7

Themes and Skins

COPYRIGHTED MATERIAL

When you build a Web application, it usually has a similar look and feel across all its pages. Not too many applications are designed with each page dramatically different from the next. Generally, for your applications you use similar fonts, colors, and server control styles across all the pages.

You can apply these common styles individually to each and every server control or object on each page, or you can use a new capability provided by ASP.NET 2.0 to centrally specify these styles. All pages or parts of pages in the application can then access them.

Themes are the text-based style definitions in ASP.NET 2.0 that are the focus of this chapter.

Using ASP.NET 2.0 Packaged Themes

Themes are similar to Cascading Style Sheets (CSS) in that they enable you to define visual styles for your Web pages. Themes go further than CSS, however, in that they allow you to apply styles, graphics, and even CSS files themselves to the pages of your applications. You can apply ASP.NET themes at the application, page, or server control level.

To make life easy for the developer, ASP.NET comes with free prepackaged themes that you can use for your pages or applications. You find these themes located at `C:\WINDOWS\Microsoft .NET\Framework\v2.0.xxxxx\ASP.NETClientFiles\Themes`. The available themes that come with ASP.NET 2.0 include

- ❑ BasicBlue
- ❑ SmokeAndGlass

Applying a theme to a single ASP.NET page

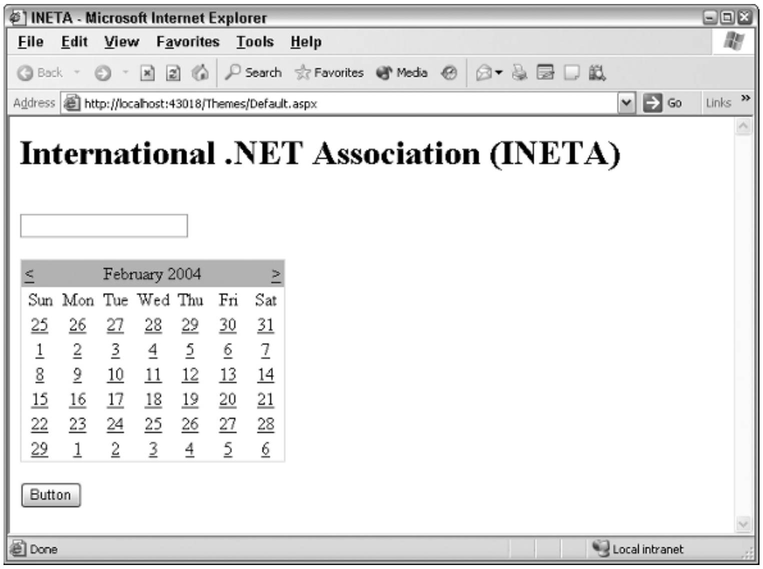
In order to see how to use one of these themes, create a basic page, which includes text, a text box, a button, and a calendar. This is shown in Listing 7-1.

Listing 7-1: An ASP.NET page that does not use themes

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>INETA</title>
</head>
<body>
    <form id="form1" runat="server">
        <h1>International .NET Association (INETA)</h1><br />
        <asp:Textbox ID="TextBox1" Runat="server" /><br />
        <br />
        <asp:Calendar ID="Calendar1" Runat="server" /><br />
        <asp:Button ID="Button1" Runat="server" Text="Button" />
    </form>
</body>
</html>
```

This simple page shows some default server controls that appear just as you would expect, but that you can change with one of the ASP.NET built-in themes. When the page is called in the browser, it should look like Figure 7-1.


INETA - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:43018/Themes/Default.aspx
Go Links
Button
Done
Local intranet

International .NET Association (INETA)

<			February 2004			>
Sun	Mon	Tue	Wed	Thu	Fri	Sat
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	1	2	3	4	5	6

Figure 7-1

To instantly change the appearance of this page without changing the style of each server control on the page, you simply apply one of the ASP.NET default themes from within the `Page` directive:

```
<%@ Page Language="VB" Theme="SmokeAndGlass" %>
```

Adding the `Theme` attribute to the `Page` directive changes the appearance of everything on the page that is defined in the `SmokeAndGlass` theme file provided with ASP.NET 2.0. When you invoke the page in the browser, you see the result shown in Figure 7-2.

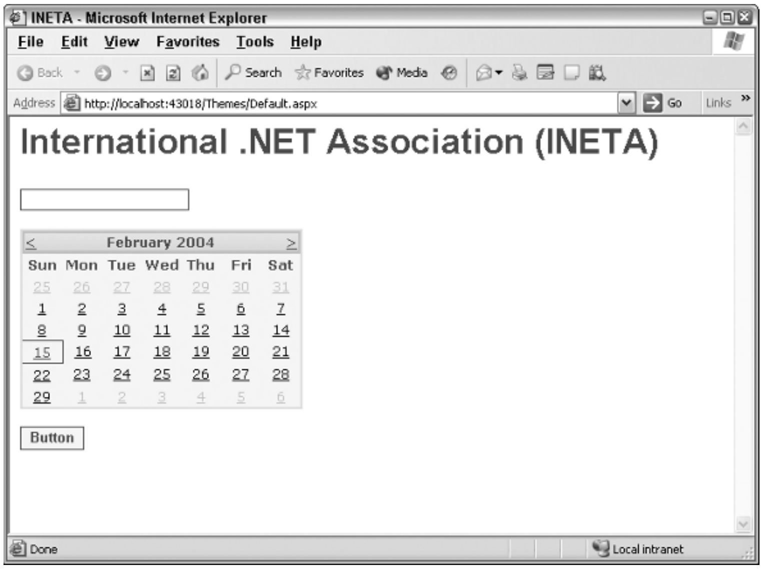

INETA - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:43018/Themes/Default.aspx
Go Links
International .NET Association (INETA)
February 2004
Sun Mon Tue Wed Thu Fri Sat
Button
Done
Local intranet

Figure 7-2

Applying a theme to an entire application

In addition to applying an ASP.NET 2.0 predefined theme to your ASP.NET pages using the `Theme` attribute within the `Page` directive, you can also apply it at an application level from the `web.config` file. This is illustrated in Listing 7-2.

Listing 7-2: Applying a theme application-wide from the web.config file

