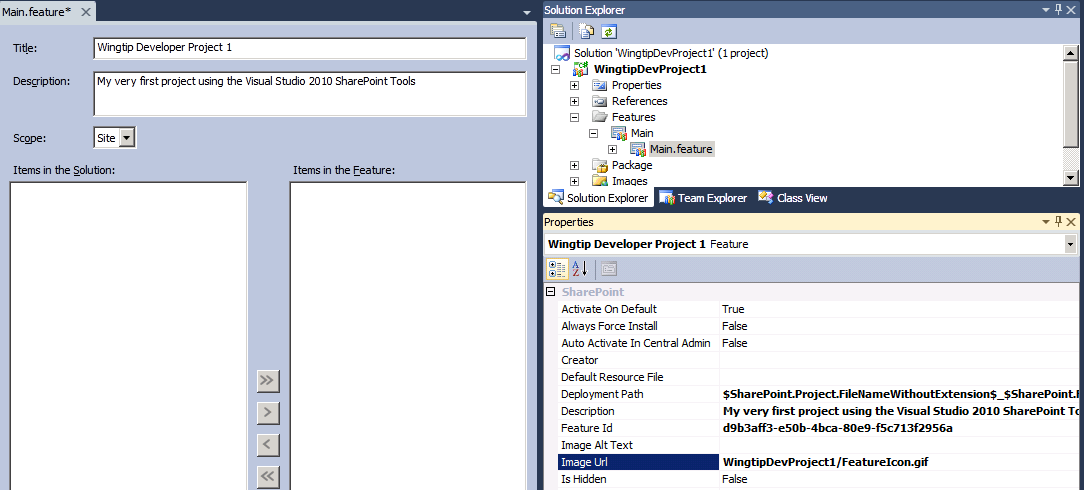
1. Now you are going to add a few graphic images into your project so they are deployed inside the SharePoint **Images** folder. The first step to doing this correctly is to add a mapped folder to your project for the SharePoint **Images** folder. Accomplish this by right-clicking the **WingtipDevProject1** project in the Solution Explorer, expanding the **Add** menu and selecting the **SharePoint "Images" Mapped Folder** command.



1. 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 **WingtipDevProject1**. 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 **WingtipDevProject1** to avoid file name conflicts with the image files that Microsoft deploys inside the **Images** directory.
2. There are two files inside the folder **[[LAB FILES]]\StarterFiles**. These files are named FeatureIcon.gif and SiteIcon.gif. Right-click on the **WingtipDevProject1** 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 both GIF files to the project.



1. The first GIF file that you will put to use is the FeatureIcon.gif by modifying the **Image Url** property of the **Main** feature.
2. Open the **Main** feature in the feature designer and make sure it is the active window.
3. Find the property sheet for the **Main** feature.
4. Update the **Image Url** property with an URL that is relative to the **Images** which should be WingtipDevProject1/FeatureIcon.gif.



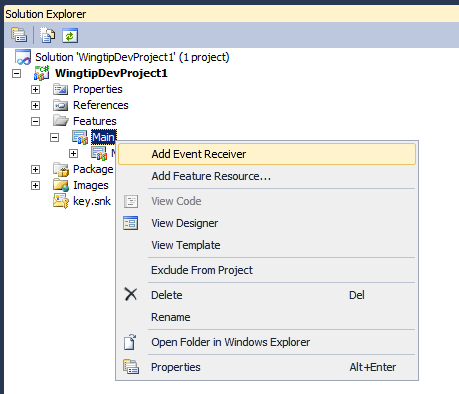
1. Save all the work you have done to the **Main** feature.

In this exercise you created your first feature using the SharePoint Developer Tools and added a mapped folder to the project. In future exercises you’ll add additional pieces to the project to make it more useful.

### Exercise 2: Adding a Feature Event Receiver

In exercise 1 you learned how to modify project properties and features properties using visual designers and property sheets. Now you are going to add some code. Over the next few steps, you are going to add a feature receiver so you can write code against the SharePoint object model that will automatically execute during feature activation and feature deactivation.

1. Add an event receiver by right-clicking on the **Main** featurenode in the Solution Explorer and clicking on the context menu with the caption of **Add Event Receiver**.



