Creating Consistent Looking Web Sites

Creating a Master Page

1. Open the Project.
2. Create the `MasterPages` folder, right-click the new folder, choose `Add New Item`, and select `Master Page`. Make sure that the master page uses Code Behind and that it is using your preferred programming language. Name the master page `Frontend.master`.
3. Add the following code between the `<form>` tags of the master page, replacing the `<div>` tags and the `ContentPlaceHolder` that VS added for you when you created the master. Note that this is almost the same code you added to `Default.aspx`, except for the `<asp:ContentPlaceHolder>` element and the `<a>` element within the Header `<div>`. The `<a>` element takes the user back to the home page, and will be styled later.

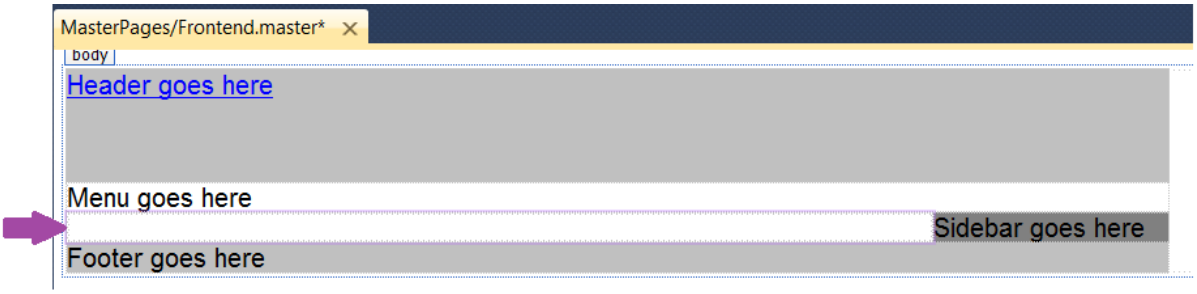
```
<form id="form1" runat="server">
<div id="PageWrapper">
    <div id="Header"><a href="~/" runat="server">Header goes here </a></div>
    <div id="MenuWrapper">Menu goes here</div>
    <div id="MainContent">
        <asp:ContentPlaceHolder id="cpMainContent" runat="server">

        </asp:ContentPlaceHolder>
    </div>
    <div id="Sidebar">Sidebar goes here</div>
    <div id="Footer">Footer goes here</div>
</div>
</form>
```

   Make sure that you have the `ContentPlaceHolder` within the `MainContent` `<div>` tags. You can drag one from the Toolbox onto the page or enter the code directly, using IntelliSense's helpful hints. In both cases you should give the control an `ID` of `cpMainContent`.
4. Next, switch the master page into Design View and then drag `Styles.css` from the Styles folder in the Solution Explorer onto the master page. As soon as you drop the file, VS updates the Design View and shows the layout for the site that you created. ***If***

the design doesn't change, switch to Source View and ensure there's a `<link>` tag in the head of the page pointing to your CSS file:

```
<head runat="server">
    <title></title>
    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
    <link href="../Styles/Styles.css" rel="stylesheet" type="text/css" />
</head>
```

The page should now look like:



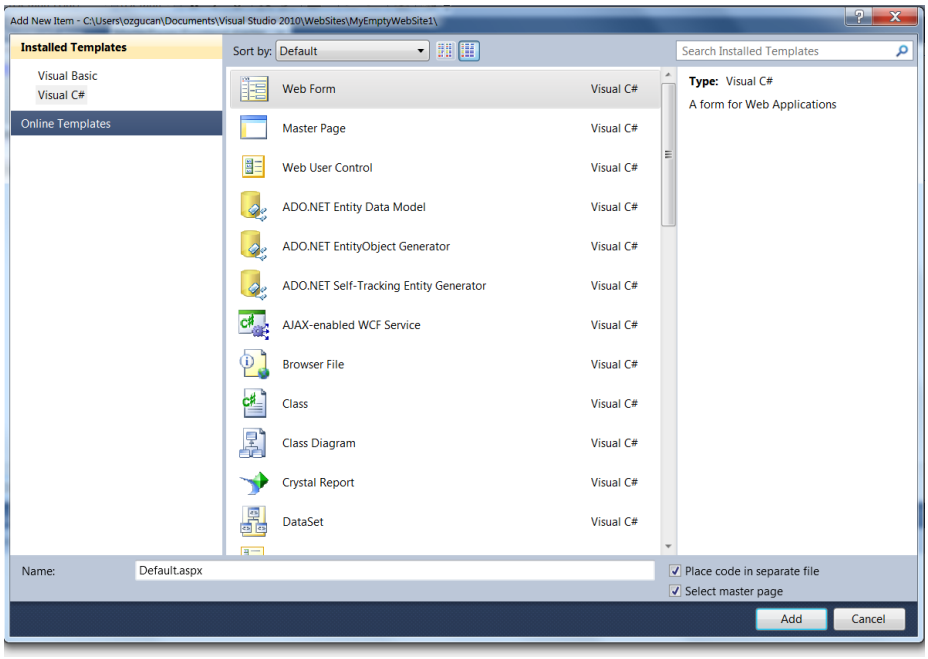
Note the area with the purple border around it between the menu and the footer region in your Design View. This is the `ContentPlaceHolder` control that is used by the content pages.

5. You can save and close the master page because you're done with it for now.

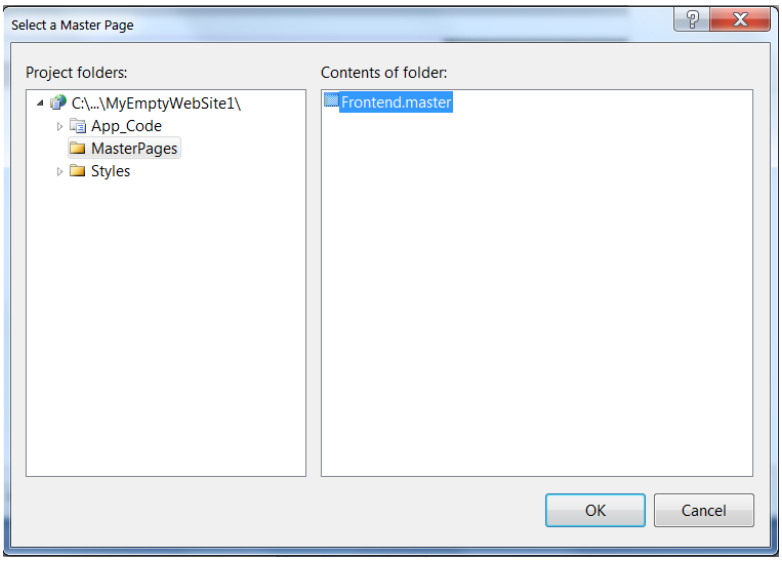
Adding a Content Page

1. In previous exercises you added standard ASPX pages to your project, which should now be "upgraded" to make use of the new master page. Because VS has no built-in support to change a standard page into a content page, you need to manually copy the content from the old ASPX page into the new one. If you want to keep the welcome text you added to `Default.aspx` earlier, copy all the HTML between the `MainContent` `<div>` tags to the clipboard (that is, the `<h1>` and the two `<p>` elements that you created earlier) and then delete the `Default.aspx` page from the Solution Explorer. Next, right-click the web site in the Solution Explorer and choose `Add New Item`. Select the correct programming language, click `Web Form`, name the page `Default.aspx`, and then, at the bottom of the dialog box, select the check

boxes for `Place Code in Separate File` and Select `Master Page`. Finally, click the Add button.