```
<?xml version="1.0" encoding="UTF-8" ?>

<configuration>
   <system.web>
      <pages theme="SmokeAndGlass" />
   </system.web>
</configuration>
```

If you specify the theme in the `web.config` file, you don't need to define the theme again in the `Page` directive of your ASP.NET pages. This theme is applied automatically to each and every page within your application.

Applying a theme to all applications on a server

If you want to take it even one level higher, you can specify the theme that you want to use within the `machine.config` file. This is illustrated in Listing 7-3.

Listing 7-3: Specifying the theme in the machine.config file

```
<pages buffer="true" enableSessionState="true" enableViewState="true"
   enableViewStateMac="true" autoEventWireup="true" validateRequest="true"
   enablePersonalization="false" theme="SmokeAndGlass" >...</pages>
```

The `machine.config` file is located at `C:\WINDOWS\Microsoft.NET\Framework\v2.0.xxxxx\CONFIG`. The `pages` node is about one-third of the way through the file. Adding the `Theme` attribute to the `pages` node within the `machine.config` file causes every Web application on that server to use the specified theme. This is a great solution if the server has multiple applications that should all be using the same theme.

If you set a theme in the `machine.config` file, you are not in any way required to use this theme for all the applications on the server. To override the theme setting placed in the `machine.config` file, you just specify another theme in the application's `web.config` file or in the Web page's `Page` directive. Remember settings that are set in the `web.config` file override settings that are in the `machine.config` file. Settings that are placed in the `Page` directive override both settings in the `machine.config` and in the `web.config` files.

Removing themes from server controls

Whether themes are set on a server, at the application level, or on a page, at times you want an alternative to the theme that has been defined. For example, change the text box server control that you have been working with (from Listing 7-1) by making its background black and using white text:

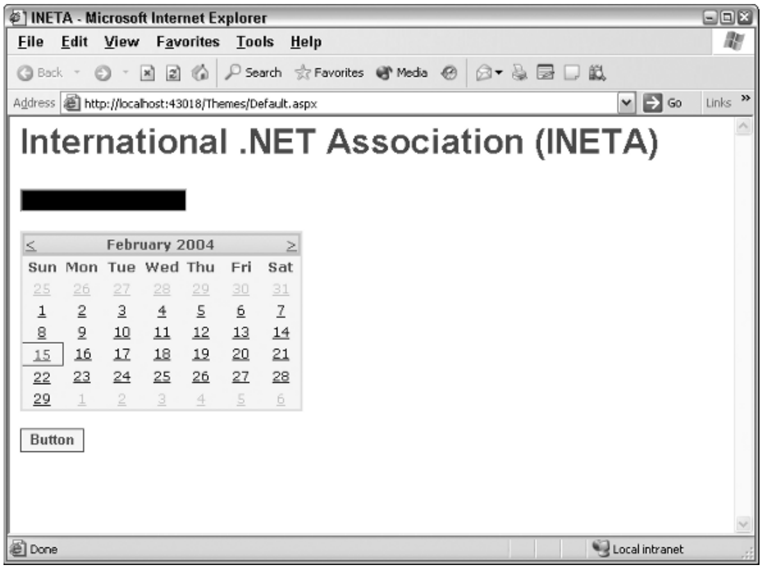
```
<asp:Textbox ID="TextBox1" Runat="server"
 BackColor="#000000" ForeColor="#ffffff" />
```

The black background color and the color of the text in the text box are specified directly in the control itself with the use of the `BackColor` and `ForeColor` attributes. If you have applied a theme to the page where this text box control is located, however, you won't see this black background or white text because these changes are overridden by the theme itself.

To apply a theme to your ASP.NET page but not to this text box control, you simply use the `EnableTheming` property of the text box server control:

```
<asp:Textbox ID="TextBox1" Runat="server"
 BackColor="#000000" ForeColor="#ffffff" EnableTheming="false" />
```

If you apply this control to the text box server control from Listing 7-1 with the `SmokeAndGlass` theme applied to the entire page, the theme is applied to every control on the page *except* the text box. This result is shown in Figure 7-3.


INETA - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:43018/Themes/Default.aspx
Go Links
Button
Done
Local intranet

International .NET Association (INETA)

<			February 2004			>
Sun	Mon	Tue	Wed	Thu	Fri	Sat
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	1	2	3	4	5	6

Figure 7-3

If you want to turn off theming for multiple controls within a page, consider using the Panel control to encapsulate a collection of controls and then set the `EnableTheming` attribute of the Panel control to `False`. This disables theming for each control contained within the Panel control.

Removing themes from Web pages

Now what if, when you set the theme for an entire application in the `web.config` file, you want to exclude a single ASP.NET page? It is quite possible to remove a theme setting at the page level, just as it is at the server control level.

The `Page` directive includes an `EnableTheming` attribute that can be used to remove theming from your ASP.NET pages. To remove the theme that would be applied by the theme setting in the `web.config` or `machine.config` file, you simply construct your `Page` directive in the following manner:

```
<%@ Page Language="VB" EnableTheming="False" %>
```

This construct sets the theme to nothing — thereby removing any settings that were specified in the `web.config` or `machine.config` files.

Removing themes from applications

Because themes can be set in the `machine.config` file that affect every application on the server, you might sometimes want to remove the theme setting from the application that you are working on. To do this, you specify no theme in the `web.config` file. This construct is shown in Listing 7-4.

Listing 7-4: Removing the server-set theme in the web.config file

