

Unit 11

13.0 Styles, Themes and Master Pages (Text Book Chapter 12)

➤ 13.0 Styles, Themes and Master Pages

13.0 Styles, Themes and Master Pages

13.1 Introduction

Using the techniques you've learned so far, you can create polished web pages and let users surf from one page to another. However, to integrate your web pages into a unified, consistent website, you need a few more tools. In this chapter, you'll consider three of the most important tools that you can use: styles, themes, and master pages.

Styles are part of the Cascading Style Sheet (CSS) standard. They aren't directly tied to ASP.NET, but they're still a great help in applying consistent formatting across your entire website. With styles, you can define a set of formatting options once, and reuse it to format different elements on multiple pages. You can even create styles that apply their magic automatically—for example, styles that change the font of all the text in your website without requiring you to modify any of the web page code. Best of all, once you've standardized on a specific set of styles and applied them to multiple pages, you can give your entire website a face-lift just by editing your style sheet.

Styles are genuinely useful, but there are some things they just can't do. Because styles are based on the HTML standard, they have no understanding of ASP.NET concepts like control properties. To fill the gap, ASP.NET includes a themes feature, which plays a similar role to styles but works exclusively with server controls. Much as you use styles to automatically set the formatting characteristics of HTML elements, you use themes to automatically set the properties of ASP.NET controls.

Another feature for standardizing websites is master pages. Essentially, a master page is a blueprint for part of your website. Using a master page, you can define web page layout, complete with all the usual details such as headers, menu bars, and ad banners. Once you've perfected a master page, you can use it to create content pages. Each content page automatically acquires the layout and the content of the linked master page.

➤ 13.0 Styles, Themes and Master Pages

By using styles, themes, and master pages, you can ensure that all the pages on your website share a standardized look and layout. In many cases, these details are the difference between an average website and one that looks truly professional.

13.2 Styles

In the early days of the Internet, website designers used the formatting features of HTML to decorate these pages. These formatting features were limited, inconsistent, and sometimes poorly supported. Worst of all, HTML formatting led to horribly messy markup, with formatting details littered everywhere.

The solution is the CSS standard, which is supported in all modern browsers. Essentially, CSS gives you a wide range of consistent formatting properties that you can apply to any HTML element. Styles allow you to add borders, set font details, change colors, add margin space and padding, and so on. Many of the examples you've seen so far have in this book have used CSS formatting.

In the following sections, you'll learn the basics of the CSS standard. You'll see how web controls use CSS to apply their formatting, and you'll learn how you can explicitly use styles in your ASP.NET web pages.

13.2.1 Style Types

Web pages can use styles in three different ways:

Style Type	Description
Inline style	An inline style is a style that's placed directly inside an HTML tag. This can get messy, but it's a reasonable approach for one-time formatting. You can remove the style and put it in a style sheet later.
Internal style sheet	An internal style sheet is a collection of styles that are placed in the <head> section of your web page markup. You can then use the styles from this style sheet to format the web controls on that page. By using an internal style sheet, you get a clear separation between formatting (your styles) and your content (the rest of your HTML markup). You can also reuse the same style for multiple elements.
External style sheet	An external style sheet is similar to an internal style sheet, except it's placed in a completely separate file. This is the most powerful approach, because it gives you a way to apply the same style rules to many pages. You can use all types of styles with ASP.NET web pages. You'll see how in the following sections.

Figure13.2-1 Web Page Style Types

➤ 13.0 Styles, Themes and Master Pages

You can use all types of styles with ASP.NET web pages. You'll see how in the following sections.

To apply a style to an ordinary HTML element, you set the style attribute. Here's an example that gives a blue background to a paragraph:

```
<p style="background: Blue">This text has a blue background.</p>
```

Every style consists of a list of one or more formatting properties. In the preceding example, the style has a single formatting property, named background, which is set to the value Blue. To add multiple style properties, you simply separate them with semicolons, as shown here:

```
<p style="color:White; background:Blue; font-size:x-large; padding:10px">  
This text has a blue background.</p>
```

This style creates large white text with a blue background and 10 pixels of spacing between the edge of the element (the blue box) and the text content inside.

NOTE: The full list of formatting properties is beyond the scope of this book (although you can get all the details at www.w3schools.com/css). However, you'll soon see that Visual Studio includes tools that can help you build the styles you want, so you don't need to remember style property names or write styles by hand.

You can use the same approach to apply formatting to a web control using a style. However, you don't need to, because web controls provide formatting properties. For example, if you create a Label control like this:

```
<asp:Label ID="MyLabel" runat="server" ForeColor="White"  
    BackColor="Blue" Font-Size="X-Large">Formatted Text</asp:Label>
```

it's actually rendered into this HTML, which uses an inline style:

```
<span id="MyLabel" style="color:White; background-color:Blue;  
    font-size:X-Large"> Formatted Text</span>
```

Incidentally, if you specify a theme and set formatting properties that overlap with your style, the properties have the final say.

➤ 13.0 Styles, Themes and Master Pages

13.2.2 The Style Builder

Visual Studio provides an indispensable style builder that lets you create styles by picking and choosing your style preferences in a dedicated dialog box. To try it out, begin by creating a new page in Visual Studio. Then drop a few controls onto your page (for example, a label, text box, and button).