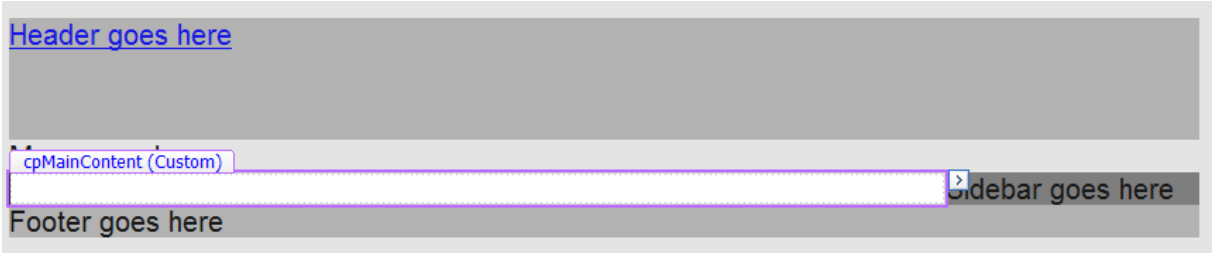
2. In the `Select a Master Page` dialog box, click the `MasterPages` folder in the lefthand pane, and then in the area at the right, click `Frontend.master`. Click OK to add the page to your web site.



Instead of getting a full page with HTML as you got with standard ASPX pages, you now only get two `<asp:Content>` placeholders as shown:

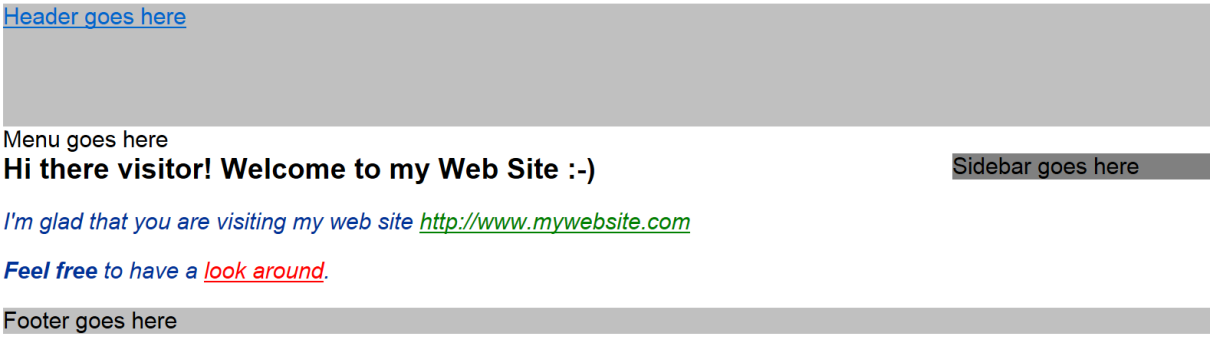
```
<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" Runat="Server">
</asp:Content>
```

3. Switch to Design View and note that everything is grayed out and read-only, except for the `<asp:Content>` region for `cpMainContent`.



4. If you still have the old markup from the `Default.aspx` on the clipboard, click once inside the `cpMainContent` placeholder and press Ctrl+V. (Note: you can do this both in Design View and in Markup View). This adds the markup to the page, right between the `<asp:Content>` tags.

5. Save your changes by pressing Ctrl+S and press Ctrl+F5 to open the page in your browser. The browser should display the page very closely to what you saw in Design View.



6. Now take a look at the HTML for the page in the browser. You can do this by right-clicking the page and choosing View Source or View Page Source. **Note** that the source of the final page in the browser is a combination of the source of the master page and the content page:

```
<div id="PageWrapper">
    <div id="Header"><a href="./">Header goes here </a></div>
    <div id="MenuWrapper">Menu goes here</div>
    <div id="MainContent">

    <h1 style="padding: 0px; margin: 0px 0px 10px 0px">
        Hi there visitor! Welcome to my Web Site :-)
    </h1>

    <p class="Introduction">
        I&#39;m glad that you are visiting my web site <a href="http://www.mywebsite.com">
        http://www.mywebsite.com</a>
    </p>
    <p class="Introduction">
        <strong>Feel</strong> <span class="style1"><strong>free</strong></span> to have a
        <a href="DefaultWcss2.aspx">look around</a>.
    </p>

    </div>
    <div id="Sidebar">Sidebar goes here</div>
    <div id="Footer">Footer goes here</div>
</div>
```

The first four lines come from the master page and lines of HTML code come from the content page.

7. Switch back to VS and create a new page called `Login.aspx` in the root of the site based on the master page. Notice how VS remembered your last settings with regard to the master page and Code Behind (make sure both are checked in case you unchecked them earlier). Switch to Source View and create an `<h1>` element inside the `cpMainContent` placeholder with the text **`Log in to My Website`**.

There's no need to add any other controls to this page just yet, but it serves as the basis for the login functionality you'll create later. Without any content in the `MainContent` element, the `Sidebar` will be moved to the left of the page.

8. Go back to `Default.aspx` and switch to Design View. Beneath the welcome text with the header and two `<p>` elements, create a new paragraph (press Enter in Design View) and type some text (for example, **`You can log in here`**).

Notice how this new paragraph has a `class` attribute called `Introduction` as VS applies the previous class to new paragraphs automatically.

```
<p class="Introduction">
    You can log in here.</p>
```

Remove this class using the `Clear Styles` option of the `Apply Styles` window, or manually remove it from the code in Source View.

```
<p>
    You can log in here.</p>
```

9. Highlight the words "`log in`" and choose Format ➪ Convert to Hyperlink from the main menu. In the dialog box that follows, click the Browse button and select the `Login.aspx` page that you just created. Click OK twice.
10. Save all changes and press Ctrl+F5 again to view `Default.aspx` in the browser. Then click the link you created in the preceding step. You should now be taken to `Login.aspx`. **Note** that the general layout, like the header and the sidebar, is

maintained. The only thing that changes when you go from page to page is the content in the main content area.

Creating a Base Page

1. Right-click the `App_Code` folder in the Solution Explorer and choose `Add New Item`. Select `Class` in the `Templates` list and name the file **`BasePage`**. You can choose any name you like but `BasePage` clearly describes the purpose of the class, making it easier to understand what it does.
2. Clear the contents of the file, and then add the following code:

```
public class BasePage:System.Web.UI.Page
{
    private void Page_PreRender(object sender, EventArgs e)
    {
        if (this.Title == "Untitled Page" || string.IsNullOrEmpty(this.Title))
        {
            throw new Exception("Page Title can not be \"Untitled Page\" or an empty string");
        }

    }

    public BasePage()
    {
        //
        // TODO: Add constructor logic here
        //

        this.PreRender += new EventHandler(Page_PreRender);
    }
}
```

3. Save the file and close it, and then open the `Login.aspx` page that you created earlier. Open its Code Behind file and change the colon `[:]` so the login page inherits from the `BasePage` class you created earlier:

```
public partial class Login : BasePage
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
}
```

4. Save the page and then request it in the browser by pressing Ctrl+F5. If you haven't changed the title of the page earlier, you should be greeted by the error shown in the browser.

   Instead of this generic error, you may see an error that displays the source for the `BasePage` class where the title is checked.