```
<?xml version="1.0" encoding="UTF-8" ?>

<configuration>
   <system.web>
      <pages theme="" />
   </system.web>
</configuration>
```

Creating Your Own Themes

When you are applying themes to your applications, you are in no way limited just to default themes provided with ASP.NET. You can easily create your own themes. The themes that you create can be applied at the application level or put in the server theme repository along with the Microsoft default themes that come with the ASP.NET 2.0 install. As you can see, themes are a great way to easily apply a consistent look and feel across your entire application.

Creating the proper folder structure

In order to create your own themes for an application, you first need to create the proper folder structure in your application. To do this, right-click your project and add a new folder. Name the folder `Themes`. Notice when you do this that the `Themes` folder does not have the typical folder icon next to it, but instead has a folder icon that includes a paint brush. This is shown in Figure 7-4.

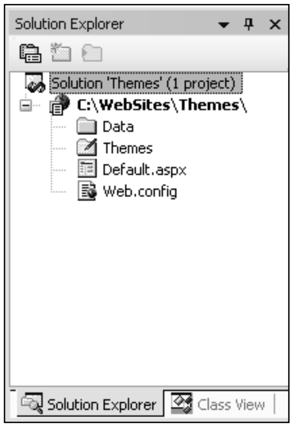


Figure 7-4

Within the `Themes` folder itself, you create an additional theme folder for each and every theme that you might use in your application. For instance, if you are going to have four themes — Summer, Fall, Winter, and Spring — then you create four folders that are named appropriately.

You might use more than one theme in your application for many reasons — season changes, day/night changes, category of user, or even user preferences.

Each theme folder must contain the elements of the theme, which can include

- ❑ A single skin file
- ❑ CSS files
- ❑ Images

Creating a skin

A *skin* is a definition of styles applied to server controls in your ASP.NET page. Skins can work in conjunction with CSS files or images. To create a theme to use in your ASP.NET applications, you use just a single skin file in the theme folder. The skin file can have any name, but it must have a `.skin` file extension.

Even though you have four theme folders in your application, concentrate on the creation of the Summer theme for the purposes of this chapter. Within the `Summer` folder in your project, create a text file called `Summer.skin`. If you try to right-click the `Summer` theme folder and select Add New Item, notice that a skin file isn't listed among the options. Therefore, select the Text File option and name the file `Summer.skin`. Then create a skin file as shown in Listing 7-5.

Listing 7-5: The Summer.skin file

```
<asp:Label Runat="server" ForeColor="#004000" Font-Names="Verdana"
           Font-Size="X-Small" />

<asp:Textbox Runat="server" ForeColor="#004000" Font-Names="Verdana"
           Font-Size="X-Small" BorderStyle="Solid" BorderWidth="1px"
           BorderColor="#004000" Font-Bold="True" />

<asp:Button Runat="server" ForeColor="#004000" Font-Names="Verdana"
           Font-Size="X-Small" BorderStyle="Solid" BorderWidth="1px"
           BorderColor="#004000" Font-Bold="True" BackColor="#FFE0C0" />
```

This is just a sampling of what the `Summer.skin` file should be. If you are going to use it in a real application, you actually make a definition for each and every server control option. In this case, you have a definition in place for three different types of server controls — the Label, TextBox, and Button controls. After saving the `Summer.skin` file in the `Summer` folder, your file structure should resemble Figure 7-5 from the Solution Explorer of Visual Studio 2005.

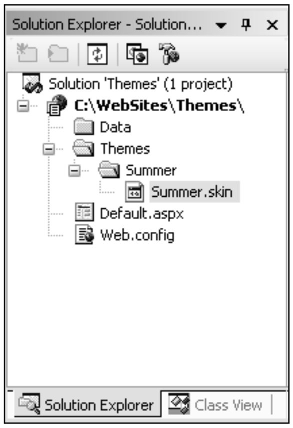


Figure 7-5

Just like the regular server control definitions that you put on a typical `.aspx` page, these control definitions must contain the `Runat="server"` attribute. If you specify this attribute in the skinned version of the control, you also include it in the server control you put on an `.aspx` page that uses this theme. Also notice is that no `ID` attribute is specified in the skinned version of the control. If you specify an `ID` attribute here, you get an error when a page tries to use this theme.

As you can see, you supply a lot of different visual definitions to these three controls and this should give the page a summery look and feel. An ASP.NET page in this project can simply use this custom theme as it would any global Microsoft pre-installed theme (see Listing 7-6).

Listing 7-6: Using the Summer theme in an ASP.NET page

VB

```
<%@ Page Language="VB" Theme="Summer" %>

<script runat="server">
    Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = "Hello " & Textbox1.Text
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>INETA</title>
</head>
<body>
    <form id="form1" runat="server">
        <asp:Textbox ID="TextBox1" Runat="server">
        </asp:Textbox>
        <br />
        <br />
        <asp:Button ID="Button1" Runat="server" Text="Submit Your Name"
         OnClick="Button1_Click" />
        <br />
```

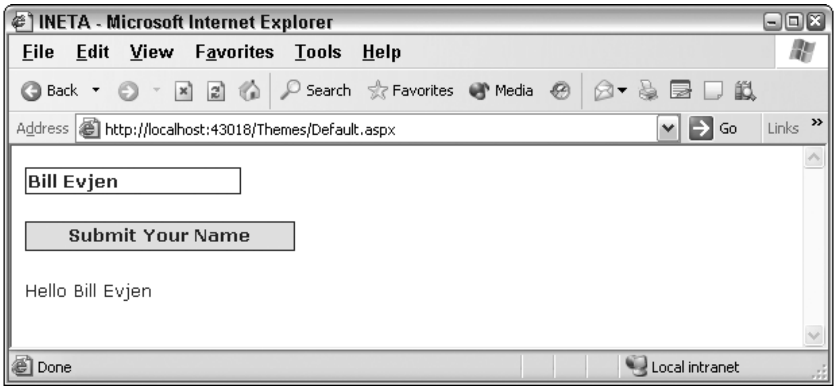
```
            <br />
            <asp:Label ID="Label1" Runat="server" />
        </form>
    </body>
    </html>
```

C#

