Deploying Branding Solutions for SharePoint 2010 Sites Using Sandboxed Solutions

**Overview**

The SharePoint development tools in Microsoft Visual Studio 2010 provide a simple and effective approach to packaging and deploying the files and code that are required to apply branding to Microsoft SharePoint 2010 sites using a sandboxed solution. This article describes a best practice for creating sandbox-compatible branding solutions by using custom master pages, cascading style sheets (CSS files), and images that can be deployed to SharePoint 2010 farms that are running either SharePoint Foundation 2010 or SharePoint Server 2010.

**Motivation for Targeting the Sandbox**

There is only one way to deploy a business solution that is developed for the preceding version of SharePoint Products and Technologies. A SharePoint solution that targets a Windows SharePoint Services 3.0 or an Office SharePoint Server 2007 farm must be deployed at farm-level scope by a farm administrator. Because farm solution deployment requires scoping custom files to front-end web servers, it poses certain risks to the health of the farm. Furthermore, most farm solutions install custom assembly DLLs in the global assembly cache on the web server, which allows the code inside to run with full trust. Therefore, many farm administrators require SharePoint solutions to undergo lengthy code reviews and rigid testing procedures before they can be deployed in a SharePoint farm. Farm administrators in some SharePoint environments go one step further and prohibit farm-level deployment of SharePoint solutions altogether.

SharePoint 2010 introduces new sandboxing architecture that provides a valuable alternative to farm-level deployment. Unlike farm solution deployment, sandboxed solution deployment does not require a farm administrator. Instead, a SharePoint solution that is developed as a sandboxed solution can be uploaded and activated by a business user within the scope of a site collection. This can significantly speed up the process of getting a custom SharePoint solution into service. Sandboxed solutions also let you develop custom SharePoint solutions in Visual Studio 2010 that target environments that do not allow for farm solution deployment, such as SharePoint Online.

There is an important point about the new sandbox architecture that you should fully understand. Designing and developing SharePoint solutions that are compatible with the sandbox increases your options for deployment. This is because you can deploy a SharePoint solution that has been developed to target the sandbox as either a sandboxed solution or as a farm solution. Just because you develop a SharePoint solution as a sandboxed solution does not mean that you always have to deploy it as a sandboxed solution.

This article contains an example of how a sandboxed solution provides greater flexibility. In a SharePoint farm that does not allow for farm-level solution deployment, users can upload and activate your SharePoint solution within the context of a site collection. In another SharePoint farm that does not pose these restrictions, the same SharePoint solution can be deployed as a farm solution, which makes it possible to obtain higher levels of performance and maintainability. This provides a design motivation for always targeting the sandbox when the constraints of the sandbox do not prevent you from accomplishing what you want.

**Development Techniques for the Sandbox**

Many SharePoint developer techniques cannot be used when you are developing sandboxed solutions. For example, a sandboxed solution cannot deploy files within the SharePoint root directory or any other location on the file system of the web server. That means you cannot deploy branding files to familiar locations that are commonly used in farm solutions, such as the IMAGES directory or the LAYOUTS directory. You must instead provision branding files, such as images and CSS files, inside the content database within the scope of the hosting site collection.

When you write code for a sandboxed solution, there are other noteworthy limitations. Although you can access the server-side object model of SharePoint Foundation 2010, the functionality that is available in a sandboxed solution is quite limited when compared to functionality that is available when you write code in a farm solution.

The high-level design of a sandboxed solution typically starts with adding one or more Features. When you add a Feature to a sandboxed solution, you can assign the Feature with an activation scope at either the site level or the site collection level. However, a sandboxed solution cannot contain a Feature that has an activation scope at the farm level or the web application level.

When you begin developing sandboxed solutions, it is important to distinguish between Features that activate at site collection scope and Features that activate at site scope. This is because a Feature that is configured to activate at site collection scope is automatically activated when the user activates the sandboxed solution that contains it. On the other hand, Features that are configured to activate at site scope are not automatically activated. They must be manually activated by the user or activated by using custom code. For this reason, you should generally design a sandboxed solution with at least one Feature configured to activate at site collection scope. This enables you to take control when the user activates your sandboxed solution.

You can add a Feature receiver to a Feature inside a sandboxed solution just like you can for a farm solution. However, be aware that the code you write for a Feature receiver in a sandboxed solution executes within the SharePoint Foundation Sandboxed Code Service and is subject to the constraints of the sandbox architecture. Fortunately, you can access the essential functionality in the server-side object model that enables you to configure all the sites in a site collection to use custom master pages and custom CSS files.