Every new page starts with an empty <div> element. This <div> is simply a content container—by default, it doesn't have any appearance. However, by applying style settings to the <div>, you can create a bordered content region, and you can change the font and colors of the content inside. In this example, you'll see how to use Visual Studio to build a style for the <div> element.

```
<div>
  <asp:Label ID="Label1" runat="server">Type something here:
</asp:Label>
  <asp:TextBox ID="TextBox1" runat="server">
</asp:TextBox>
  <br /><br />
  <asp:Button ID="Button1" runat="server" Text="Button">
</asp:Button>
</div>
```

Example 13.2-1 <div> with styles

NOTE: CSS supports a feature it calls inheritance. With inheritance, some formatting properties (such as the font family) are passed down from a parent element to other nested elements. In other words, if you set the font family for a <div> element, all the elements inside will inherit the same font (unless they explicitly specify otherwise). Other properties, like margin and padding settings, don't use inheritance. To learn more about this behavior and the specific properties that use inheritance, you can experiment on your own, consult a dedicated book such as Eric Meyer's *CSS: The Definitive Guide* (O'Reilly, 2006), or use the tutorials at www.w3schools.com/css.

Before formatting the page, make sure all your controls are nested inside the <div> element. Your markup should look something like this – Example 13.2-2

➤ 13.0 Styles, Themes and Master Pages

In the design window, click somewhere inside the `<div>` (but not on another control). You'll know you're in the right spot when a border appears around your controls, showing you the outline of the `<div>`, as shown in Figure 13.2-2

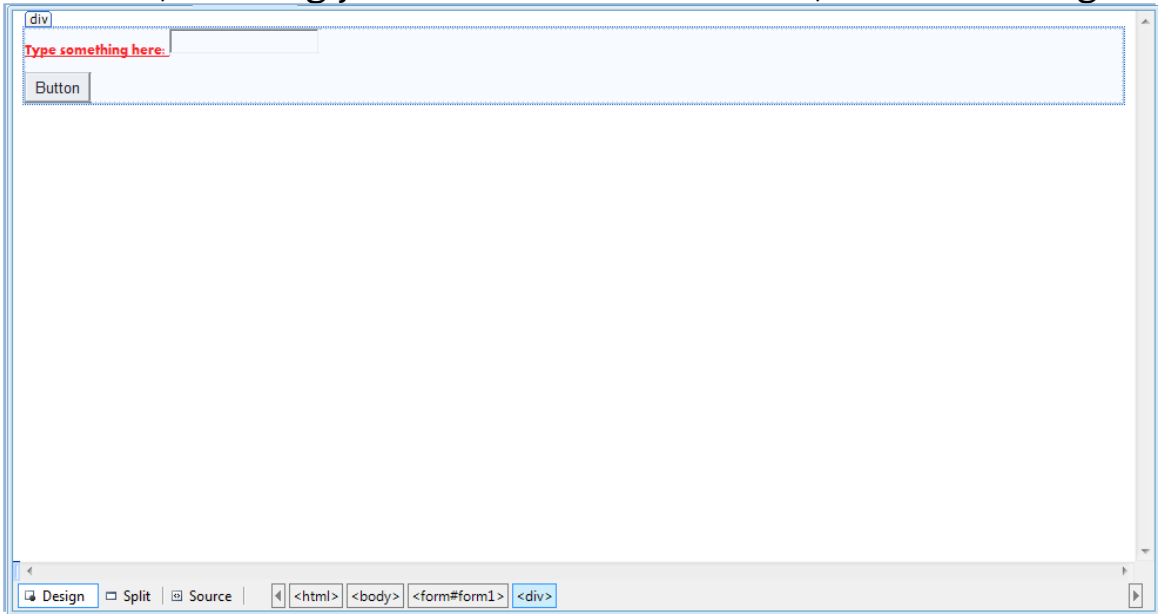


Figure13.2-2`<div>` with styles appearance

Next, choose Format New Style from the menu. This opens the New Style dialog box shown in Figure 13.2-2. In the Selector box at the top of the window, choose Inline Style to specify that you're creating your style directly in the HTML markup.

To specify style settings, you first need to choose one of the categories in the Category list. For example, if you choose Font you'll see a list of font-related formatting settings, such as font family, font size, text color, and so on. You can apply settings from as many different categories as you want. Table in figure 13.2-3 provides a brief explanation for each category.

➤ 13.0 Styles, Themes and Master Pages

Table Style Settings in the New Style Dialog Box	
Category	Description
Font	Allows you to choose the font family, font size, and text color, and apply other font characteristics (like italics and bold).
Block	Allows you to fine-tune additional text settings, such as the height of lines in a paragraph, the way text is aligned, the amount of indent in the first list, and the amount of spacing between letters and words.
Background	Allows you to set a background color or image.
Border	Allows you to define borders on one or more edges of the element. You can specify the border style, thickness, and color of each edge.
Box	Allows you to define the margin (the space between the edges of the element and its container) and the padding (the space between the edges of the element and its nested content inside).
Position	Allows you to set a fixed width and height for your element, and use absolute positioning to place your element at a specific position on the page. Use these settings with care. When you make your element a fixed size, there's a danger that the content inside can become too big (in which case it leaks out the bottom or the side). When you position your element using absolute coordinates, there's a chance that it can overlap another element.
Layout	Allows you to control a variety of miscellaneous layout settings. You can specify whether an element is visible or hidden, whether it floats at the side of the page, and what cursor appears when the user moves the mouse overtop, among other settings.
List	If you're configuring a list (a or element), you can set the numbering or bullet style. These settings aren't commonly used in ASP.NET web pages, because you're more likely to use ASP.NET list controls like the BulletedList.
Table	Allows you to set details that only apply to table elements (such as <tr> and <td>). For example, you can control whether borders appear around an empty cell.