```
<%@ Page Language="C#" Theme="Summer" %>

<script runat="server">
    void Button1_Click(object sender, System.EventArgs e)
    {
        Label1.Text = "Hello " + TextBox1.Text.ToString();
    }
</script>
```

As you can see from the server controls on this .aspx page, no styles are associated with them. These are just the default server controls that you drag and drop onto the design surface of Visual Studio 2005. There is, however, the style that you defined in the `Summer.skin` file, as shown in Figure 7-6.


INETA - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Media
Address
http://localhost:43018/Themes/Default.aspx
Go
Links
Bill Evjen
Submit Your Name
Done
Local intranet

Hello Bill Evjen

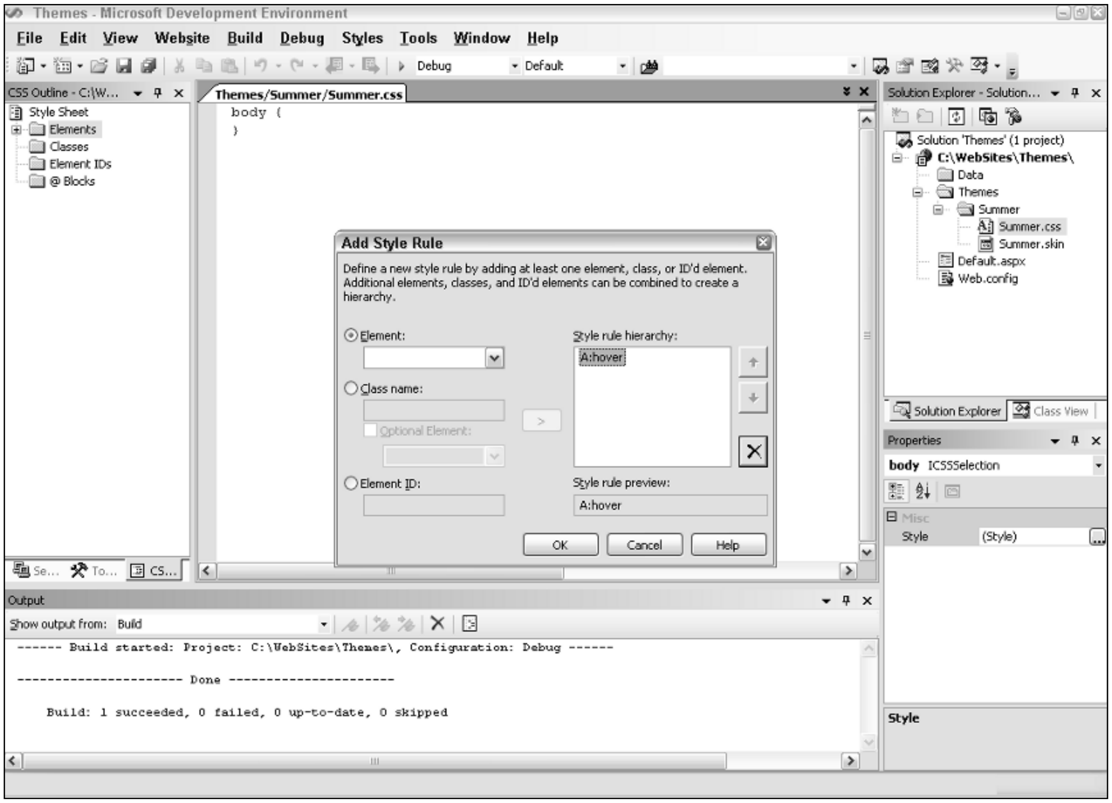
Figure 7-6

Including CSS files in your themes

In addition to the server control definitions that you create from within a .skin file, you can make further definitions using Cascading Style Sheets (CSS). You might have noticed, when using a .skin file, that you could define only the styles associated with server controls and nothing else. But developers usually use quite a bit more than server controls in their ASP.NET pages. For instance, ASP.NET pages are routinely made up of HTML server controls, raw HTML, or even raw text. As the Summer theme stands at present it has only a `Summer.skin` file associated with it. Any of these other items would have no style whatsoever applied to them.

For a theme that goes beyond the server controls, you must further define the theme style so that HTML server controls, HTML, and raw text are all changed according to the theme. You achieve this with a CSS file within your `Themes` folder.

It is rather easy to create CSS files for your themes when using Visual Studio 2005. Right-click the `Summer` theme folder and select Add New Item. In the list of options, select the option Style Sheet and name it `Summer.css`. The `Summer.css` file should be sitting right next to your `Summer.skin` file. This creates an empty `.css` file for your theme. I won't go into the details of how to make a CSS file using Visual Studio 2005 and the CSS creation tool. The process is the same as in previous versions. I just want to point out that it is quite simple to do this because the dialog that comes with Visual Studio 2005 enables you to completely define your CSS page with no need to code. A sample dialog is shown in Figure 7-7.



```
body {
}
```

Define a new style rule by adding at least one element, class, or ID'd element. Additional elements, classes, and ID'd elements can be combined to create a hierarchy.