1. Open the source file named Main.EventReceiver.cs and inspect what is inside. You should be able to see that there are several method stubs inside the class definition that are commented out. There is also a GUID attribute that has been applied to the receiver class to give a unique identifier. Do not remove the GUID from the class because it will be used behind the scenes by the SPT during the packaging process. Uncomment the two methods named FeatureActivated() and FeatureDeactivating(). Then remove all the extraneous comments so your code looks like this.

using System;

using System.Runtime.InteropServices;

using System.Security.Permissions;

using Microsoft.SharePoint;

using Microsoft.SharePoint.Security;

namespace WingtipDevProject1.Features.Main {

[Guid("19d897a9-1520-48aa-87e1-0eaa12c08c88")] *// your GUID will be different*

public class MainEventReceiver : SPFeatureReceiver {

public override void FeatureActivated(SPFeatureReceiverProperties properties) {

}

public override void FeatureDeactivating(SPFeatureReceiverProperties properties) {

}

}

}

**Note:** In the previous version of SharePoint, developers were required to override all four Feature receiver methods because SPFeatureReceiver was an abstract class. This is no longer the case in SharePoint 2010 as now it is a concrete class so developers only need override the events they want to handle.

1. Implement the FeatureActivated() method to modify Title property and the ImageUrl property of the top-level site and to call the Update() method to save this modification back to the content database. Before changing of the Title property, your code should track its original value so that your feature can restore it upon feature deactivation. Implementation the FeatureActivated() method using the following steps.
2. Obtain the SPSite reference which point to the site collection in which the feature is being activated.
3. Obtain a SPWeb reference which points to the top-level site
4. Query to top-level site's Title property and store the value as a name/value pair inside the SPWeb.Properties collection.
5. Modify the site's Title property with a creative text value such as "VS2010 SPT Rocks!"
6. Modify the site's SiteLogoUrl property to point to SiteIcon.gif inside the **Images** folder.

\_layouts/images/WingtipDevProject1/SiteIcon.gif;

1. Your code should like something like this.

public override void FeatureActivated(SPFeatureReceiverProperties properties) {

SPSite siteCollection = properties.Feature.Parent as SPSite;

if (siteCollection != null) {

// save top site's original Title and SiteLogoUrl

SPWeb site = siteCollection.RootWeb;

site.Properties["OriginalTitle"] = site.Title;

site.Properties.Update();

// update the Title and SiteIconUrl

site.Title = "VS 2010 SPT Rocks";

site.SiteLogoUrl = "\_layouts/images/WingtipDevProject1/SiteIcon.gif";

site.Update();

}

}

1. Now, implement FeatureDeactivating() so that your code restores the original Title property value and changes the SiteLogoUrl property value back to an empty string.

public override void FeatureDeactivating(SPFeatureReceiverProperties properties) {

SPSite siteCollection = properties.Feature.Parent as SPSite;

if (siteCollection != null) {

// restore top site's original Title and SiteLogoUrl

SPWeb site = siteCollection.RootWeb;

site.Title = site.Properties["OriginalTitle"];

site.SiteLogoUrl = string.Empty;

site.Update();

}

}

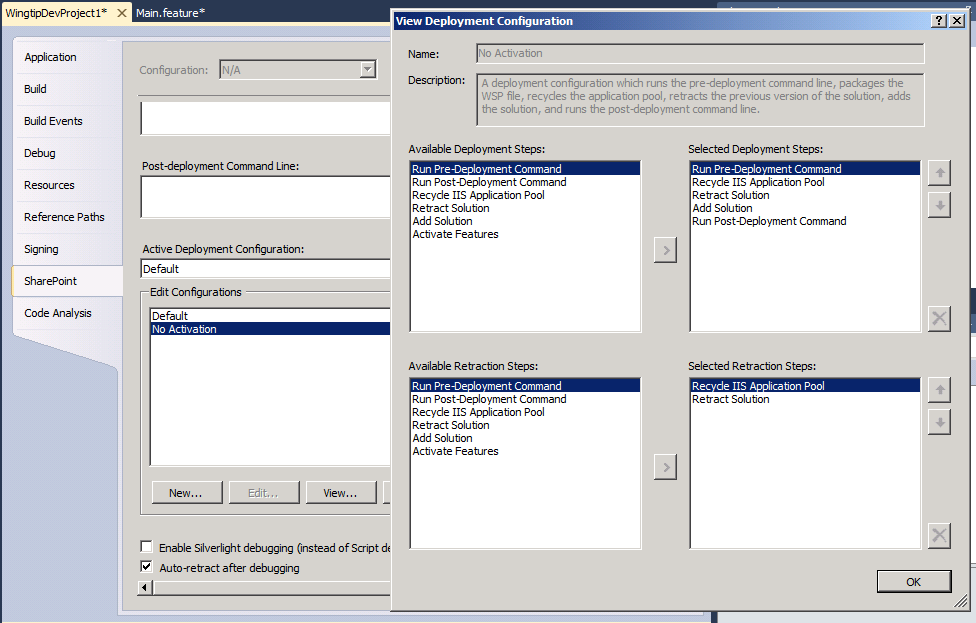
1. Right-click on the **WingtipDevProject1** project in Solution Explorer and run the **Build** command. This will ensure that all the code you have written in this exercise compiles. If you have compilation errors in your code, fix them until you are able to run the **Build** command without any errors.