**Provisioning Files Using Modules**

When you create a SharePoint solution to brand a SharePoint 2010 site, you are required to deploy custom files such as master pages, CSS files, and images. When you are creating a sandboxed solution, you cannot rely on common techniques that are used in farm solutions, where custom files are deployed inside the SharePoint root directory on the file system of the web server. Instead, you must provision instances of these files from file templates by using the [Module Element (Module)](http://msdn.microsoft.com/en-us/library/ms434127.aspx).

For example, let's say that you want to deploy a master page named MyCustomLayout.master to the master page gallery in the top-level site. You can start by creating a Feature that is scoped at site collection level. Next, you can add a template file named MyCustomLayout.master in your Feature directory that contains the HTML layout for your custom branding solution. Finally, you add an element manifest that contains the following **Module** element that provisions an instance of MyCustomLayout.master to the master page gallery.

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

<Module Name="MasterPageGallery" Url="\_catalogs/masterpage" >

<File Url="MyCustomLayout.master" Type="GhostableInLibrary" >

<Property Name="UIVersion" Value="4" />

<Property Name="ContentTypeId" Value="0x010105" />

</File>

</Module>

</Elements>

When you create a **Module** element like this one to provision files in a document library, such as the master page gallery, it is important that you configure the **Url** attribute of the **Module** element by using the site-relative path to the root of the document library. When you provision a master page to the master page gallery, the **Url** attribute of the **Module** element should always be configured with the standard path of \_catalogs/masterpage.

When you provision a file into the scope of a document library, each [File Element (Module)](http://msdn.microsoft.com/en-us/library/ms459213.aspx) should contain a Type attribute with a value of GhostableInLibrary, as seen in the previous code example. When you provision master pages in SharePoint 2010, you should also add two [Property](http://msdn.microsoft.com/en-us/library/cc264281.aspx) elements inside the **File** element to configure two important properties named [UIVersion](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spfile.uiversion.aspx) and **ContentTypeId**.

SharePoint Foundation uses the **UIVersion** property of a master page to differentiate between master pages designed for the new SharePoint 2010 user interface (UI) and those designed for the older UI of Office SharePoint Server 2007. If you design your master page based on the SharePoint 2010 UI, you should configure the **UIVersion** property with a value of **4**. You can configure the **UIVersion** property with a value of **3** in Visual Upgrade scenarios when you have designed the master page based on the older UI.

The **ContentTypeId** property is used to differentiate between master pages and page layouts that are used by SharePoint Server 2010 publishing sites. When you deploy a master page, you should configure it with the correct **ContentTypeId** value for master pages, which is **0x010105**.

**Deploying CSS Files and Images to the Style Library**

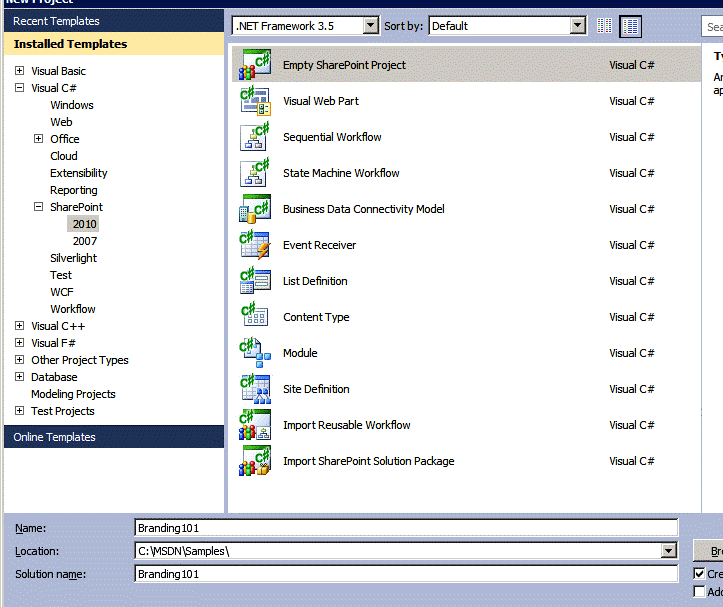
In Office SharePoint Server 2007, the publishing features create a special document library, named the Style Library, which Microsoft uses to deploy standard CSS files and image files that are used in publishing sites. The Style Library is also commonly used as a deployment target by web designers and developers who are using CSS files and image files to apply branding elements to Office SharePoint Server 2007 publishing sites.

When you are developing a generic and reusable branding solution for SharePoint Server 2010 farms, you cannot use the Style Library because it exists only in publishing sites. Windows SharePoint Services 3.0 does not create the Style Library when you create other types of sites, such as a Team site, Blank site, or a Document Workspace. Fortunately, this is no longer a problem in SharePoint 2010.