Figure13.2-3 Table Style Settings in Style Dialog Box

Creating an Inline Style:

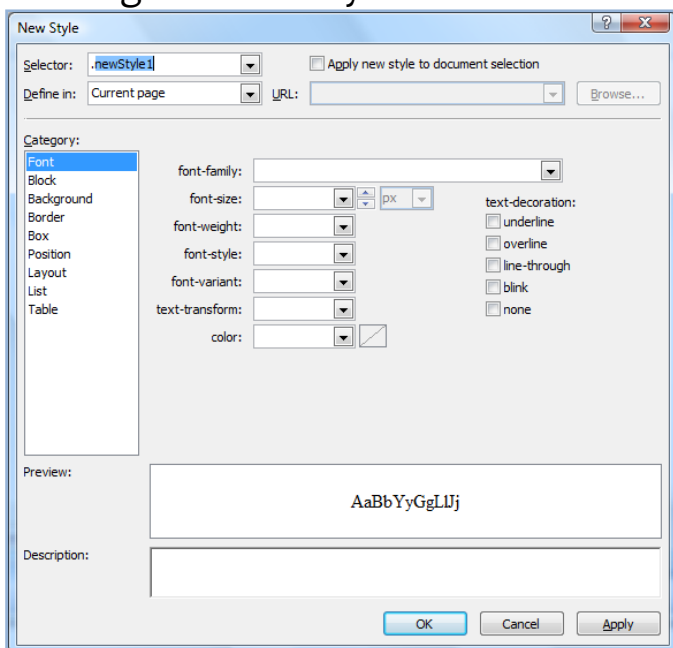


Figure13.2-4 Using The Style Dialog

NOTE : Remember, you can build a style for any HTML element, not just the <div> element. You'll always get exactly the same New Style dialog box with the same formatting options.

➤ 13.0 Styles, Themes and Master Pages

As you make your selections, Visual Studio shows what your style will look like when applied to some sample text (in the Preview box) and the markup that's needed to define your style (in the Description) box. Figure 13.2-5 shows the New Style dialog box after formatting the <div> into a nicely shaded and bordered box. In the Category list, all the categories with formatting settings are highlighted in bold.

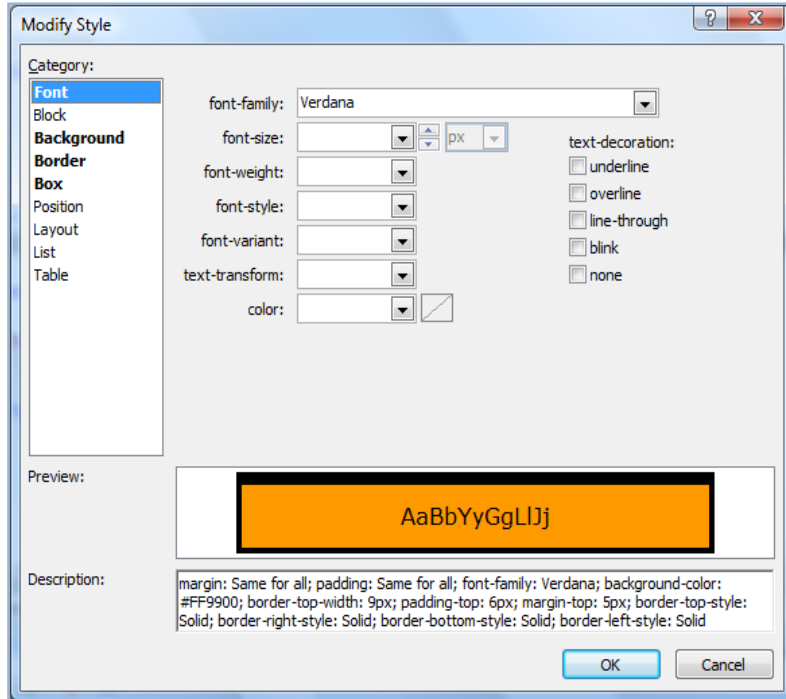


Figure13.2-5 New Style Dialog Box after Formatting

After setting the parameters, this is how the <div> tag will appear

```
<div style="margin: Same for all; padding: Same for all;
font-family: Verdana; background-color: #FF9900;
border-top-width: 9px; padding-top: 6px; margin-
top: 5px; border-top-style: Solid; border-right-style:
Solid; border-bottom-style: Solid; border-left-style:
Solid;">
```

Example 13.2-2 <div> with styles

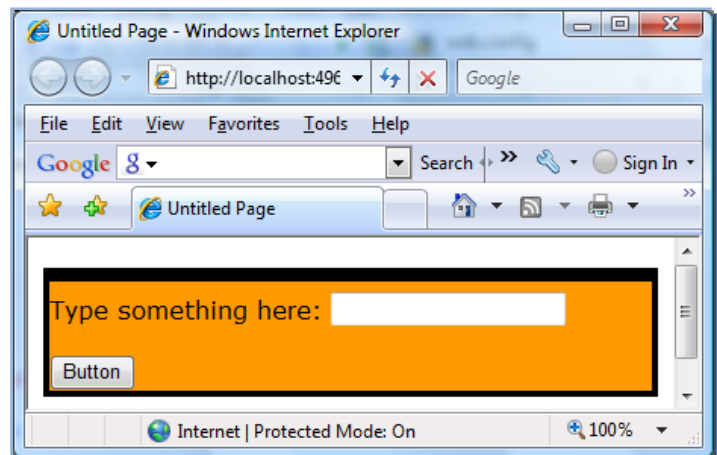


Figure13.2-6 New Style added Dialog Box

➤ 13.0 Styles, Themes and Master Pages

13.2.3 CSS Property Window

Once you've created a style, you have two easy options for modifying it in Visual Studio. Both revolve around the CSS Properties window, which allows you to dissect the formatting details of any style.

To show the CSS Properties window, open a web page in Visual Studio and choose View CSS Properties. The CSS Properties window is usually grouped with the Toolbox and Server Explorer windows at the left of the Visual Studio window.

Now that the CSS Properties window is visible, you can use it to view one of your styles. First, find the element or web control that uses the style attribute. Then, click to select it (in design view) or click somewhere in the element's start tag (in source view). Either way, the style information for that element will appear in the CSS Properties window. For example, Figure 13.5 -1shows what you'll see for the <div> element that was formatted in the previous section.