5. Go back to VS and open the login page in Source View. Locate the `Title` attribute in the `@Page` directive (or add one if it isn't there) and set its value to **`Log in to My WebSite`**.

```
<%@ Page Title="Log in to my WebSite" Language="C#" MasterPageFile="~/MasterPages/Frontend.master"
```

6. Repeat steps 3 and 5 for all the pages in your site. To make this a bit quicker, you can use Find and Replace to quickly replace all the occurrences of

`System.Web.UI.Page` with `BasePage`. **Make sure** you don't accidentally replace it in the `BasePage` file in the `App_Code` folder itself. To prevent this from happening, make sure you search only in Code Behind files, like this:

➤➤Open the Replace in Files dialog box (press Ctrl+Shift+H or select Edit ➪ Find and Replace ➪ Replace in Files).

➤➤ In the Find What box enter `System.Web.UI.Page`. In the Replace With text box enter `BasePage`.

➤➤ Expand the Find Options section and in the Look at These File Types text box enter `*.aspx.cs` depending on the language you use. This leaves the `BasePage` file, which has a single extension of `.cs`.

➤➤ Click Replace All and then click `Yes` to confirm the Replace operation.

7. Save the changes you made to any open page and then browse to `Login.aspx` again. If everything worked out as planned, the error should be gone and you now see the `Login` page.

Remember, though, that all other pages in your site now throw an error when you try to Access them. The fix is easy; just give them all a valid `Title`. For pages without a `Title` attribute in their page directive, you need to do this manually. For other pages, with an empty `Title=""` attribute, you can quickly do this by searching the site for `Title=""` and replacing it with something like `Title="My WebSite"`. (Don't forget to reset the Look At These File Types back to `*.*`).

Creating a Reusable Page Template

1. Add a new Web Form to the site and call it `Temporary.aspx`. Make sure it uses Code Behind, uses your programming language, and is based on the master page in the `MasterPages` folder.
2. Open the Code Behind of this new page (by pressing F7) and change the colon so the page inherits from `BasePage` instead of from `System.Web.UI.Page`. Also rename the class from `Temporary` to `$relurlnamespace$_$safeitemname$`:

```
public partial class $relurlnamespace$_$safeitemname$ : BasePage
{
    protected void Page_Load(object sender, EventArgs e)
    {

    }
}
```

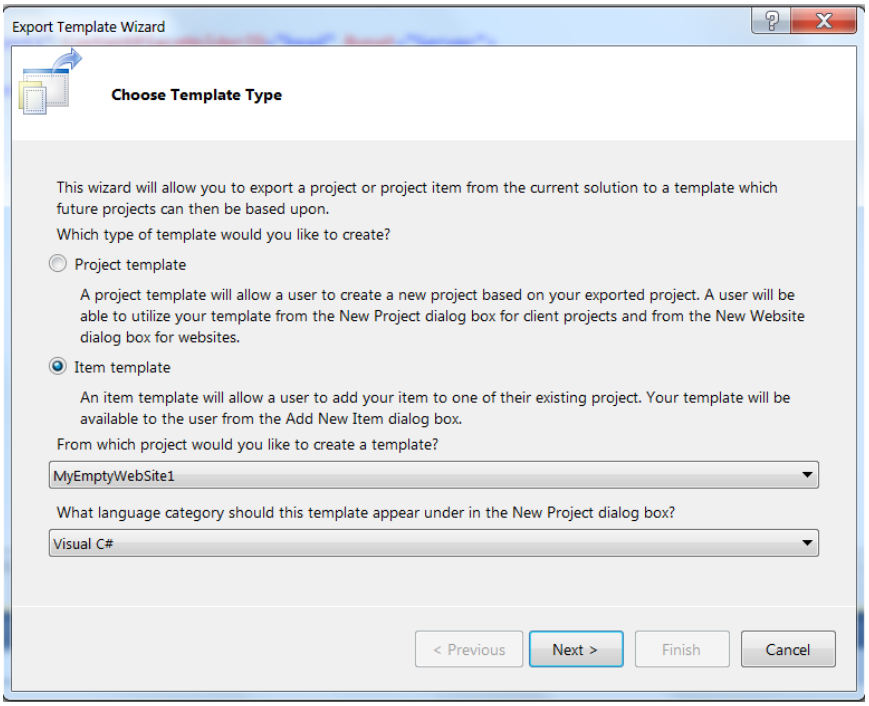
   Make sure you don't remove any of the existing code, like the using statements or the `Page_Load` method.

   Don't worry about any compile errors you may get about unexpected characters like `$`. Once you start adding pages based on this template, `$relurlnamespace$_$safeitemname$` will be replaced by the name of the page you're adding.
3. Switch to Source View, and change the Inherits attribute from `Temporary` to `$relurlnamespace$_$safeitemname$`:

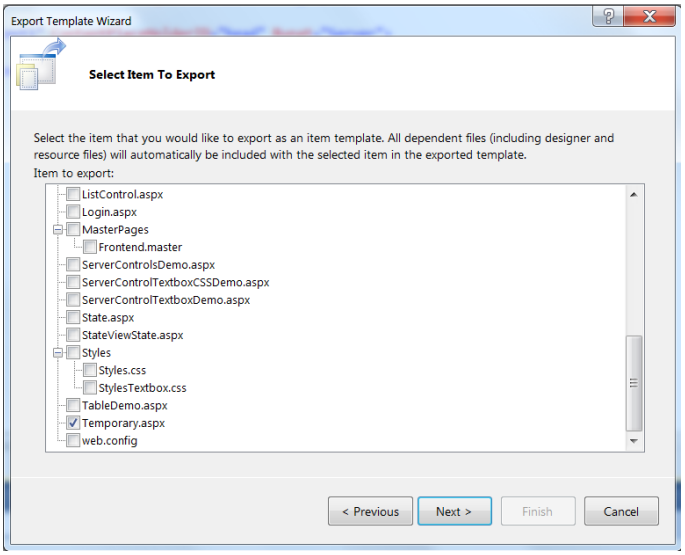
```
<%@ Page Title="" Language="C#" MasterPageFile="~/MasterPages/Frontend.master"
    AutoEventWireup="true" CodeFile="Temporary.aspx.cs" Inherits="$relurlnamespace$_$safeitemname$" %>
```

   You can leave the `CodeFile` attribute alone; VS will change it to the right Code Behind file automatically whenever you add a new page to the site.
4. Optionally, add other code you want to add to your pages by default, like a comment block with a copyright notice.

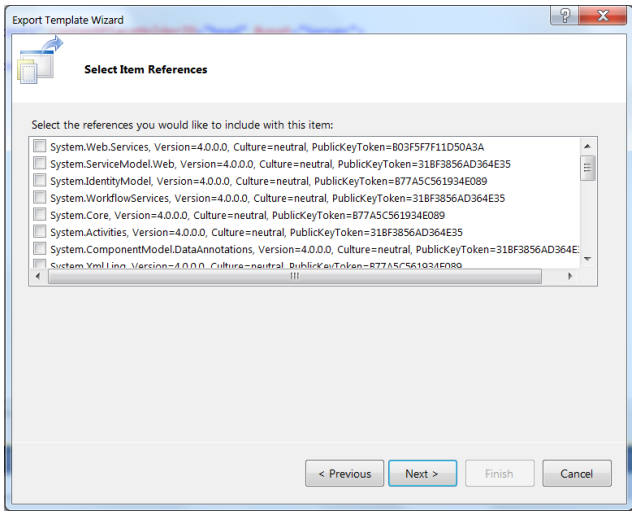
5. Save all changes and then choose File ➪ Export Template. In the dialog box that follows, select `Item Template` and choose your programming language from the drop-down list at the bottom of the screen.