```
------ Build started: Project: C:\WebSites\Themes\, Configuration: Debug ------

--------------------- Done ---------------------

    Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped
```

Figure 7-7

To create a comprehensive theme with this dialog, you define each HTML element that might appear in the ASP.NET page. This can be a lot of work, but it's worth it in the end. For now, create a small CSS file that changes some of the nonserver control items on your ASP.NET page. This CSS file is shown in Listing 7-7.

Listing 7-7: A CSS file with some definitions

```
body
{
 font-size: x-small;
```

```
    font-family: Verdana;
    color: #004000;
   }

   A:link {
    color: Blue;
    text-decoration:none;
   }

   A:visited
   {
    color: Blue;
    text-decoration:none;
   }

   A:hover {
    COLOR: Red;
    text-decoration:underline overline;
   }
```

In this CSS file, you define four things. First, you define text that is found within the `<body>` tag of the page (basically all the text). Plenty of text appears in a typical ASP.NET page that is not placed inside of an `<asp:label>` or `<asp:literal>` tag. Therefore, you define how your text should appear — otherwise, your Web page appears quite odd at times. In this case, a definition is in place for the size, the font family, and the color of the text. You make this definition the same as the one for the `<asp:label>` server control in the `Summer.skin` file.

The next three definitions in this CSS file revolve around the `<a>` element (for hyperlinks). One cool feature that many Web pages use is responsive hyperlinks — or hyperlinks that change when you hover a mouse over them. The `A:link` definition defines what a typical link looks like on the page. The `A:visited` definition defines the look of the link if the end user has clicked on the link previously (without this definition, it is typically purple in IE). Then the `A:hover` definition defines the appearance of the hyperlink when the end user hovers the mouse over the link. You can see that not only are these three definitions changing the color of the hyperlink, but they are also changing how the underline is used. In fact, when the end user hovers the mouse over a hyperlink on a page using this CSS file, an underline and an overline appear on the link itself.

In CSS files, the order in which the style definitions appear in the `.css` file is important. This is an interpreted file — the first definition that appears in the CSS file is applied first to the page, then the second is applied, and so forth. Some styles might change previous styles, so make sure your style definitions are in the proper order. For instance, if you put the `A:hover` style definition first, you would never see it. The `A:link` and `A:visited` definitions would supersede it because they are defined after the fact.

In working with your themes that include `.css` files, you must understand what they can and cannot do for you. For instance, examine an `.aspx` file that contains two text boxes — one text box created using a server control and another text box created using a typical `<input>` HTML element:

```
   <asp:Textbox ID="TextBox1" Runat="server" />&nbsp;
   <input type="text" />
```

Suppose that there is a definition for the TextBox server control in the `.skin` file:

```
<asp:Textbox Runat="server" ForeColor="#004000" Font-Names="Verdana"
 BackColor="#ffffff" Font-Size="X-Small" BorderStyle="Solid" BorderWidth="1px"
 BorderColor="#004000" Font-Bold="True" />
```

But, what if you also had a definition in your `.css` file for each `<input>` element in the ASP.NET page as shown here:

```
INPUT
{
 background-color: black;
}
```

When you run the `.aspx` page with these kinds of style conflicts, the `.skin` file takes precedence over styles applied to every HTML element that is created using ASP.NET server controls regardless of what the `.css` file says. In fact, this sort of scenario gives you a page in which the `<input>` element that is created from the server control is white as defined in the `.skin` file and the second text box is black as defined in the `.css` file. This is shown in Figure 7-8.

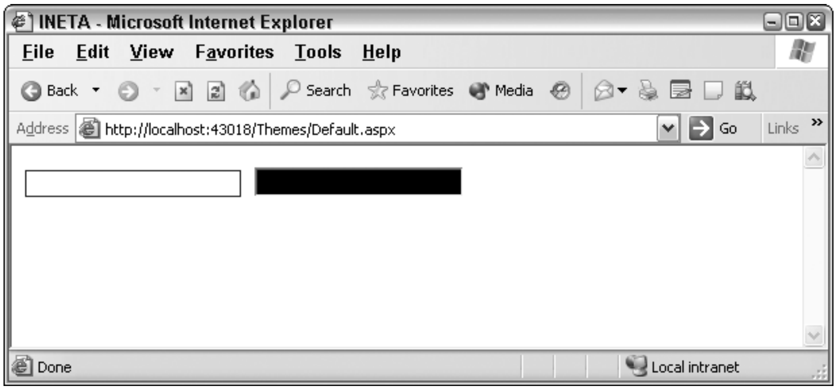

INETA - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:43018/Themes/Default.aspx
Go Links
Done
Local intranet

Figure 7-8

Having your themes include images

Probably one of the coolest reasons why themes, rather than CSS, are the better approach for applying a consistent style to your Web page is that themes enable you to incorporate actual images into the style definitions.

A lot of controls use images to create a better visual appearance. The first step in incorporating images into your server controls that consistently use themes is to create an `Images` folder within the `Themes` folder itself, as illustrated in Figure 7-9.

You have a couple of easy ways to use the images that you might place in this folder. The first is to incorporate the images directly from the `.skin` file itself. You can do this with the TreeView server control. The TreeView control can contain images used to open and close nodes for navigation purposes. You can place images in your theme for each and every TreeView control in your application. If you do that, you can then define the TreeView server control in the `.skin` file, as shown in Listing 7-8.

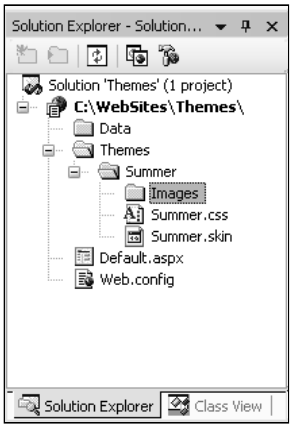


Figure 7-9

Listing 7-8: Using images from the theme folder in a TreeView server control