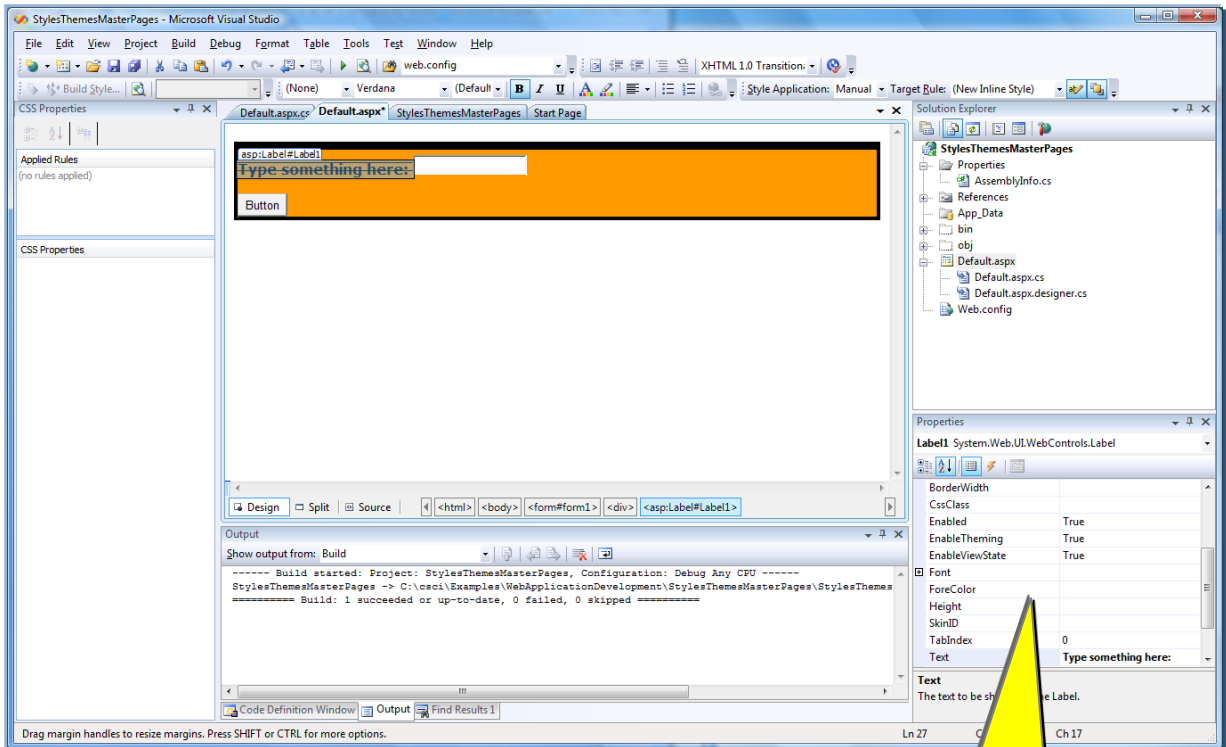


Figure 13.2-7 Properties Window

CSS Properties

➤ 13.0 Styles, Themes and Master Pages

The CSS Properties window provides an exhaustive list of all the formatting properties you can use in a style. This list is grouped into categories, and the properties in each category are sorted alphabetically. The ones that are currently set are displayed in bold.

You can use the CSS Properties window to modify existing style properties or set new ones, in the same way that you modify the properties for web controls using the Properties window. For example, in Figure 14-5 the font size is being changed.

Depending on your personal taste, you may find that the CSS Properties window is more convenient than the style builder because it gives you access to every style property at once. Or, you may prefer the more organized views in the style builder. (Your preference might also depend on how much screen real estate you have to devote to the CSS Properties window.) If you decide that you want to return to the style builder to change a style, the process is fairly straightforward. First, select the element that has the inline style. Next, look at the Applied Rules list at the top of the CSS Properties window, which should show the text `< inline style >`. Right-click that text and choose **Modify Style** to open the **Modify Style** dialog box, which looks identical to the **New Style** dialog box you considered earlier.

NOTE : You can't use the CSS Properties window to create a style. If you select an element that doesn't have a style applied, you won't see anything in the CSS Properties window (unless you select an element that's inside another element, and the containing element uses a style).

➤ 13.0 Styles, Themes and Master Pages

13.3 Themes

With the convenience of CSS styles, you might wonder why developers need anything more. The problem is that CSS rules are limited to a fixed set of style attributes. They allow you to reuse specific formatting details (fonts, borders, foreground and background colors, and so on), but they obviously can't control other aspects of ASP.NET controls. For example, the `CheckBoxList` control includes properties that control how it organizes items into rows and columns. Although these properties affect the visual appearance of the control, they're outside the scope of CSS, so you need to set them by hand. Additionally, you might want to define part of the behavior of the control along with the formatting. For example, you might want to standardize the selection mode of a `Calendar` control or the wrapping in a `TextBox`. This obviously isn't possible through CSS.

The themes feature fills this gap. Like CSS, themes allow you to define a set of style details that you can apply to controls in multiple pages. However, with CSS, themes aren't implemented by the browser. Instead, ASP.NET processes your themes when it creates the page.

NOTE : Themes don't replace styles. Instead, they complement each other. Styles are particularly useful when you want to apply the same formatting to web controls and ordinary HTML elements. Themes are indispensable when you want to configure control properties that can't be tailored with CSS.

All themes are application specific. To use a theme in a web application, you need to create a folder that defines it. This folder needs to be placed in a folder named `App_Themes`, which must be placed inside the top-level directory for your web application. In other words, a web application named `SuperCommerce` might have a theme named `FunkyTheme` in the folder `SuperCommerce\App_Themes\FunkyTheme`. An application can contain definitions for multiple themes, as long as each theme is in a separate folder. Only one theme can be active on a given page at a time.