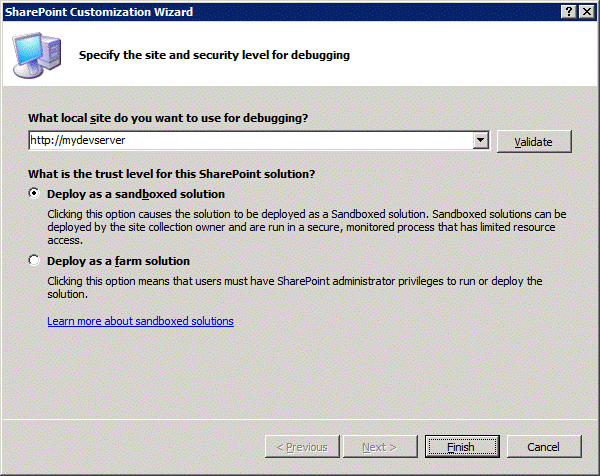
In SharePoint 2010, every site collection has its own Style Library. That's because Microsoft has moved the standard provisioning instructions for creating the Style Library out of the publishing features and into the Global site definition. Each time SharePoint Foundation 2010 creates a new site collection, it adds the Style Library to the top-level site. This makes the Style Library an ideal candidate for deploying CSS files and image files in a generic branding solution.

**Creating the Branding101 Project**

Now it is time to move through the steps of creating a generic branding solution using the SharePoint development tools in Visual Studio 2010. You can start by creating an **Empty SharePoint Project** named Branding101, as shown in Figure 1.

**Figure 1. New Empty SharePoint Project**  
  


When you create the new SharePoint project, the **SharePoint Customization Wizard** prompts you to provide a URL to a local SharePoint test site and to select either **Deploy as a sandboxed solution** or **Deploy as a farm solution** for testing. Be sure to select **Deploy as a sandboxed solution**, as shown in Figure 2.

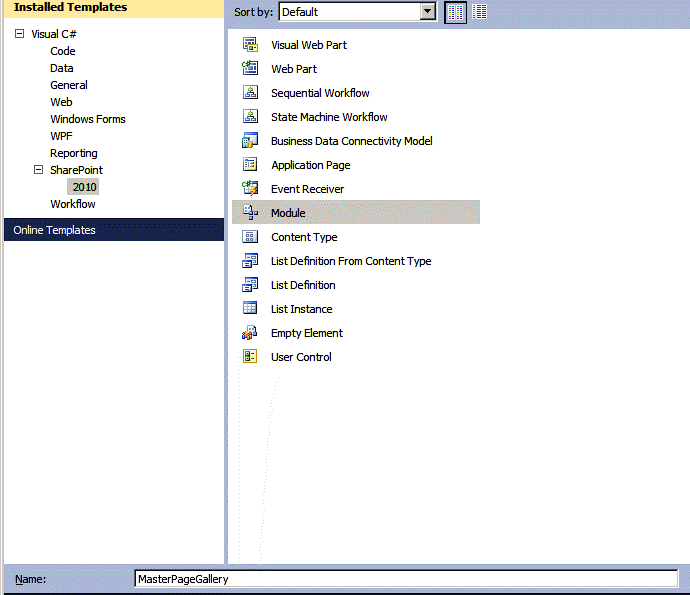
**Figure 2. Deploy as a sandboxed solution**  
  


**Creating and Deploying a Custom Master Page**

The first step in creating a custom branding solution is to create a new custom master page. Begin by creating a new **Module** project item that is used to deploy the custom master page into the master page gallery.

In Visual Studio 2010, in **Solution Explorer**, right-click the **Branding101** project node. On the **Project** menu, select **Add** and then **New Item**.

In the **Add New Item** dialog box, create a new **Module** project item named MasterPageGallery, as shown in Figure 3.

**Figure 3. MasterPageGallery Module**  
  


After you create the new Module, it contains an element manifest named Elements.xml and a sample element file named Sample.txt. Right-click the Sample.txt file, and rename it **Branding101.master**, as shown in Figure 4.

**Figure 4. MasterPageGallery project item**  
  


The next step is to modify the contents of the Branding101.master template file with the starting point for a custom master page. A popular technique is to copy and paste the text from the standard SharePoint 2010 master page named v4.master. Use Windows Explorer to locate the v4.master template file, which exists at the following path.

C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\TEMPLATE\GLOBAL\v4.master

When you locate the v4.master template file, open it and copy its contents to the Clipboard. Next, open the Branding101.master template file in your project and delete all the contents in the file. Paste the contents from v4.master inside Branding101.master, and then save your changes. After you successfully copy the content from v4.master, close this template file without saving any changes.