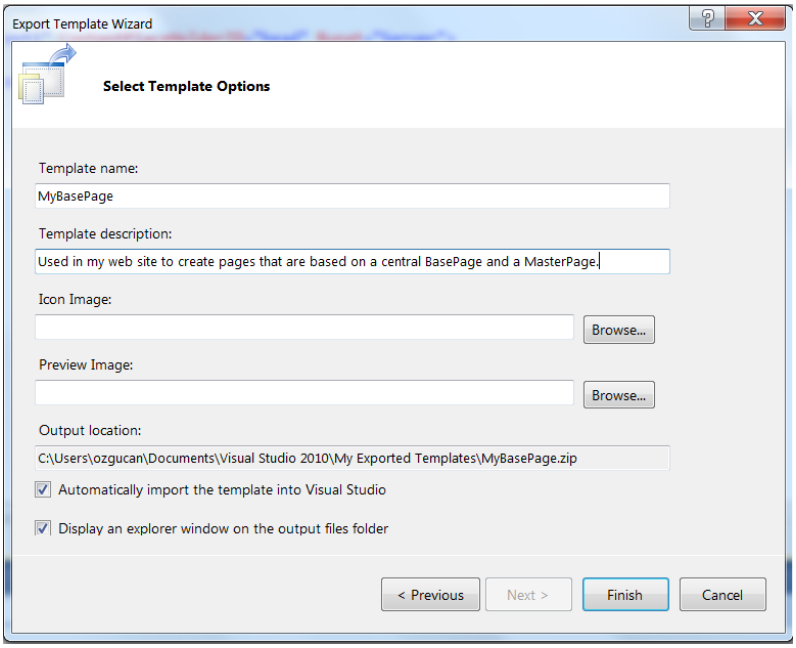
6. Click Next and place a check mark in front of `Temporary.aspx` that you find near the bottom of the list. Click Next again to go to the `Select Item References` dialog box.



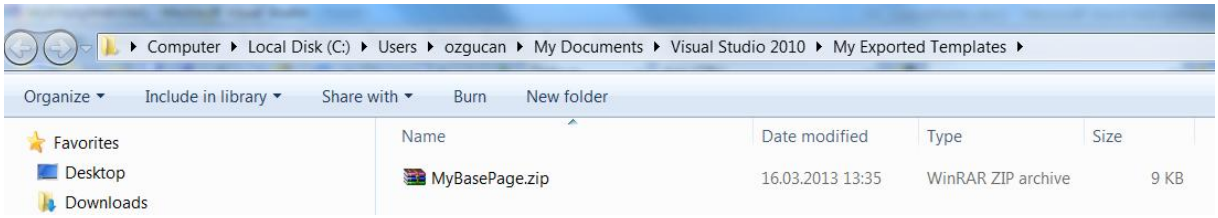
7. There is no need to set anything in the Select Item References dialog box now. If you had a web page referencing specific assemblies (`.dll` files) you could pick them here, so VS adds the references for you automatically next time you add a file based on this template.



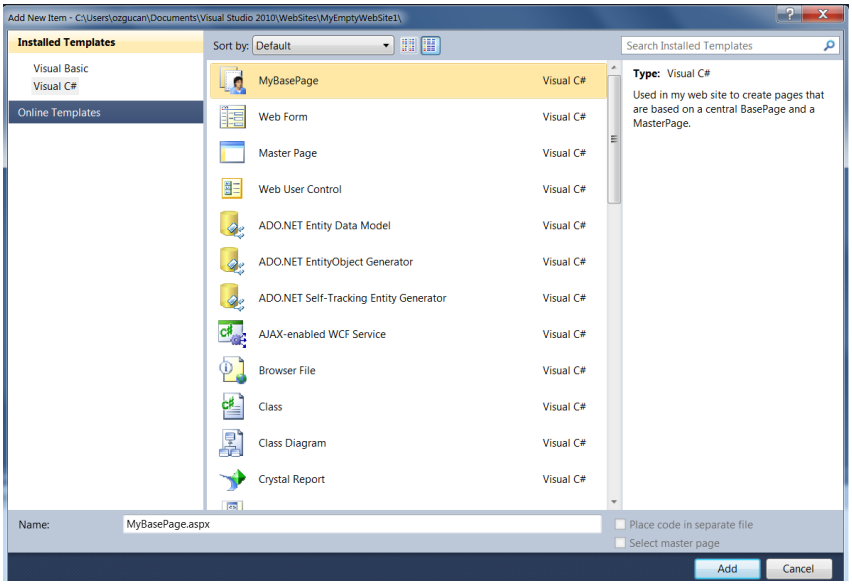
In this case, click Next again to go to the `Select Template Options` screen. Type `MyBasePage` as the new template name, and optionally type a short note describing the purpose of the template.



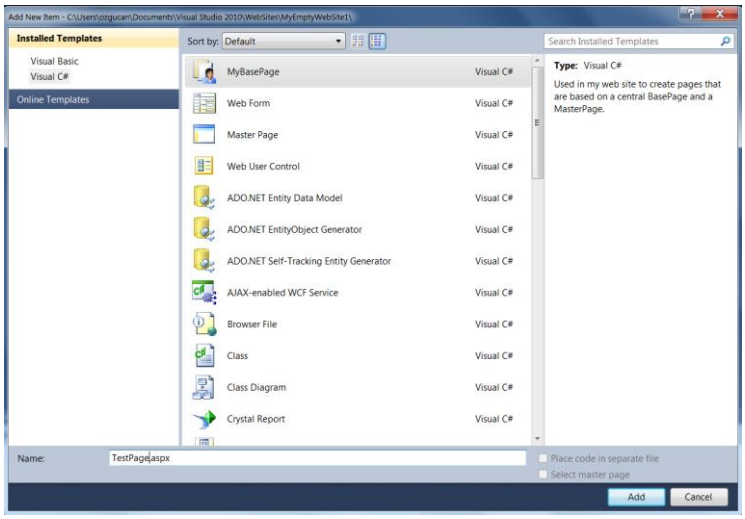
8. Click Finish to create the template. VS opens a Windows Explorer showing the new template as a ZIP file. You can close that window, because you don't need it.



9. Back in VS, delete the temporary file `Temporary.aspx` you created. Then right-click the web site in the Solution Explorer and choose `Add New Item`. Note that your custom template now shows up in the list of templates.