➤ 13.0 Styles, Themes and Master Pages

To actually make your theme accomplish anything, you need to create at least one skin file in the theme folder. A skin file is a text file with the .skin extension. ASP.NET never serves skin files directly—instead, they're used behind the scenes to define a theme.

A skin file is essentially a list of control tags - with a twist. The control tags in a skin file don't need to completely define the control. Instead, they need to set only the properties that you want to standardize. For example, if you're trying to apply a consistent color scheme, you might be interested in setting only properties such as ForeColor and BackColor. When you add a control tag for the ListBox in the skin file, it might look like this:

```
<asp:ListBox runat="server" ForeColor="White" BackColor="Orange"/>
```

The runat="server" portion is always required. Everything else is optional. You should avoid setting the ID attribute in your skin, because the page that contains the ListBox needs to define a unique name for the control in the actual web page.

It's up to you whether you create multiple skin files or place all your control tags in a single skin file. Both approaches are equivalent, because [ASP.NET](#) treats all the skin files in a theme directory as part of the same theme definition. Often, it makes sense to put the control tags for complex controls (such as the data controls) in separate skin files. Figure 13.3-1 shows the relationship between themes and skins in more detail.

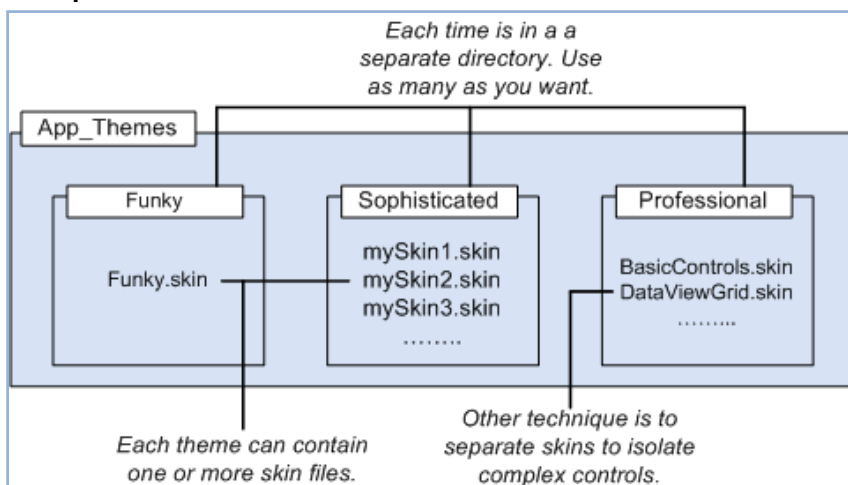


Figure 13.3-1 App Themes

➤ 13.0 Styles, Themes and Master Pages

[ASP.NET](#) also supports global themes. These are themes you place in the `:\inetpub\wwwroot\aspnet_client\system_web\v2.0.50727\Themes` folder. However, it's recommended that you use local themes, even if you want to create more than one website that has the same theme. Using local themes makes it easier to deploy your web application, and it gives you the flexibility to introduce site-specific differences in the future. If you have a local theme with the same name as a global theme, the local theme takes precedence, and the global theme is ignored. The themes are not merged together.

To add a theme to your project, select Website Add New Item, and choose Skin File (See Figure 13.3-2). Visual Studio will warn you that skin files need to be placed in a subfolder of the App_Themes folder and ask you whether that's what you intended. If you choose Yes, Visual Studio will create a folder with the same name as your theme file. You can then rename the folder and the file to whatever you'd like to use. Figure 13.3-3 shows an example with a theme that contains a single skin file.

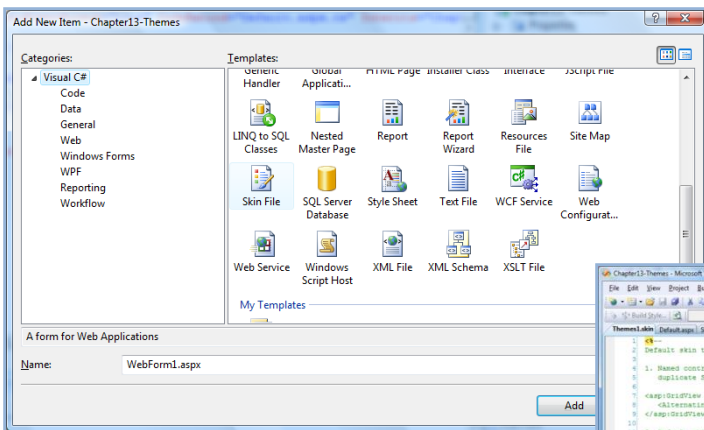


Figure 13.3-2 Adding a Skin

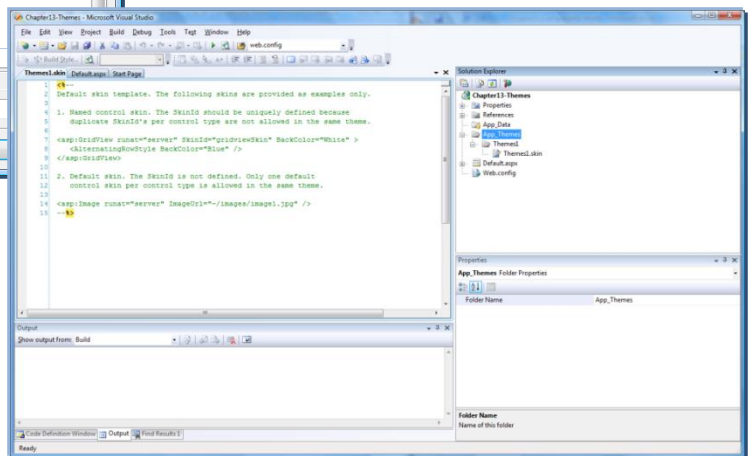


Figure 13.3-3 Theme with a Skin

➤ 13.0 Styles, Themes and Master Pages

Unfortunately, Visual Studio doesn't include any design-time support for creating themes, so it's up to you to copy and paste control tags from other web pages.