Currently, your custom master page has the same content as the standard SharePoint 2010 master page, v4.master. At this point, it makes sense to add at least one quick modification to Branding101.master so that you have some visual feedback that helps you determine when your custom master page is being used. You can do this by locating the opening body tag in Branding101.master and adding the following element just below it.

<div style="background-color:yellow">Branding101.master</div>

Now that you have created your custom master page, you must modify the Elements.xml file inside the MasterPageGallery module to ensure that it is correctly deployed to the master page gallery during Feature activation. Open the Elements.xml file and update the **Module** element and its inner **File** element with the following XML content.

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

<Module Name="MasterPageGallery"

Path="MasterPageGallery"

Url="\_catalogs/masterpage" >

<File Url="Branding101.master" Type="GhostableInLibrary" >

<Property Name="UIVersion" Value="4" />

<Property Name="ContentTypeId" Value="0x010105" />

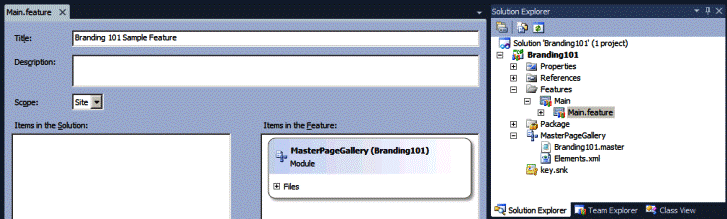
</File>

</Module>

</Elements>

At this point, you have finished your initial work on the MasterPageGallery module. Now you should turn your attention to the Feature that will be used to deploy this module. Notice that when you added the module to the **Branding101** project, the SharePoint development tools in Visual Studio 2010 automatically create a new Feature named Feature1. Locate this Feature in **Solution Explorer**, right-click it and rename it to **Main**.

After you rename the Feature, double-click the Feature node so that you can see the Feature in the **Feature Designer**. Change the **Title** of the Feature to something more appropriate, such as **Branding 101 Sample Feature**, as shown in Figure 5. It is also important to change the **Scope** of the Feature from **Web** to **Site** so that it activates at the level of the site collection and not the level of the site.

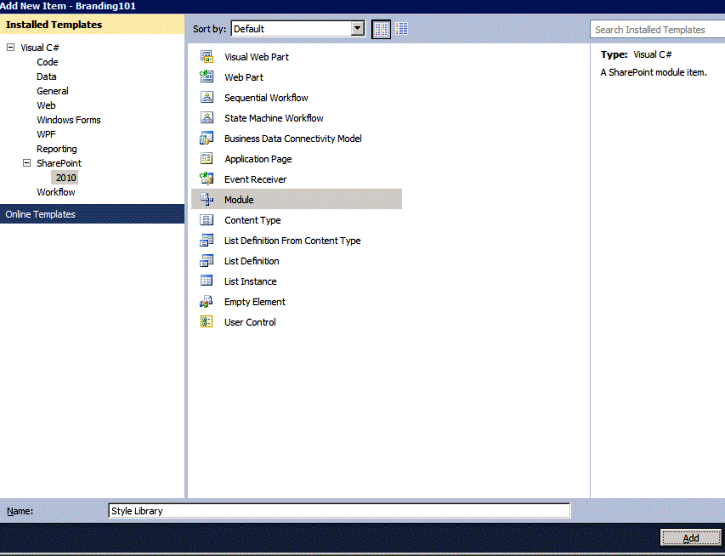
**Figure 5. Branding 101 Sample Feature in Feature Designer**  
  


**Creating and Deploying a Custom CSS File**

After you create a custom master page, the next step is to add a custom CSS file that contains the CSS rules for your branding solution. As discussed earlier, it is a best practice in SharePoint 2010 to deploy custom CSS files in the Style Library because it works with both sandboxed solutions and farm solutions. This technique works in farms that are running SharePoint Server 2010 and also farms that are running only SharePoint Foundation. It must be noted that many other common approaches to branding SharePoint 2010 sites do not provide this level of flexibility.

In **Solution Explorer**, right-click the **Branding101** project node, and on the **Project** menu, select **Add** and then **New Item**.