```
<asp:TreeView runat="server" BorderColor="#FFFFFF" BackColor="#FFFFFF"
   ForeColor="#585880" Font-Size=".9em" Font-Names="Verdana"
   LeafNodeImageURL="images\summer_iconlevel.gif"
   RootNodeImageURL="images\summer_iconmain.gif"
   ParentNodeImageURL="images\summer_iconmain.gif" NodeIndent="30"
   CollapseImageURL="images\summer_minus.gif"
   ExpandImageURL="images\summer_plus.gif">
  ...
</asp:TreeView>
```

When you run a page containing a TreeView server control, it is populated with the images held in the `Images` folder of the theme.

It is easy to incorporate images into the TreeView control. It even specifically asks for an image location as an attribute of the control. The new WebParts controls are used to build portals. Listing 7-9 is an example of a Web Part definition from a `.skin` file that incorporates images from the `Images` folder of the theme.

Listing 7-9: Using images from the theme folder in a WebPartZone server control

```
<asp:WebPartZone runat="server"  PartFrameType="TitleAndBorder"
 DragHighlightColor="#6464FE"  ShowIconInPartTitle="True" BorderStyle="double"
 BorderColor="#E7E5DB" BorderWidth="2pt" BackColor="#F8F8FC"
 cssclass="theme_fadeblue" Font-Size=".9em" Font-Names="Verdana">
    <PartContentStyle     ForeColor="#585880" BorderStyle="double"
     BorderColor="#585880" BorderWidth="1pt"
     BackColor="#FFFFFF"></PartContentStyle>
    <FooterStyle          ForeColor="#585880" BackColor="#CCCCCC"></FooterStyle>
    <WebPartHelpVerb      ImageURL="images/SmokeAndGlass_help.gif"
     checked="False" enabled="True"   visible="True"></WebPartHelpVerb>
    <WebPartCloseVerb     ImageURL="images/SmokeAndGlass_close.gif"
     checked="False" enabled="True"   visible="True"></WebPartCloseVerb>
    <WebPartRestoreVerb   ImageURL="images/SmokeAndGlass_restore.gif"
```

```
        checked="False" enabled="True"   visible="True"></WebPartRestoreVerb>
       <WebPartMinimizeVerb  ImageURL="images/SmokeAndGlass_minimize.gif"
        checked="False" enabled="True"   visible="True"></WebPartMinimizeVerb>
       <WebPartEditVerb      ImageURL="images/SmokeAndGlass_edit.gif"
        checked="False" enabled="True"  visible="True"></WebPartEditVerb>
       <TitleStyle           ForeColor="#FFFFFF" Font-Names="Verdana"
        BorderStyle="double" BorderWidth="0" Font-Bold="true"  BorderColor="#E7E5DB"
        BackColor="#232377"></TitleStyle>
       <PartStyle            ForeColor="#585880" Font-Names="Verdana" Font-Size=".9em"
        BorderColor="#44448A" BackColor="#F8F7F4"></PartStyle>
       <PartTitleStyle       ForeColor="#585880" Font-Names="Verdana"
        BorderStyle="solid" Font-Bold="true" BorderWidth="1pt" Font-Size=".9em"
        BorderColor="#494979" BackColor="#F8F7F4"
        cssclass="theme_header"></PartTitleStyle>
       <PartVerbStyle        ForeColor="#FFFFFF" Font-Names="Verdana" Font-
        Underline="False" Font-Size=".7em" BorderColor="#000066" BorderWidth="1pt"
        BackColor="#8383B6"></PartVerbStyle>
       <EditWebPartStyle     ForeColor="#6464FE" BorderColor="#6464FE"
        BackColor="#6464FE" Font-Size=".9em" Font-Names="Verdana"/>
   </asp:WebPartZone>
```

As you can see here, this series of toolbar buttons that are in a WebPart now use images that come from the SmokeAndGlass theme. When this WebPart is generated, the style is defined directly from the .skin file, but the images specified in the .skin file are retrieved from the Images folder in the theme itself.

Not all server controls enable you to work with images directly from the Themes folder by giving you an image attribute to work with. If you don't have this capability, you must work with the .skin file and the CSS file together. If you do, you can place your theme-based images in any element you want. The SmokeAndGlass theme that comes with ASP.NET 2.0 is a good example of how to do this.

Place the image that you want to use in the Images folder just as you normally would. Then define the use of the images in the .css file. The SmokeAndGlass example in Listing 7-10 demonstrates this.

Listing 7-10: Part of the CSS file from SmokeAndGlass.css

```
   .theme_header {
    background-image :url( images/smokeandglass_brownfadetop.gif);
   }

   .theme_highlighted {
    background-image :url( images/smokeandglass_blueandwhitef.gif);
   }