You have now implemented the feature receiver. Save all your work and move on to the next exercise where you will deploy and test this code.

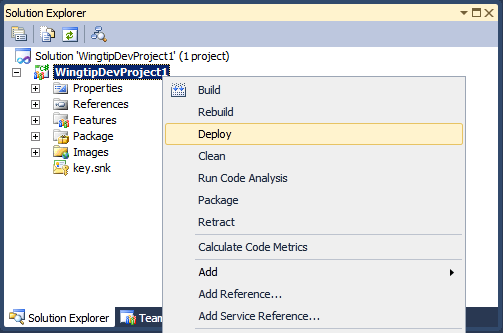
### Exercise 3: Deploying and Debugging SharePoint Projects

Now it's time to test your work and see how your project behaves when tested inside a SharePoint site collection.

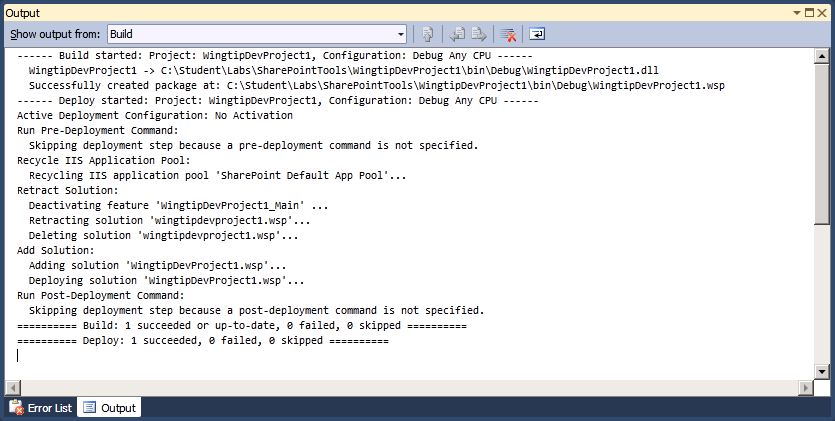
1. Start your work by inspecting and configuring the Visual Studio 2010 SPT project deployment options:
2. Right-click on the **WingtipDevProject1** project node in the Solution Explorer and click **Properties**.
3. Navigate to the **SharePoint** tab to see the project deployment options.
4. Note that there are two text boxes that allow you to add command-line instructions which will execute either just before and directly after the SPT deploy steps are processed. You are not going to add anything to either of the two top text boxes.
5. Find the combo box control with the caption **Active Deployment Configuration** and change the selected item from **Default** to **No Activation**.
6. Inside the **Edit Configurations** list box, select the **No Activation** configuration and click the **View…** button to inspect its deployment configuration steps.
7. Click **OK** to close the **View Deployment Configuration** dialog. Save all your work and close the page which shows the project properties.



1. Now it's time to run the **Deploy** command. First, make sure the **Output Window** is visible so you can see how the things progress during the execution of the **Deploy** command. Then right-click on the **WingtipDevProject1** project in Solution Explorer and run the project **Deploy** command.