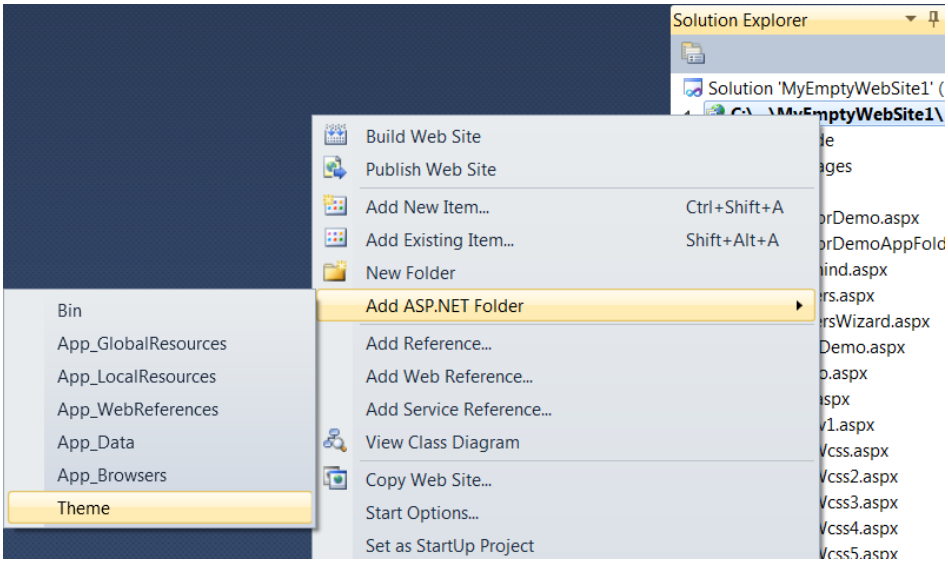
10. Type a new name for the page such as `TestPage.aspx` and click Add to add it to your site. Look at the markup and the Code Behind of the file and verify that `$relurlnamespace$_$safeitemname$` has been renamed to `_TestPage` to reflect the new name of the page. If everything looks OK, you can delete `TestPage.aspx` because it's not used in the web site.



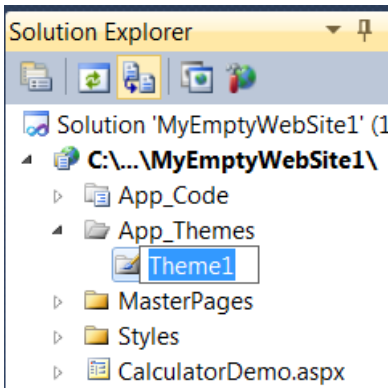

```
<%@ Page Title="" Language="C#" MasterPageFile="~/MasterPages/Frontend.master" AutoEventWireup="true"
    CodeFile="TestPage.aspx.cs" Inherits="_TestPage" %>
```

Creating a New Theme for Your Web Site

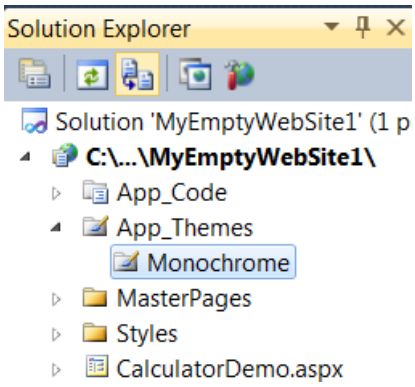
1. Add the special `App_Themes` folder to your web site. To do this, rightclick the web site in the Solution Explorer and choose Add ASP.NET Folder ➪ Theme.



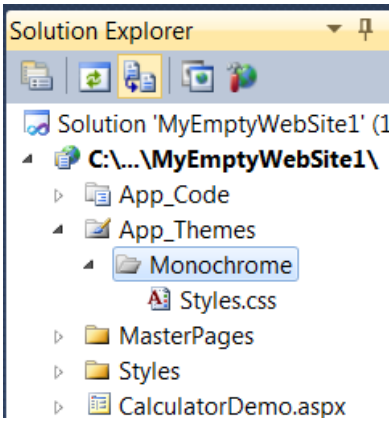
This not only creates the `App_Themes` folder, but immediately creates a subfolder for the theme called `Theme1` by default.



Type **`Monochrome`** as the new name instead.



2. From the `Styles` folder, move the `Styles.css` file into this new `Monochrome` folder. You can either drag it directly into the new folder or use Ctrl+X to cut the file, click the `Monochrome` folder, and press Ctrl+V to paste it again. You can leave the empty `Styles` folder because it's used again later.



3. To make it clear later where your CSS is coming from, rename the file from `Styles.css` to `Monochrome.css`.

4. Because the main layout is now going to be controlled by the theme, you no longer need the `<link>` element in the `<head>` section of the master page pointing to the old CSS file, so you can remove it. To this end, open the master page, switch to Source View, and remove the `<link>` element in the `<head>` section of the master page from the code:

```
<head runat="server">
    <title></title>
    <asp:ContentPlaceHolder id="head" runat="server">
    </asp:ContentPlaceHolder>
    <link href="../Styles/Styles.css" rel="stylesheet" type="text/css" />
</head>
```

5. The next step is to apply the theme to the entire web site. Open the `web.config` file from the root of the site and directly inside the `<system.web>` element, add a `<pages>` element with the theme attribute:

```
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.0"/>
    <pages theme="Monochrome"></pages>
  </system.web>
</configuration>
```

6. To test the theme, save all your changes and then request the `Default.aspx` page in your browser. The design of the site should be identical to how it was.

NOTE Change `Title` of `Default.aspx`.

```
<%@ Page Title="Welcome!"
```

Instead of linking to the CSS file from the master page, the CSS is now included in the page source through the theme set in the `web.config` file. Open the HTML source of the page in the browser. At the top you should see the following code:

```
<head><title>
	Welcome!
</title>
<link href="App_Themes/Monochrome/Monochrome.css" type="text/css" rel="stylesheet" /></head>
```

Note that a link to the style sheet from the `Monochrome` theme folder is injected in the `<head>` of the page. The ASP.NET runtime does this for every CSS file it finds in the currently active theme folder, so be sure to keep your theme folder clean to avoid unnecessary files from being included and downloaded by the browser. Also note that the `<link>` is added just right before the closing `</head>` tag. This ensures that the theme file is included after all other files you may have added yourself (through the master page, for example). This is in contrast to how the `styleSheetTheme` attribute works. Because this type of theme allows its settings to be overridden, it's imported at the top of the file, giving room for other CSS files that follow it to change the look and feel of the page.

7. Return to Visual Studio and open the master page file in Design View. Notice how all the design is gone and VS now shows the basic layout of the page again. Unfortunately, VS does not display the `theme` you've set using the theme attribute. However, you can overcome this limitation by setting the `styleSheetTheme` instead. To do this, open the `web.config` file again, locate the `<pages>` element you created earlier, and add the following attribute:

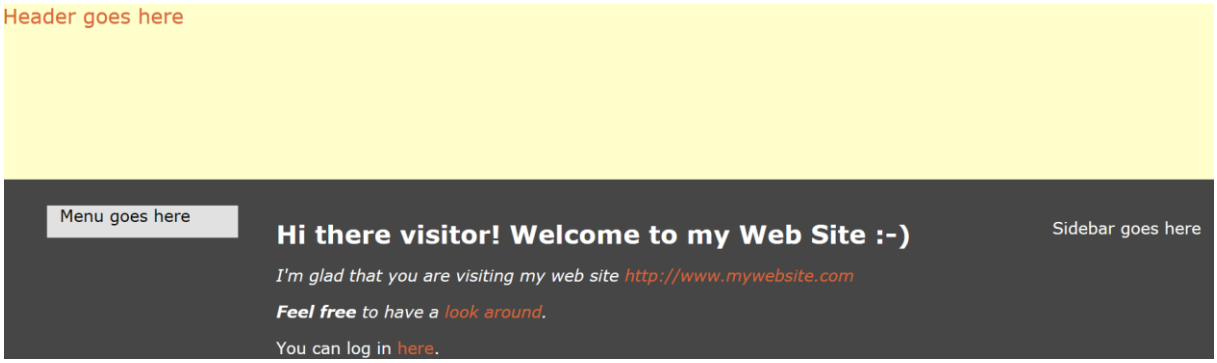
```
<pages theme="Monochrome" styleSheetTheme="Monochrome"></pages>
```

8. Save the changes to `web.config`, close and reopen the master page, and switch to Design View. You'll see that VS now applies the correct styling information to your pages.