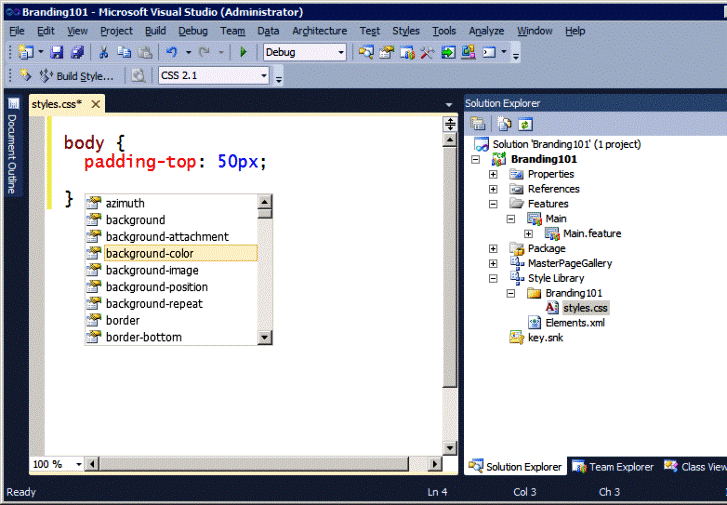
In the **Add New Item** dialog box, create a new **Module** project item named Style Library, as shown in Figure 6.

**Figure 6. Style Library module**  
  


The new **Module** project item initially contains an element manifest named Elements.xml and a sample element file named Sample.txt. Right-click Sample.txt and rename the file **Styles.css**.

Next, right-click the **Style Library** module node, and on the **Project** menu, select **Add** and then **New Folder** to create a new child folder in the **Style Library** module. Name the folder **Branding101**.

After you create this folder, you can move the styles.css file by dragging it to the folder in **Solution Explorer**.

**Figure 7. Styles.css file inside Style Library module**  
  


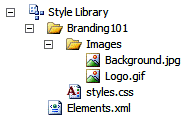
After you move the styles.css file into the **Branding101** folder, open the file, and delete all the existing content. Next, add a simple CSS rule for the body tag, and ensure that IntelliSense is working correctly. Add the following padding-top attribute so that you have some visual feedback as to when you are correctly linking to this CSS file.

body {

padding-top: 50px;

}

The next step is to add some image files to the project. The best approach is to add your custom image files inside the Style Library so that they are easier to reference from within your custom CSS file. In the **Branding101** folder, create an **Images** folder, and add some image files to this folder. This example uses two images: One image file, named Background.jpg, is used as the repeat background image for the page body; a second image named Logo.gif is used as the site logo.

**Figure 8. Image files in Style Library**  
  


After you add the image files to the **Images** folder, you can easily reference them from the CSS rules inside styles.css. To demonstrate this, add the following updates to the styles.css file to use Background.jpg as the repeating background image for the page body.

body {

padding-top: 50px;

background-image: url('Images/Background.jpg');

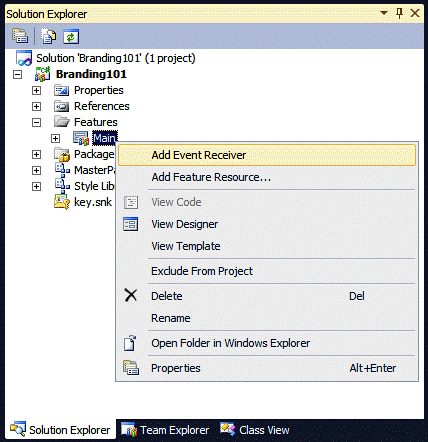
background-repeat: repeat;

}

At this point, you have completed your initial work with the Style Library. Unlike with the **MasterPageGallery** module that you created earlier, you do not have to modify the Elements.xml file manually. The SharePoint development tools in Visual Studio 2010 can add all the appropriate **File** elements for you behind the scenes. Furthermore, you can continue to add more image files into the **Images** folder, and they are automatically deployed in the correct location in the Style Library for you.

**Adding a Feature Receiver to Apply Branding Attributes**

Now that you have added two **Module** project items to deploy a custom master page and a custom CSS file, it is time to write some code to put them to use. You do this by adding a Feature receiver to the Feature named **Main**. In **Solution Explorer**, right-click the **Main** Feature node, and then select **Add Event Receiver** to add a Feature receiver class.

**Figure 9. Add Event Receiver**  
  


Inside the Feature receiver class, you must override and implement two methods named [FeatureActivated](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spfeaturereceiver.featureactivated.aspx) and [FeatureDeactivating](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spfeaturereceiver.featuredeactivating.aspx). Begin your coding by cleaning up and restructuring the source file for your Feature receiver class to look like the following example.

using System;

using System.Runtime.InteropServices;

using Microsoft.SharePoint;

namespace Branding101.Features.Main {

[Guid("cc5874a5-695b-49d2-9cd2-4fa12be83874")]

public class MainEventReceiver : SPFeatureReceiver {

public override void FeatureActivated(

SPFeatureReceiverProperties properties) {

// TODO: add activation code here.