Here's a sample skin that sets background and foreground colors for several common controls:

```
<asp:ListBox runat="server" ForeColor="White" BackColor="Orange"/>  
<asp:TextBox runat="server" ForeColor="White" BackColor="Orange"/>  
<asp:Button runat="server" ForeColor="White" BackColor="Orange"/>
```

To apply the theme in a web page, you need to set the Theme attribute of the Page directive to the folder name for your theme. (ASP.NET will automatically scan all the skin files in that theme.)

```
<%@ Page Language="C#" AutoEventWireup="true" ... Theme="Themes1" %>
```

You can make this change by hand, or you can select the DOCUMENT object in the Properties window at design time and set the Theme property (which provides a handy drop-down list of all your web application's themes). Visual Studio will modify the Page directive accordingly.

When you apply a theme to a page, [ASP.NET](#) considers each control on your web page and checks your skin files to see whether they define any properties for that control. If [ASP.NET](#) finds a matching tag in the skin file, the information from the skin file overrides the current properties of the control.

Figure 13.3-4 shows the result of applying the Themes1 to a simple page. You'll notice that conflicting settings (such as the existing background for the list box) are overwritten. However, changes that don't conflict (such as the custom font for the buttons) are left in place.

Here is the example implementation of the **Themes1 .skin** into a set of controls (before and after)

➤ 13.0 Styles, Themes and Master Pages

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="Default.aspx.cs"
Inherits="Chapter13_Themes._Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
<title>Untitled Page</title>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:TextBox ID="TextBox1" Text="Testing"
Width="95" runat="server" /> <br />
<asp:ListBox ID="ListBox1" Width="100"
Height="100" runat="server" /> <br />
<asp:Button ID="Button1" Width="100" Text="Test
Button" runat="server"/>
</div>
</form>
</body>
</html>
```

```
<%@ Page Language="C#" Theme="Themes1"
AutoEventWireup="true"
CodeBehind="Default.aspx.cs"
Inherits="Chapter13_Themes._Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML
1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
<title>Untitled Page</title>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:TextBox ID="TextBox1" Text="Testing"
Width="95" runat="server" /> <br />
<asp:ListBox ID="ListBox1" Width="100"
Height="100" runat="server" /> <br />
<asp:Button ID="Button1" Width="100" Text="Test
Button" runat="server"/>
</div>
</form>
</body>
</html>
```

Example 13.3-1a Before Adding a Theme

Example 13.3-1b After Adding a Theme

Here are the two results – before and after applying the themes (themes1.skin)

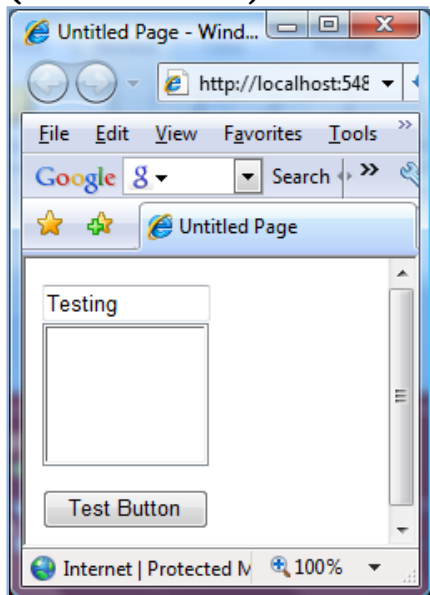


Figure 13.3-4a Before

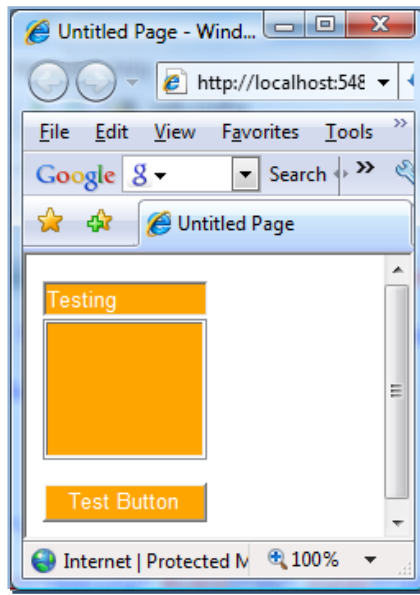


Figure 13.3-4b After

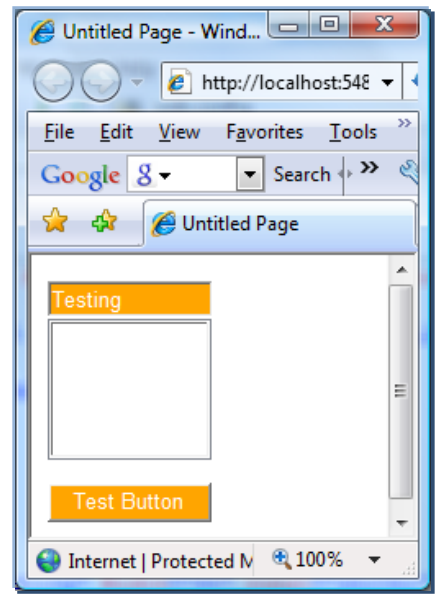


Figure 13.3-4c Controlled

➤ 13.0 Styles, Themes and Master Pages

13.3.1 Controlling Themes

You can decide if you wish to avoid adding the theme to a specific control of your page. For example in the example 13.3-1, if you decide to avoid the theme Themes1.skin on the List Box control, all you have to is to set the **EnableTheming** property of that control to **false**.