9. To add another theme to the site, create a new folder under `App_Themes` and call it `DarkGrey`. Copy `DarkGrey.css` from Windows Explorer into the `DarkGrey` theme folder in VS.
10. Open the `web.config` file once more and change both occurrences of `Monochrome` to `DarkGrey` in the `<pages>` element.

```
<pages theme="DarkGrey" styleSheetTheme="DarkGrey"></pages>
```

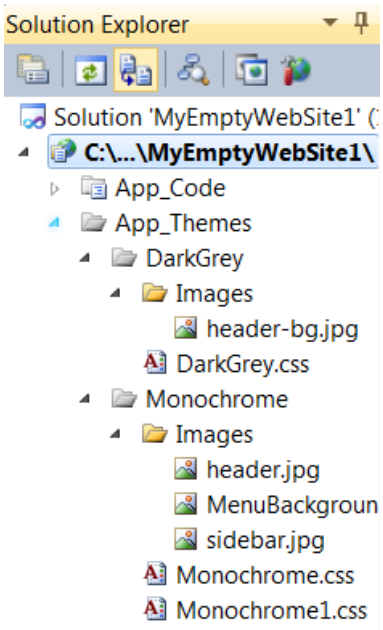
Save the changes again and press Ctrl+F5. Instead of the purple `Monochrome` theme, you'll now see the site with the `DarkGrey` theme.



If you don't see the menu, the main content and the sidebar all next to each other, make sure your browser window is wide enough to display all content.

Adding Images to Your Theme

1. Open Windows Explorer and navigate to the files where you can find `Images` folder and `Monochrome.css`. Select the `Images` folder and the `Monochrome.css` file.
2. Drag (or copy and paste) the selected folder and files from Windows Explorer into the `Monochrome` theme folder in VS. Click Yes when you're asked to overwrite `Monochrome.css`.
3. Repeat steps 1 and 2, but this time drag (or copy and paste) only the `Images` folder from the Windows Explorer's `DarkGrey` folder into the `DarkGrey` theme folder in VS.

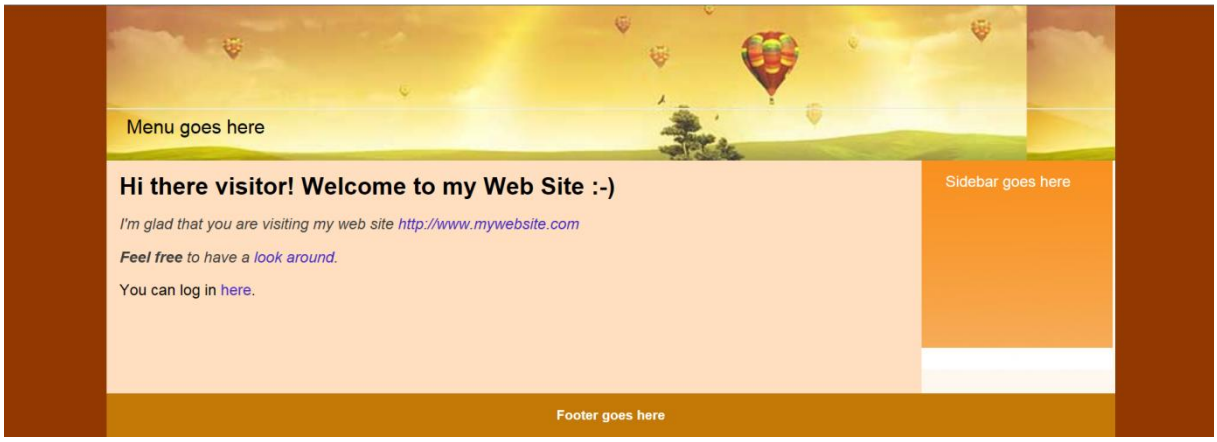


4. Open up the master page from the `MasterPages` folder and remove the text `Header Goes Here` between the `<a>` tags. Because the header will be filled with an image, you no longer need this text.

```
<div id="Header"><a href="~/App_Themes/DarkGrey/Images/header.jpg" runat="server"> </a></div>
```

5. Request the `Default.aspx` page in your browser by right-clicking it and choosing View in Browser. You should now see the web page with images from the `DarkGrey` theme.

6. Go back to VS, open the `web.config` file, and switch the two theme attributes of the `<pages>` element from `DarkGrey` to `Monochrome` again. Open `Default.aspx` in your browser and you'll see the page with the new theme and images.

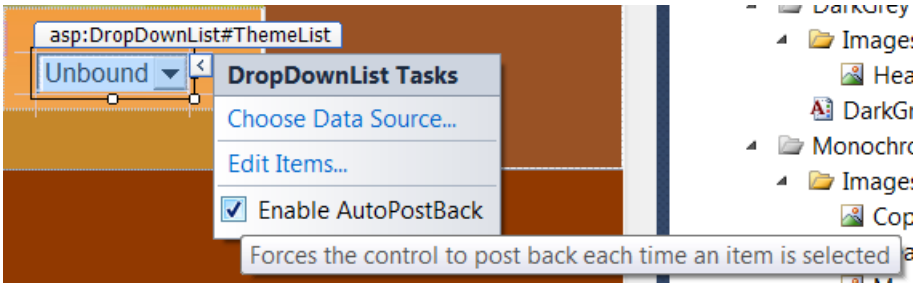


Letting the User Select a Theme

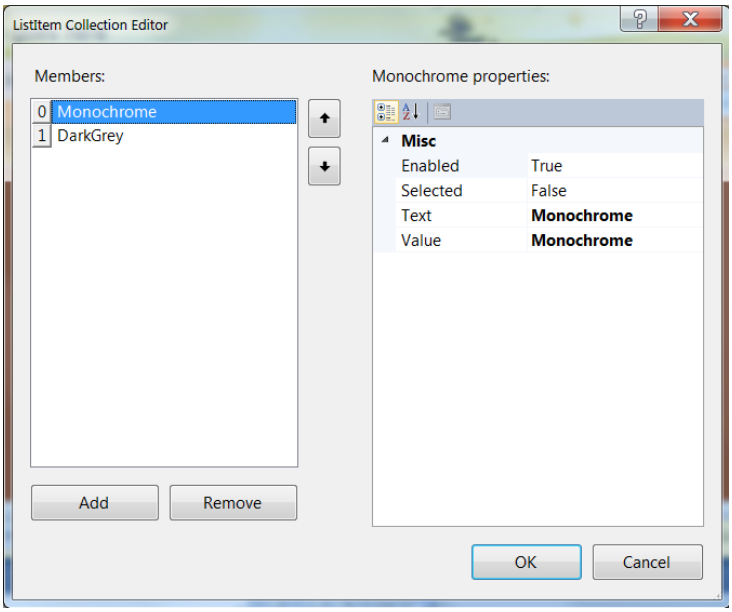
1. Open the master page in Source View and locate the `<div>` called `Sidebar`. Remove the static text `Sidebar Goes Here` and replace it with a `DropDownList` control by dragging it from the Toolbox between the two div tags. Change the `ID` of the control from `DropDownList1` to `ThemeList`. Type some text (for example, `Select a Theme`) followed by a line break (`<br />`) in front of the drop-down list to clarify the purpose of the list.

```
<div id="Sidebar">Select a Theme<br />
    <asp:DropDownList ID="ThemeList" runat="server">
    </asp:DropDownList>
</div>
```

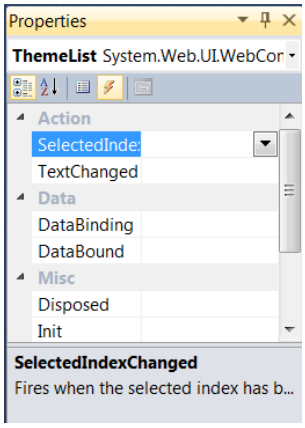
2. Switch to Design View, open the control's `Smart Tasks` panel, and select `Enable AutoPostBack`.