}

public override void FeatureDeactivating(

SPFeatureReceiverProperties properties) {

// TODO: add deactivation code here.

}

}

}

Now let's step through what you have to do when the main Feature is activated. First, you must determine the path to Branding101.master in the master page gallery so that you can configure sites to use it as their master page. Note that the path to the master page must be calculated relative to the root of the hosting web application. Next, you must enumerate all the sites in the current site collection and update several properties of each site to use the custom master page and the custom CSS file. The following implementation of the **FeatureActivated** method updates the required **SPWeb** properties for each site.

public override void FeatureActivated(

SPFeatureReceiverProperties properties) {

SPSite siteCollection = properties.Feature.Parent as SPSite;

if (siteCollection != null) {

SPWeb topLevelSite = siteCollection.RootWeb;

// Calculate relative path to site from Web Application root.

string WebAppRelativePath = topLevelSite.ServerRelativeUrl;

if (!WebAppRelativePath.EndsWith("/")) {

WebAppRelativePath += "/";

}

// Enumerate through each site and apply branding.

foreach (SPWeb site in siteCollection.AllWebs) {

site.MasterUrl = WebAppRelativePath +

"\_catalogs/masterpage/Branding101.master";

site.CustomMasterUrl = WebAppRelativePath +

"\_catalogs/masterpage/Branding101.master";

site.AlternateCssUrl = WebAppRelativePath +

"Style%20Library/Branding101/Styles.css";

site.SiteLogoUrl = WebAppRelativePath +

"Style%20Library/Branding101/Images/Logo.gif";

site.UIVersion = 4;

site.Update();

}

}

}

The [MasterUrl](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spweb.masterurl.aspx) property of the [SPWeb](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spweb.aspx) object is the property that you use to redirect site pages and application pages to link to a custom master page, such as Branding101.master. The previous code example calculates the path to Branding101.master by combing the web application–relative path to the site and the site-relative path to the master page gallery in the top-level site, which always has a value of \_catalogs/masterpage.

|  |
| --- |
| **noteNote:** |
| This example updates the **SPWeb** property named [CustomMasterUrl](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spweb.custommasterurl.aspx) in addition to the **MasterUrl** property. Updating the **CustomMasterUrl** property is important only in publishing sites that contain publishing pages inside the **Pages** document library. The **CustomMasterUrl** property is used to reassign the master page for publishing pages. Assigning a new value to the **CustomMasterUrl** property in a SharePoint Foundation site has no effect nor does it cause any problems. |

The [AlternateCssUrl](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spweb.alternatecssurl.aspx) property is used to link the pages in a site to the custom CSS file named Styles.css. The linking behavior that is associated with the **AlternateCssUrl** property is implemented by the SharePoint [CssLink](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.webcontrols.csslink.aspx) control, which is defined in the head section in all the standard SharePoint 2010 master pages. The SharePoint **CssLink** control also adds a link to an essential CSS file named CoreV4.css and should therefore be included in any custom master page that targets SharePoint 2010.

Although the branding solution in this article relies on the approach of linking to a custom CSS file using the **AlternateCssUrl** property, be aware that some branding solutions take an alternative approach of linking to a custom CSS file by using the [CSSRegistration](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.webcontrols.cssregistration.aspx) control. For example, you can add the following **CssRegistration** element to the head section of a custom master page to link to a CSS file inside the Style Library.

<SharePoint:CssRegistration

name="<% $SPUrl:~sitecollection/Style Library/styles.css %>"

After="corev4.css"

runat="server"

/>

One benefit of using the [CssRegistration](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.webcontrols.cssregistration.aspx) control over the **AlternateCssUrl** property is that it enables you to link to more than one CSS file. A second advantage is that you can use the **CssRegistration** control in individual pages for scenarios where you have a CSS file that is used by some, but not all, pages within a site. However, using the **CssRegistration** control also has a downside because it relies on the $SPUrl expression, which requires the hosting farm to be running SharePoint Server 2010. If there is a chance that your branding solution will be used in farms that are running only SharePoint Foundation, you should prefer the technique of linking to a custom CSS file using the **AlternateCssUrl** property over using the **CssRegistration** control.

The [SiteLogoUrl](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spweb.sitelogourl.aspx) property is used in this example because it provides a quick and effective way of replacing the site image in the upper-left side of the page. Note that the behavior that is associated with the **SiteLogoUrl** property is implemented by the SharePoint [SiteLogoImage](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.webcontrols.sitelogoimage.aspx) control, which is defined in the Title Row section of standard SharePoint 2010 master pages, such as v4.master.