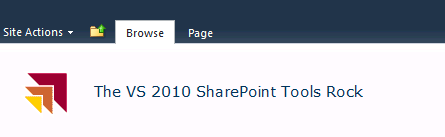
1. Examine the **Output Window** after the **Deploy** command has completed. Verify the steps that provided output to the Output Windows.



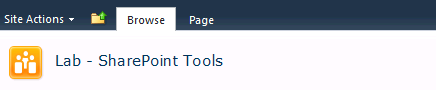
1. At this point, the solution package for the **WingtipDevProject1** project has been deployed on the local SharePoint. Let's test out your work by trying to activate the feature you defined inside the **WingtipDevProject1** project.
2. In browser, navigate to the test site at **http://intranet.wingtip.com/sites/SharePointTools**.
3. Click **Site Actions » Site Settings** to navigate to the **Site Settings** page.
4. 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.
5. Locate the feature you have been working on with a title of **Wingtip Developer Project 1**. You should also be able to see the feature's custom feature icon.



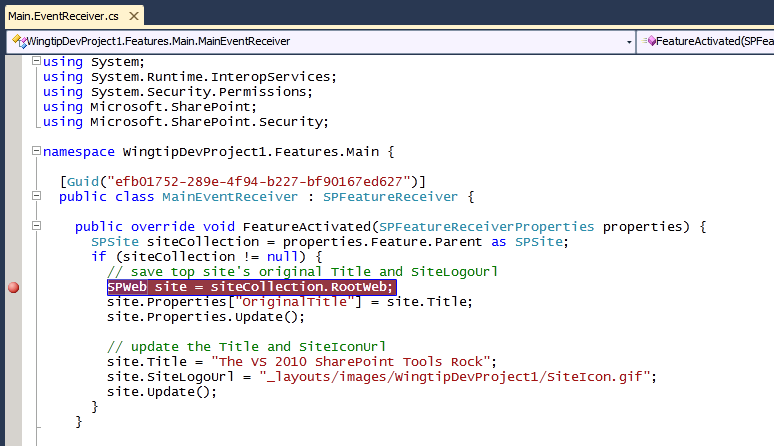
1. Activate the **Wingtip Developer Project 1** feature. If you return to the site's home page you should be able to verify that the code inside the FeatureActivated() event handler executed and changed the site's title and its site icon.



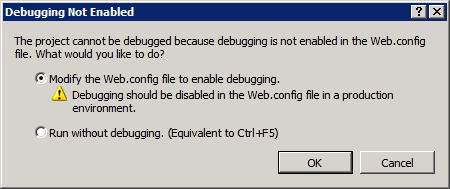
1. Now, navigate back to the Site Collection Administration for Features page. Deactivate the Wingtip Web Parts feature so that you can test the code in the **FeatureDeactivating()** event handler. Return to the site's home page and verify that the site title has been restored to its original value and that the site icon has been changed back to the default site icon for SharePoint sites.



1. Now it's time to practice entering debug mode so that you can single step through the code while it's executing inside the context of the test site.
2. Open the source file Main.EventReceiver.cs and add a breakpoint to FeatureActivated() method.



1. Start the project in debug mode by running **Debug » Start Debugging** menu command or pressing the equivalent shortcut key which is **[F5]**.
2. If this is the first time you have debugged a site within a particular SharePoint Web Application, the SharePoint Tools might display the following dialog. If you get this dialog, click the **OK** button and allow the SharePoint Tools to make the modifications to web.config and to continue debugging.



1. After running the Start Debugging command wait until the Visual Studio debugger has launched the Internet Explorer and navigated to the test site.
2. Within the test site, navigate to the **Site Collection Administration » Features** page and activate the **Wingtip Web Parts** feature. At this point you should fall into the Visual Studio debugger and break at the line where you set of the breakpoint. If you do not fall into the Visual Studio debugger when you first activate the feature, try to deactivate and reactivate the feature.
3. Press the **[F11]** key to single-step through the remaining lines of code. You should be able to hit **[F11]** repeatedly until the Visual Studio debugger relinquishes control back to the Internet Explorer.
4. Return to Visual Studio and stop the debugger by running the **Debug » Stop Debugging** menu command or by pressing the equivalent shortcut key which is **[Shift]+[F5]**.

In this exercise you deployed and tested your SharePoint project as well as saw the native debugging experience.