3. On the same Smart Tasks panel, click the `Edit Items` link and insert two items: one with the text `Monochrome` and one with the text `DarkGrey`.



4. Double-click the drop-down list to set up an event handler for the `SelectedIndexChanged` event. Instead of double-clicking, you can also select the `DropDownList`, press `F4` to open its `Properties Grid`, click the button with the lightning bolt to switch to the `Events` tab, and double-click `SelectedIndexChanged`.



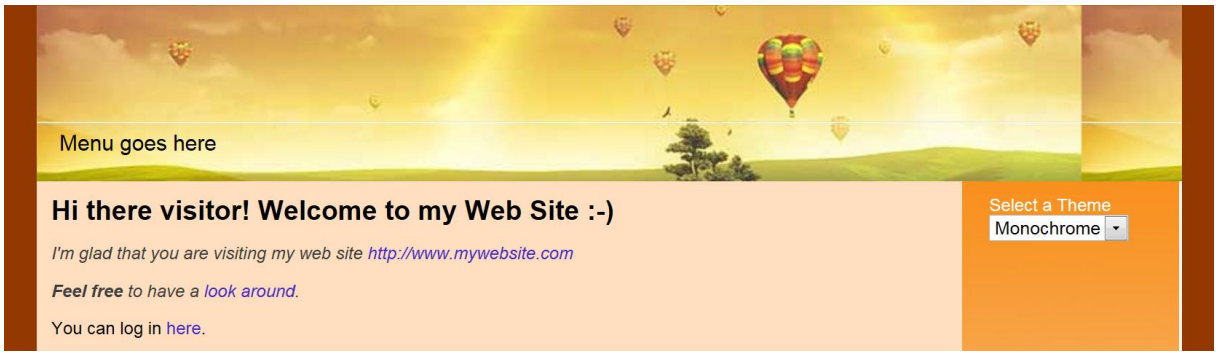
Any code you write in the `SelectedIndexChanged` handler fires at the server when the user makes a new selection in the drop-down list at the client. Within the handler block, add the following code that retrieves the selected theme from the list and stores it in a cookie:

```
protected void ThemeList_SelectedIndexChanged(object sender, EventArgs e)
{
    HttpCookie preferredTheme = new HttpCookie("PreferredTheme");
    preferredTheme.Expires = DateTime.Now.AddMonths(3);
    preferredTheme.Value = ThemeList.SelectedValue;
    Response.Cookies.Add(preferredTheme);
    Response.Redirect(Request.Url.ToString());
}
```

5. Still in the Code Behind of the master page, you need to add some code that preselects the correct item in the list again when the page loads. The best place to do this is in the `Page` class's `Load` event. Within the `Page_Load` handler block, add the following code:

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        string selectedTheme = Page.Theme;
        HttpCookie preferredTheme = Request.Cookies.Get("PreferredTheme");
        if (preferredTheme != null)
        {
            selectedTheme = preferredTheme.Value;
        }
        if(!string.IsNullOrEmpty(selectedTheme) && ThemeList.Items.FindByValue(selectedTheme)!=null)
        {
            ThemeList.Items.FindByValue(selectedTheme).Selected = true;
        }
    }
}
```

6. Save all changes and then request `Default.aspx` in your browser again. The drop-down list in the sidebar should display the first item in the list as selected. Select the other option from the list and the page will reload. The item you chose last in the drop-down list should now be preselected in the drop-down list. **Notice** that you keep seeing the same theme because you haven't written any code yet that applies the selected theme.



Applying the User Selected Theme

1. Open the `BasePage` file from the `App_Code` folder and add the following code that sets the selected theme during the `PreInit` event. You can add this code before or after the method that checks the page title.

```
private void Page_PreInit(object sender, EventArgs e)
{
    HttpCookie preferredTheme = Request.Cookies.Get("PreferredTheme");
    if (preferredTheme != null)
    {
        Page.Theme = preferredTheme.Value;
    }
}
```

2. You need to set up an event handler in the class's constructor for the `PreInit` event, just as you did with the `PreRender` event handler. This tells the ASP.NET runtime which method will handle the `PreInit` event:

```
public BasePage()
{
    //
    // TODO: Add constructor logic here
    //

    this.PreRender += new EventHandler(Page_PreRender);
    this.PreInit += new EventHandler(Page_PreInit);
}
```

3. Save changes to all open documents and then request `Default.aspx` in the browser. The page should load with the theme you chose last in the drop-down list in the previous exercise.
4. Choose a new item from the list. The page should reload and should now show the other theme.

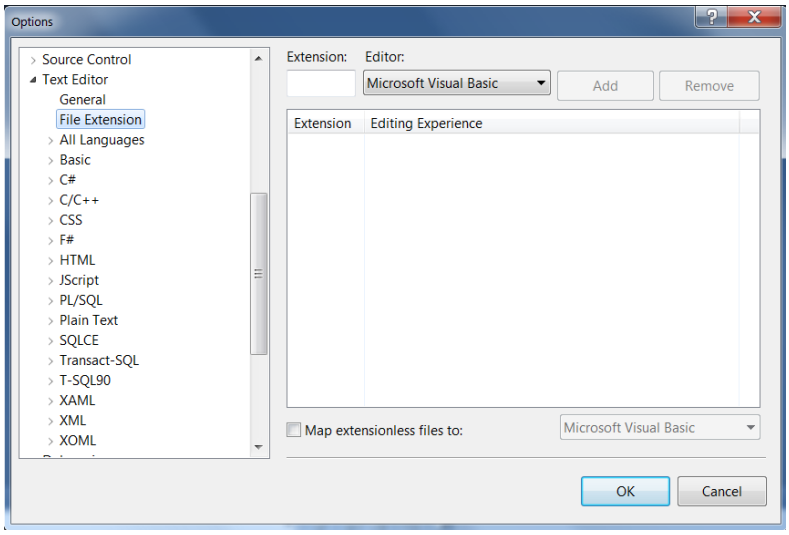
   If you find that the page in the browser is showing a combination of the two themes, go back to VS, open `web.config`, and remove the `styleSheetTheme` attribute from the `<pages>` element, leaving the `theme` attribute in place because it serves as the default for new visitors.

Creating a Skin File

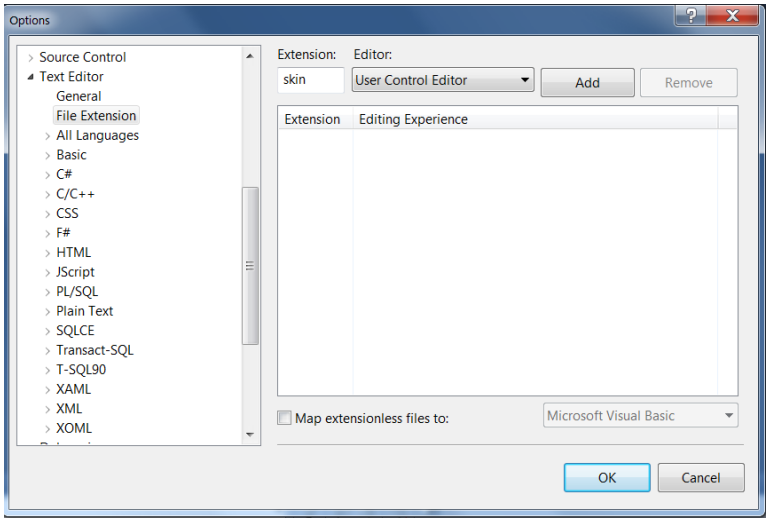
Skin files must be created in the theme's folder directly. You can't store them in a subfolder like you do with the theme's images.

When you start typing in a skin file, you'll notice that the familiar IntelliSense doesn't kick in. This makes it slightly difficult to define your controls and their attributes. However, there is a simple workaround:

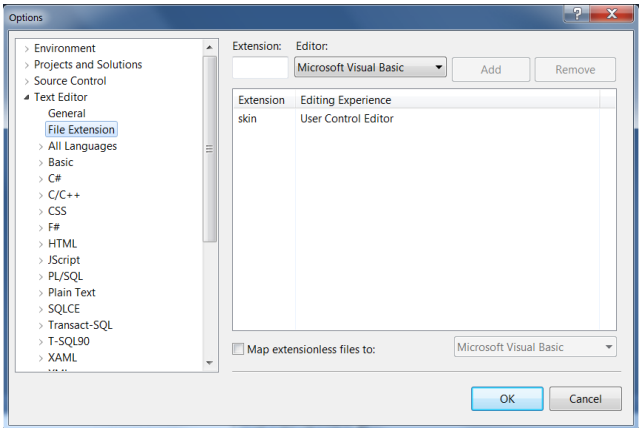
1. Open VS's Options dialog box by choosing Tools ➪ Options.
2. Expand the `Text Editor` category and click `File Extension`. If you don't see these categories, make sure that `Show All Settings` at the bottom of the screen is selected.



3. In the `Extension` box type **`skin`** and then from the Editor drop-down list choose `User Control Editor`.



4. Click the Add button and then click the OK button to dismiss the Options dialog box.

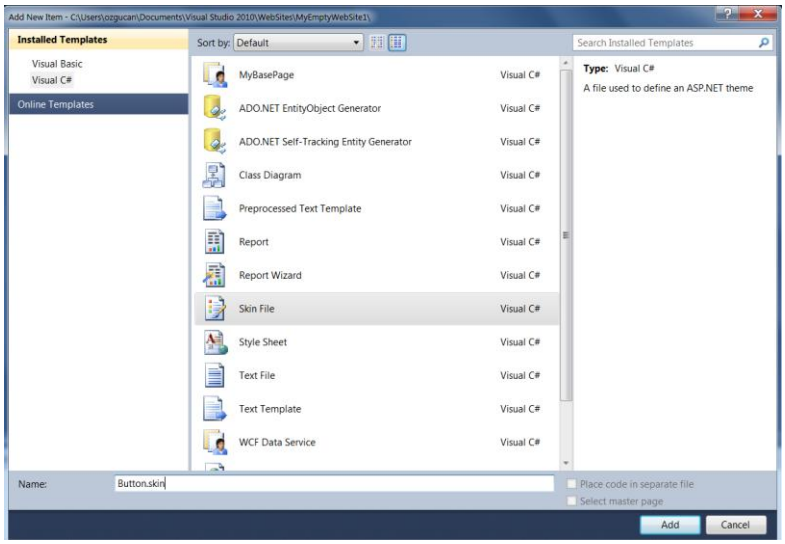


From now on, you'll get IntelliSense in skin files (you may need to reopen existing skin files first if you already created one). With this setting on, you may get a warning in the Error List about build providers when you have a skin file open. You can safely ignore this error, because skins work fine at runtime even with these settings in VS.

Creating a Skin for the Button Control

To effectively use skins, you should strive to use `CssClass` attributes as much as possible instead of applying inline attributes that all end up in the final HTML of the page, increasing its size and load time. However, to show you how it works in case you do have a special need to add inline attributes, this exercise shows you how to apply both.

1. In the `Monochrome` theme folder, add a new skin file and call it `Button.skin`. You add the file by right-clicking the `Monochrome` folder and choosing `Add New Item`. In the dialog box that follows select `Skin File` and type `Button` as the file name.



2. Delete the entire contents from the file and type the following code:

```
<asp:Button CssClass="MyButton" Backcolor="#83B715" runat="server" />
```

Note that this markup uses a combination of inline attributes for styling (the `BackColor`) and the `CssClass` to point to a selector in your CSS file. As explained earlier, you can ignore the warning about missing build providers because your skin files will work fine at runtime. As soon as you close the skin file, the warning goes away.

3. Open the `Monochrome.css` file from the theme folder and add this CSS selector at the end of the file:

```
.MyButton
{
    color:White;
}
```

4. Create a new Web Form in the `Demos` folder and call it `SkinsDemo.aspx`. Make sure you base it on the exported template you created earlier. Give the page a `Title` of `Skins Demo` and then add a `Button` by dragging it from the Toolbox into the `cpMainContent` area of the page. You end up with this code:

```
<%@ Page Title="Skins Demo" Language="C#" MasterPageFile="~/MasterPages/Frontend.master"

<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" Runat="Server">
    <asp:Button ID="Button1" runat="server" Text="Button" />
</asp:Content>
```

5. Save all changes and then request `SkinsDemo.aspx` in the browser. If necessary, switch to the `Monochrome` theme. The button you added in step 4 should now have a green background with white text on it. If the changed colors don't show up, make sure you selected the right theme in the drop-down list and that you added the `MyButton` CSS class to the CSS file of the `Monochrome` theme. If you still don't see the changes, press Ctrl+F5 or Ctrl+R to force a fresh copy of the CSS file from the server.

Creating a Named Skin for the Button Control

The easiest way to create a named skin is by copying the code for an existing one and then adding a `SkinID` attribute. Be aware that if you copy and paste a skin definition, VS automatically adds an `ID` attribute (that is, if you connected skin files to the User Control Editor as described earlier). This `ID` is not allowed, so you need to remove it.

1. Open `Button.skin` again, copy all the code, and paste it below the existing markup.
2. If VS added an `ID` attribute, remove it, together with its value (for example, remove `ID="Button1"`).
3. Remove the `CssClass` attribute and its value, change the `BackColor` of the button to `Red`, and set the `ForeColor` to `Black`.
4. Add a `SkinID` attribute of `MaroonButton`. You should end up with this code:

```
<asp:Button  Backcolor="Maroon" ForeColor="Orange" SkinID="MaroonButton" runat="server" />
```

5. Save and close the skin file.
6. Open `SkinsDemo.aspx` and add a second button. Set the `SkinID` of this button to `MaroonButton`. Notice how IntelliSense helps you pick the right `SkinID`. The code for the two buttons should now look like this:

```
<asp:Content ID="Content2" ContentPlaceHolderID="cpMainContent" Runat="Server">
    <asp:Button ID="Button1" runat="server" Text="Button" />
    <asp:Button ID="Button2" runat="server" Text="Button" SkinID="MaroonButton"/>
</asp:Content>
```

7. Open `SkinsDemo.aspx` in the browser. You should now see two buttons with different colors. If you don't see the different colors, ensure you have selected the `Monochrome` theme in the browser.