The [UIVersion](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spweb.uiversion.aspx) property is used to configure whether the current site should run in the UI mode for SharePoint 2010 sites or the older UI mode, which is used when migrating Office SharePoint Server 2007 sites to SharePoint 2010. The main effect that the **UIVersion** property setting has is whether the **CssLink** control links to the new standard CSS file that is created for SharePoint 2010 named corev4.css, or to the older standard CSS file named core.css, which is designed to style pages in Office SharePoint Server 2007. The example in this article assigns a value of **4** to the **UIVersion** property to ensure that pages are linked to corev4.css instead of core.css.

So far, you have learned how and why the Branding101 solution configures important **SPWeb** properties on every site in the current site collection during Feature activation. This section describes the code that should be executed during Feature deactivation. It makes sense to remove all the custom branding elements to return the current site collection back to its original state.

The following example is an implementation of the **FeatureDeactivating** method that returns all pages to use the standard master page v4.master and removes the link to the custom CSS file and the custom site logo.

public override void FeatureDeactivating(

SPFeatureReceiverProperties properties) {

SPSite siteCollection = properties.Feature.Parent as SPSite;

if (siteCollection != null) {

SPWeb topLevelSite = siteCollection.RootWeb;

// Calculate relative path of site from Web Application root.

string WebAppRelativePath = topLevelSite.ServerRelativeUrl;

if (!WebAppRelativePath.EndsWith("/")) {

WebAppRelativePath += "/";

}

// Enumerate through each site and remove custom branding.

foreach (SPWeb site in siteCollection.AllWebs) {

site.MasterUrl = WebAppRelativePath +

"\_catalogs/masterpage/v4.master";

site.CustomMasterUrl = WebAppRelativePath +

"\_catalogs/masterpage/v4.master";

site.AlternateCssUrl = "";

site.SiteLogoUrl = "";

site.Update();

}

}

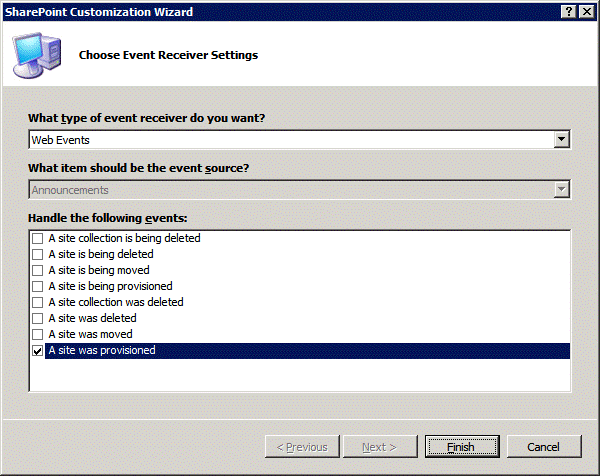
}

**Adding an Event Receiver to Brand Child Sites**

You should add one more project item to complete the Branding101 project. You still need a way to automatically apply your custom branding elements to child sites as they are created inside a site collection that has activated the Branding101 Feature. To do this in Windows SharePoint Services 3.0 or Office SharePoint Server 2007, you would have to use Feature stapling. However, SharePoint 2010 adds support for a new event named [WebProvisioned](http://msdn.microsoft.com/en-us/library/microsoft.sharepoint.spwebeventreceiver.webprovisioned.aspx), which makes the job much easier.

In **Solution Explorer**, right-click the **Branding101** project node, and on the **Project** menu, select **Add** and then **New Item**. In the **Add New Item** dialog box, create a new **Event Receiver** project item named ChildSiteInit.

When you create a new Event Receiver project item, the **SharePoint Customization Wizard** prompts you to select the type that you want. Select an Event Receiver of type **Web Events**. At the bottom of the dialog box, under **Handle the following events**, select the **A site was provisioned** option.