   .theme_fadeblue {
    background-image :url( images/smokeandglass_fadeblue.gif);
   }
```

These are not styles for a specific HTML element; instead, they are CSS classes that you can put into any HTML element that you want. In this case, each CSS class mentioned here is defining a specific background image to use for the element.

After it is defined in the CSS file, you can utilize this CSS class in the `.skin` file when defining your server controls. Listing 7-11 shows you how.

Listing 7-11: Using the CSS class in one of the server controls defined in the .skin file

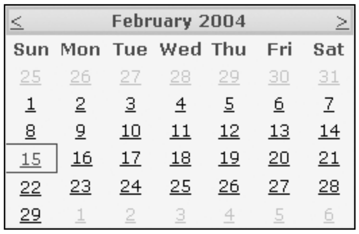
```
<asp:Calendar runat="server" BorderStyle="double" BorderColor="#E7E5DB"
 BorderWidth="2" BackColor="#F8F7F4" Font-Size=".9em" Font-Names="Verdana">
    <TodayDayStyle      BackColor="#F8F7F4" BorderWidth="1" BorderColor="#585880"
     ForeColor="#585880" />
    <OtherMonthDayStyle BackColor="transparent" ForeColor="#CCCCCC" />
    <SelectedDayStyle   ForeColor="#6464FE" BackColor="transparent"
     cssclass="theme_highlighted" />
    <TitleStyle         Font-Bold="True"  BackColor="#CCCCCC" ForeColor="#585880"
     BorderColor="#CCCCCC" BorderWidth="1pt" cssclass="theme_header" />
    <NextPrevStyle      Font-Bold="True"  ForeColor="#585880"
     BorderColor="transparent"  BackColor="transparent" />
    <DayStyle                             ForeColor="#000000"
     BorderColor="transparent" BackColor="transparent" />
    <SelectorStyle      Font-Bold="True"  ForeColor="#696969" BackColor="#F8F7F4"
     />
    <WeekendDayStyle    Font-Bold="False" ForeColor="#000000"
     BackColor="transparent" />
    <DayHeaderStyle     Font-Bold="True"  ForeColor="#585880"
     BackColor="Transparent" />
</asp:Calendar>
```

This Calendar server control definition from a `.skin` file uses one of the earlier CSS classes in its definition. It actually uses an image that is specified in the CSS file in two different spots within the control (shown in bold). It is first specified in the `<SelectedDayStyle>` element. Here you see the attribute and value `cssclass="theme_highlighted"`. The other spot is within the `<TitleStyle>` element. In this case, it is using `theme_header`. When the control is rendered, these CSS classes are referenced and finally point to the images that are defined in the CSS file.

It is interesting that the images used here for the header of the Calendar control don't really have much to them. For instance, the `smokeandglass_brownfadetop.gif` image is simply a thin, gray sliver, as shown in Figure 7-10.

Figure 7-10

This very small image (in this case, very thin) is actually repeated as often as necessary to make it equal the length of the header in the Calendar control. The image is lighter at the top and darkens toward the bottom. Repeated horizontally, any control like this gives a three-dimensional effect to the control. Try it out, and you get the result shown in Figure 7-11.



Sun	Mon	Tue	Wed	Thu	Fri	Sat
25	26	27	28	29	30	31
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	1	2	3	4	5	6

Figure 7-11

Defining Multiple Skin Options

Using the themes technology in ASP.NET 2.0, you can have a single theme; but also, within the theme's `.skin` file, you can have specific controls that are defined in multiple ways. You can frequently take advantage of this feature within your themes. For instance, you might have text box elements scattered throughout your application, but you might not want each and every text box to have the same visual appearance. In this case, you can create multiple versions of the `<asp:textbox>` server control within your `.skin` file. In Listing 7-12 you see how to create multiple versions of the `<asp:textbox>` control in the `.skin` file from Listing 7-5.

Listing 7-12: The Summer.skin file, which contains multiple version of the <asp:textbox> server control

```
<asp:Label Runat="server" ForeColor="#004000" Font-Names="Verdana"
            Font-Size="X-Small" />

<asp:Textbox Runat="server" ForeColor="#004000" Font-Names="Verdana"
            Font-Size="X-Small" BorderStyle="Solid" BorderWidth="1px"
            BorderColor="#004000" Font-Bold="True" />

<asp:Textbox Runat="server" ForeColor="#000000" Font-Names="Verdana"
            Font-Size="X-Small" BorderStyle="Dotted" BorderWidth="5px"
            BorderColor="#000000" Font-Bold="False" SkinID="TextboxDotted" />

<asp:Textbox Runat="server" ForeColor="#000000" Font-Names="Arial"
            Font-Size="X-Large" BorderStyle="Dashed" BorderWidth="3px"
            BorderColor="#000000" Font-Bold="False" SkinID="TextboxDashed" />

<asp:Button Runat="server" ForeColor="#004000" Font-Names="Verdana"
            Font-Size="X-Small" BorderStyle="Solid" BorderWidth="1px"
            BorderColor="#004000" Font-Bold="True" BackColor="#FFE0C0" />
```

In this `.skin` file, you can see three definitions in place for the TextBox server control. The first one is the same as before. Although the second and third definitions have a different style, they also contain a new attribute in the definition — `SkinID`. To create multiple definitions of a single element, you use the `SkinID` attribute to differentiate among the definitions. The value used in the `SkinID` can be anything you want. In this case, it is `TextboxDotted` and `TextboxDashed`.

Note that no `SkinID` attribute is used for the first `<asp:Textbox>` definition. By not using one, you are saying that for each `<asp:Textbox>` control on an ASP.NET page that uses this theme but has no pointer to a `SkinID`, this is the default style definition to use.

Take a look at a sample `.aspx` page that uses this `.skin` file, Listing 7-13.

Listing 7-13: A simple .aspx page that uses the Summer.skin file with multiple text-box style definitions