```
<asp:Button ID="Button1" runat="server" ... EnableTheming="false" />
```

So you get the output as follows (in Figure 13.3-4c)

13.3.2 Applying a Theme to entire Web Site.

Using the Page directive, you can bind a theme to a single page. However, you might decide that your theme is ready to be rolled out for the entire web application. The cleanest way to apply this theme is by configuring the <pages> element in the **web.config** file for your application, as shown here:

```
<configuration>
  <system.web>
    <pages theme="Theme1">
      ...
    </pages>
  </system.web>
</configuration>
```

If you want to use the style sheet behavior so that the theme doesn't overwrite conflicting control properties, use the **StyleSheetTheme** attribute instead of Theme:

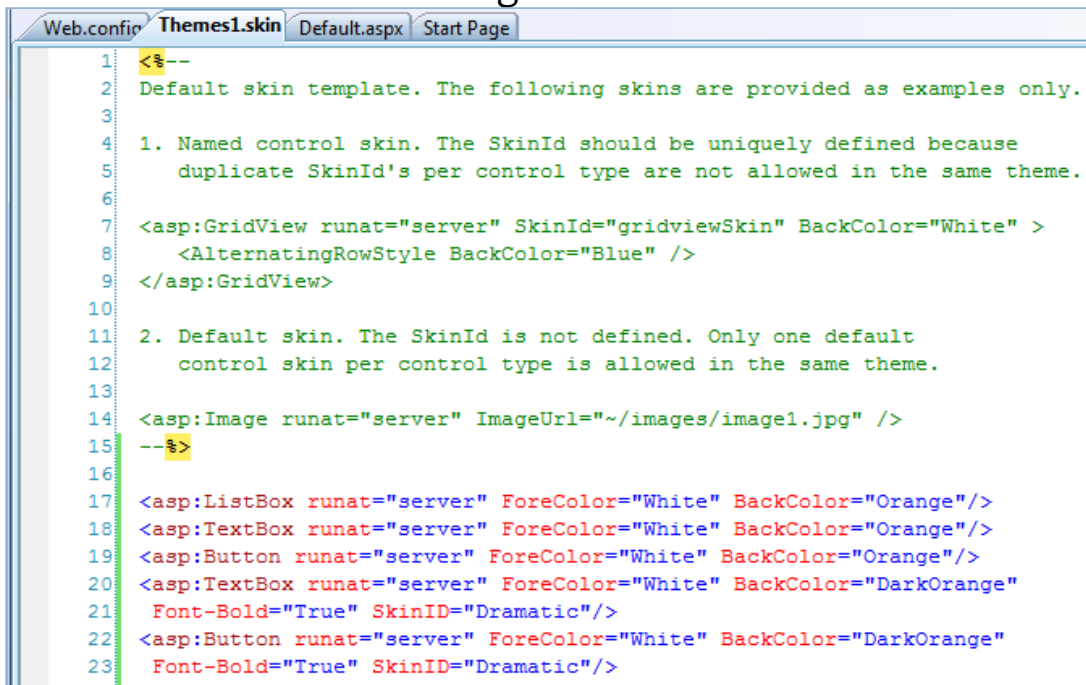
```
<configuration>
  <system.web>
    <pages styleSheetTheme="Theme1">
      ...
    </pages>
  </system.web>
</configuration>
```


➤ 13.0 Styles, Themes and Master Pages

Either way, when you specify a theme in the **web.config** file, the theme will be applied throughout all the pages in your website, provided these pages don't have their own theme settings. If a page specifies the Theme attribute, the page setting will take precedence over the **web.config** setting. If your page specifies the Theme or **StyleSheetTheme** attribute with a blank string (Theme=""), no theme will be applied at all.

Using this technique, it's just as easy to apply a theme to part of a web application. For example, you can create a separate web.config file for each subfolder and use the <pages> setting to configure different themes.

Here is an example of modified Themes1.skin (Example 13.3-2) file along with the modified web.config file.



```
1 <!--
2 Default skin template. The following skins are provided as examples only.
3
4 1. Named control skin. The SkinId should be uniquely defined because
5    duplicate SkinId's per control type are not allowed in the same theme.
6
7 <asp:GridView runat="server" SkinId="gridviewSkin" BackColor="White" >
8   <AlternatingRowStyle BackColor="Blue" />
9 </asp:GridView>
10
11 2. Default skin. The SkinId is not defined. Only one default
12   control skin per control type is allowed in the same theme.
13
14 <asp:Image runat="server" ImageUrl="~/images/image1.jpg" />
15 -->
16
17 <asp:ListBox runat="server" ForeColor="White" BackColor="Orange"/>
18 <asp:TextBox runat="server" ForeColor="White" BackColor="Orange"/>
19 <asp:Button runat="server" ForeColor="White" BackColor="Orange"/>
20 <asp:TextBox runat="server" ForeColor="White" BackColor="DarkOrange"
21   Font-Bold="True" SkinID="Dramatic"/>
22 <asp:Button runat="server" ForeColor="White" BackColor="DarkOrange"
23   Font-Bold="True" SkinID="Dramatic"/>
```

Example 13.3-2
Skin Template

Figure 13.3-5a Before

```
<pages theme="Themes1">
  <controls>
    <add tagPrefix="asp" namespace="System.Web.UI" assembly="System.Web.Extensions, Version=3.5.0.0,
      Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
    <add tagPrefix="asp" namespace="System.Web.UI.WebControls" assembly="System.Web.Extensions, Version=3.5.0.0,
      Culture=neutral, PublicKeyToken=31BF3856AD364E35"/>
  </controls>
</pages>
```

Figure 13.3-5b Skin Template added in web.config file

➤ 13.0 Styles, Themes and Master Pages

Here is the result of the example. In this the Themes-"" tag is removed in the <@page....>