**Figure 10. SharePoint Customization Wizard**  
  


When you click **Finish**, the SharePoint development tools in Visual Studio 2010 add a new event handler named **WebProvisioned**, along with an Elements.xml file, which contains the **Receivers** element that is required to register this event handler in the current site collection. The benefit of adding this event is that it occurs each time a new child site is created. This makes it fairly easy to write the code that copies the relevant **SPWeb** properties from the top-level site to the new child site. The following shows the code that is required.

using System;

using Microsoft.SharePoint;

namespace Branding101.ChildSiteInit {

public class ChildSiteInit : SPWebEventReceiver {

public override void WebProvisioned(

SPWebEventProperties properties) {

SPWeb childSite = properties.Web;

SPWeb topSite = childSite.Site.RootWeb;

childSite.MasterUrl = topSite.MasterUrl;

childSite.CustomMasterUrl = topSite.CustomMasterUrl;

childSite.AlternateCssUrl = topSite.AlternateCssUrl;

childSite.SiteLogoUrl = topSite.SiteLogoUrl;

childSite.Update();

}

}

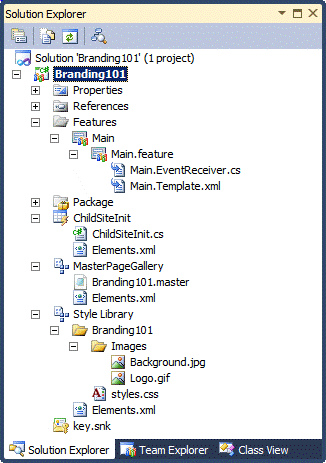
}

**Deploying and Testing**

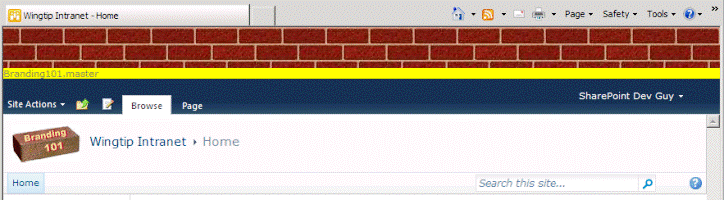
The development work on the Branding101 project is now finished. To summarize what was required, the SharePoint project was structured using four primary project items:

1. First, you added a **Module** with a custom master page and configured it to provision an instance of this master page in the master page gallery.
2. Second, you added a **Module** that targets the Style Library to provision an instance of a custom CSS file and multiple image files.
3. Third, you added a Feature receiver with code to enumerate all the sites within the current site collection and to configure the **SPWeb** properties for each site to link to your custom master page and your custom CSS file.
4. Finally, you added an event receiver to automatically initialize new child sites with the necessary branding characteristics.

Your final project should now be structured as the one shown in Figure 11.

**Figure 11. Final project in Solution Explorer**  
  


Now it is time to test your work. In **Solution Explorer**, right-click on the **Branding101** project node and then click **Deploy**. Visual Studio 2010 builds your project into a solution package named Branding101.wsp. Visual Studio then uploads the project's output solution package to the Solution Gallery of your test site and activates it. After the deployment finishes, you should be able to navigate to your test site and verify that it is now using your custom master page and your custom CSS file.

**Figure 12. Test site**  
  


Although your development work is finished, your design work is just beginning. Your challenge now is to edit your master page and CSS file to create a good-looking design. If you choose, you can continue to use Visual Studio 2010 to design your master page and CSS file.

As you edit the master page and the CSS file in Visual Studio 2010, it is easy to test your work. For example, experiment with making a small change to the HTML in Branding101.master or a CSS rule in styles.css. Next, run the **Deploy** command again and then refresh the current page in the browser to see the effects of your changes.

There is one more valuable design technique that you should consider: to integrate SharePoint Designer 2010 into your design efforts. For example, you can open your test site by using SharePoint Designer 2010 and then open the master page Branding101.master in advanced editing mode. The big advantage of using this approach is that SharePoint Designer 2010 offers a master page design experience for SharePoint sites that is much better than anything that is available in Visual Studio 2010. SharePoint Designer 2010 also offers a quite a few conveniences to the CSS editing experience.

One very important thing to understand is that SharePoint Designer does not let you directly edit the templates for master pages, such as Banding101.master, or the templates for CSS files, such as styles.css. Instead, when you are working in SharePoint Designer 2010 you are making customization changes to provisioned instances that are stored inside the content database. However, this does not pose much of a problem because you just have to know how to copy and paste from SharePoint Designer 2010 into Visual Studio 2010.

After you make the design customizations to a master page in SharePoint Designer 2010, you can just copy the contents of the customized master page in code view into the clipboard. Then, switch back over to Visual Studio 2010 and paste the clipboard contents into the template file for Branding101.master. You can use the same approach to copy the customized content of styles.css from SharePoint Designer and paste it into the template version of styles.css in Visual Studio 2010.

**Conclusion**

Now you truly have the best of both worlds. You can use the strength of SharePoint Designer 2010 to create a great-looking branding solution. You also have the means to package your branding efforts as a solution package that can be deployed as either a sandboxed solution or a farm solution in any farm that is running either SharePoint Foundation or SharePoint Server 2010.