```
<%@ Page Language="VB" Theme="Summer" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Different SkinIDs</title>
</head>
<body>
    <form id="form1" runat="server">
    <p>
        <asp:Textbox ID="TextBox1" Runat="server">Textbox1</asp:Textbox>
    </p><p>
        <asp:Textbox ID="TextBox2" Runat="server"
         SkinId="TextboxDotted">Textbox2</asp:Textbox>
    </p><p>
        <asp:Textbox ID="TextBox3" Runat="server"
         SkinId="TextboxDashed">Textbox3</asp:Textbox>
    </p>
    </form>
</body>
</html>
```

This small `.aspx` page shows three text boxes, each of a different style. When you run this page, you get the results shown in Figure 7-12.

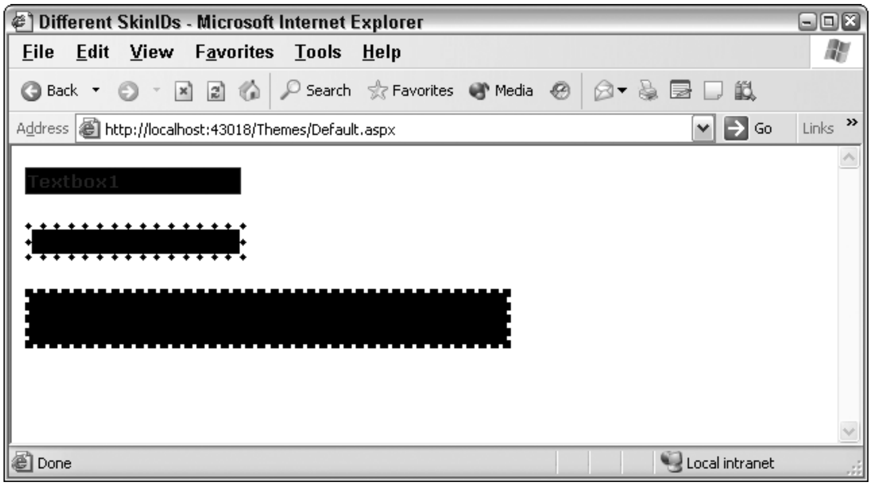

Different SkinIDs - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:43018/Themes/Default.aspx
Go Links
Textbox1
Done
Local intranet

Figure 7-12

The first text box doesn't point to any particular `SkinID` in the `.skin` file. Therefore, the default skin is used. As stated before, the default skin is the one in the `.skin` file that doesn't have a `SkinID` attribute in it. The second text box then contains `skinid="TextboxDotted"` and, therefore, inherits the style definition defined in the `TextboxDotted` skin in the `Summer.skin` file. The third text box takes the `SkinID` `TextboxDashed` and is also changed appropriately.

As you can see, it is quite simple to define multiple versions of a control that can be used throughout your entire application.

Programmatically Working with Themes

So far, you have seen examples of working with ASP.NET 2.0 themes in a declarative fashion, but you can also work with themes programmatically.

Assigning the page's theme programmatically

To programmatically assign the theme to the page, use the construct shown in Listing 7-14.

Listing 7-14: Assigning the theme of the page programmatically

VB

```
<script runat="server" language="vb">
    Sub Page_PreInit(ByVal sender As Object, ByVal e As System.EventArgs)
       Page.Theme = Request.QueryString("ThemeChange")
    End Sub
</script>
```

C#

```
<script runat="server">
    void Page_PreInit(object sender, System.EventArgs e)
    {
       Page.Theme = Request.QueryString["ThemeChange"];
    }
</script>
```

You must set the `Theme` of the `Page` property in or before the `Page_PreInit` event for any static controls that are on the page. If you are working with dynamic controls, set the `Theme` property before adding it to the Controls collection.

Assigning a control's SkinID programmatically

Another option is to assign a specific server control's `SkinID` property programmatically (see Listing 7-15).

Listing 7-15: Assigning the server control's SkinID property programmatically

VB

```
<script runat="server" language="vb">
    Sub Page_PreInit(ByVal sender As Object, ByVal e As System.EventArgs)
```

```
        TextBox1.SkinID = "TextboxDashed"
    End Sub
</script>
```

C#

```
<script runat="server">
    void Page_PreInit(object sender, System.EventArgs e)
    {
        TextBox1.SkinID = "TextboxDashed";
    }
</script>
```

Again, you assign this property before or in the `Page_PreInit` event in your code.

Themes and Custom Controls

If you are building custom controls in an ASP.NET 2.0 world, understand that end users can also apply themes to the controls that they use in their pages. By default, your custom controls are theme enabled whether your custom control inherits from `Control` or `WebControl`.

To disable theming for your control, you can simply use the `EnableTheming` attribute on your class. This is illustrated in Listing 7-16.

Listing 7-16: Disabling theming for your custom controls

VB

```
Namespace Wrox.ServerControls

    <EnableTheming(False)> _
    Public Class SimpleHello
        Inherits System.Web.UI.Control

        Private _name As String

        Public Property Name() As String
            Get
                Return _name
            End Get
            Set(ByVal Value As String)
                _name = Value
            End Set
        End Property

        Protected Overrides Sub RenderContents(ByVal controlOutput As _
           HtmlTextWriter)
              controlOutput.Write("Hello " + Name)
        End Sub

    End Class

End Namespace
```

(continued)

Listing 7-16: *(continued)*

C#

```
namespace Wrox.ServerControls
{
    [EnableTheming(false)]
    public class SimpleHello : Control
    {
        private string _name;

        public string Name
        {
            get { return _name; }
            set { _name = value; }
        }

        protected override void RenderContents (HtmlTextWriter controlOutput)
        {
            controlOutput.Write ("Hello " + Name);
        }
    }
}
```

You can also disable theming for the individual properties that might be in your custom controls. This is done as illustrated in Listing 7-17.

Listing 7-17: Disabling theming for properties in your custom controls

VB

```
Namespace Wrox.ServerControls

    Public Class SimpleHello
        Inherits System.Web.UI.Control

        Private _myValue As String

        <Themeable(False)>
        Public Property MyCustomProperty() As String
            Get
                Return _myValue
            End Get
            Set(ByVal Value As String)
                _myValue = Value
            End Set
        End Property

    End Class

End Namespace
```

C#

```
namespace Wrox.ServerControls
{
    public class SimpleHello : Control
    {
        private string _myValue;

        [Themeable(false)]
        public string Name
        {
            get { return _myValue; }
            set { _myValue = value; }
        }
    }
}
```

Summary

With the addition of themes and skins in ASP.NET 2.0, it has become quite easy to apply a consistent look and feel across your entire application. Remember that themes can just contain simple server control definitions in a `.skin` file or elaborate style definitions, which include not only `.skin` files, but also CSS style definitions and even images!

As you will see later in the book, you can use themes in conjunction with the new personalization features that ASP.NET 2.0 provides. This can enable your end users to customize their experiences by selecting their own themes. Your application can present a theme just for them, and it can remember their choices through the APIs that are offered in ASP.NET 2.0.