```
<%@ Page Language="C#" AutoEventWireup="true"  
CodeBehind="Default.aspx.cs" Inherits="Chapter13_Themes._Default" %>
```

```
<%@ Page Language="C#" AutoEventWireup="true"  
CodeBehind="Default.aspx.cs"  
Inherits="Chapter13_Themes._Default" %>  
  
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0  
Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-  
transitional.dtd">  
  
<html xmlns="http://www.w3.org/1999/xhtml" >  
<head runat="server">  
  <title>Untitled Page</title>  
</head>  
<body>  
  <form id="form1" runat="server">  
    <div>  
      <asp:TextBox ID="TextBox1" Text="Testing" Width="95"  
        runat="server" /> <br />  
      <asp:ListBox ID="ListBox1" Width="100" Height="100"  
        runat="server" EnableTheming="false" /> <br />  
      <asp:Button ID="Button1" Width="100" Text="Test Button"  
        runat="server"/>  
    </div>  
  </form>  
</body>  
</html>
```

Example 13.3-3 Skin is now coming from web.config. file

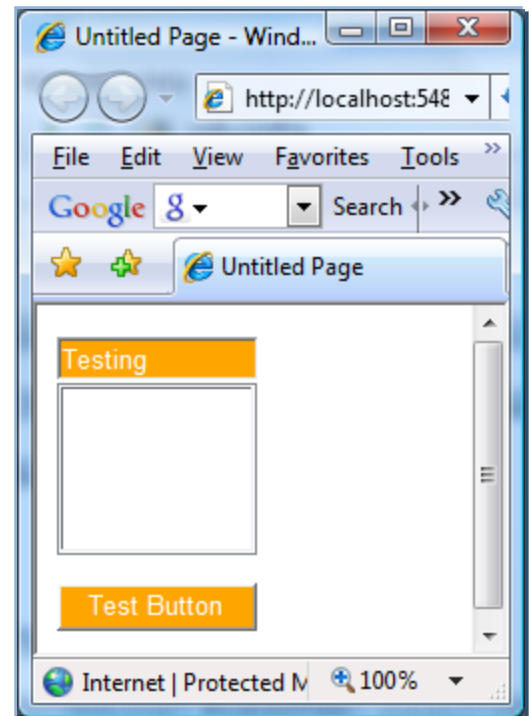


Figure 13.3-6 Controlled Skin

Figure 13.3-6 is the result. But the results not the same as in Figure 13.3-4b as this now uses the theme specified in the web.config file.

➤ 13.0 Styles, Themes and Master Pages

13.3.2 Using a SkinID

Ordinarily, if you create more than one theme for the same control, [ASP.NET](#) will give you a build error stating that you can have only a single default skin for each control. To get around this problem, you need to create a named skin by supplying a **SkinID** attribute. As an example there are multiple TextBox attributes defined in the web.config file. But the “DardOrange” one has a **SkinID=“Dramatic”**.

The catch is that named skins aren't applied automatically like default skins. To use a named skin, you need to set the SkinID of the control on your web page to match. You can choose this value from a drop-down list that Visual Studio creates based on all your defined skin names, or you can type it in by hand:

```
<asp:Button ID="Button1" runat="server"
SkinID="Dramatic" />
```

If you don't like the opt-in model for themes, you can make all your skins named. That way, they'll never be applied unless you set the control's SkinID.

[ASP.NET](#) is intelligent enough to catch it if you try to use a skin name that doesn't exist, in which case you'll get a build warning. The control will then behave as though you set EnableTheming to false, which means it will ignore the corresponding default skin. Here is the code example of the presentation.(Example 13.3-4). The result is shown on Figure 13.3-8

```
<body>
  <form id="form1" runat="server">
    <div>
      <asp:TextBox ID="TextBox1"
        Text="Testing" Width="95"
        runat="server" /> <br />
      <asp:ListBox ID="ListBox1"
        Width="100" Height="100"
        runat="server"
        EnableTheming="false" /> <br />
      <asp:Button ID="Button1"
        Width="100" Text="Test Button"
        SkinID="Dramatic"
        runat="server"/>
    </div>
  </form>
</body>
```

Example 13.3-4 Setting a SkinID

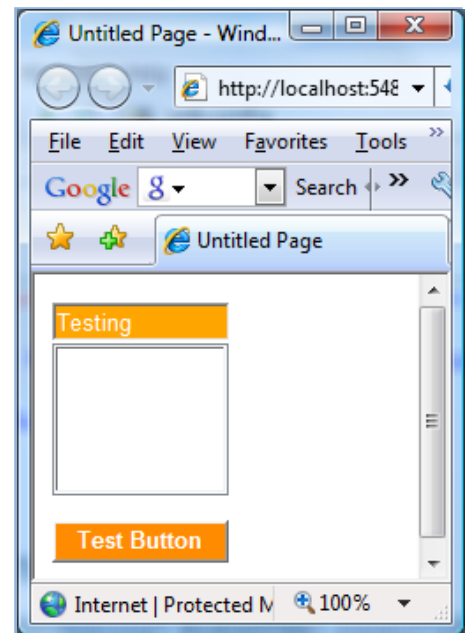


Figure 13.3-7 Controlled Skin with SkinID

➤ 13.0 Styles, Themes and Master Pages

13.4 Master Pages

ASP.NET master pages allow you to create a consistent layout for the pages in your application. A single master page defines the look and feel and standard behavior that you want for all of the pages (or a group of pages) in your application. You can then create individual content pages that contain the content you want to display. When users request the content pages, they merge with the master page to produce output that combines the layout of the master page with the content from the content page.

Student is assigned to read this chapter.