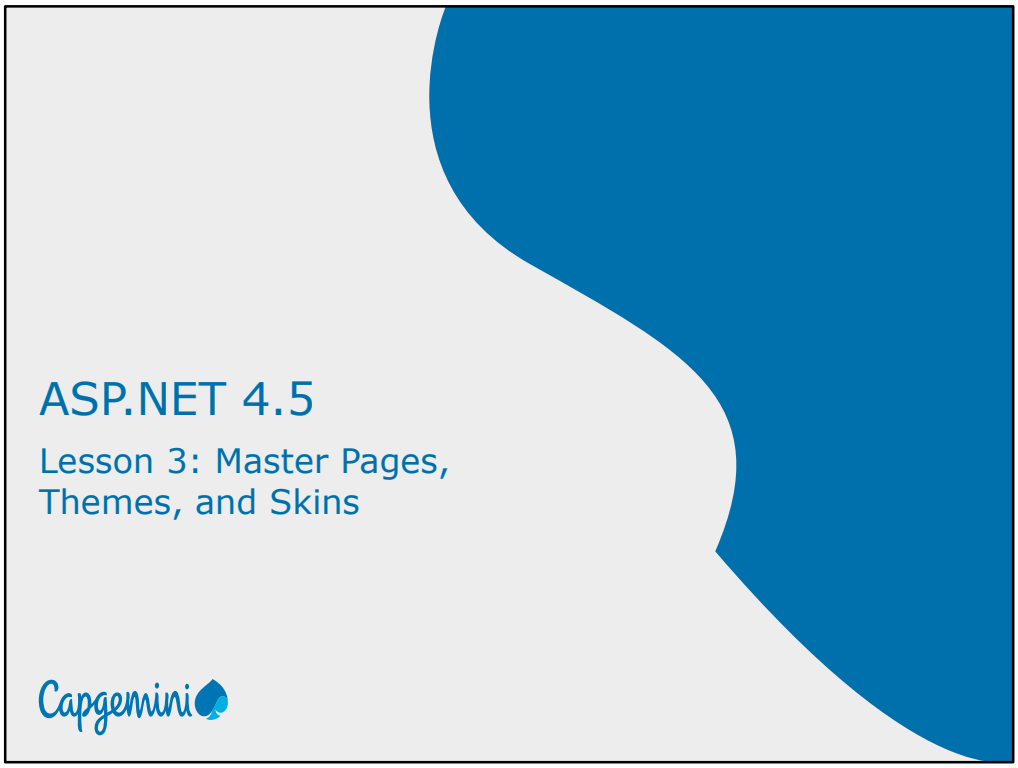

ASP.NET 4.5
Lesson 3: Master Pages,
Themes, and Skins
Capgemini

Lesson Objectives

- In this lesson, you will learn:
  - Working with Master Pages
  - Themes and Skins

3.1: Why Master Pages?

Usage

- Web sites have content which is common to all the pages or most of the pages
- A typical website may have all the pages with header, footer, and navigation sections to display content
- To achieve this, a template called as 'Master Page' was introduced in ASP.Net 2.0.

Why Master Pages?

- Most of the web sites today have common elements used throughout the entire application or most of the web pages. For instance, a typical web site may have web pages with a layout which consists of **header section**, **footer section**, and the **navigation section**. The **ASP.Net 2.0** release allowed us to create a single template page that can be used as a foundation for all or most of the pages in the ASP.Net web site. These **templates** are called as **Master Pages**.
- It is not so that before master pages web sites did not have common elements across pages. In many cases, the way you had to put these common elements across pages was quite difficult. Earlier developers simply copied and pasted common sections on all pages that required them. But whenever a change was required it was a tedious task to change the sections in all the pages.
- In **Classic ASP**, we can create an **include** file which contains the layout and in all the ASP pages we can include this file. But the problem with **include** files was that it required to keep track of the HTML tags and see that they are opened and closed appropriately in the respective files. That is, suppose a tag was opened in **header include** file. Then it had to be closed appropriately in the **navigation** or the **footer include file** to complete the tag and provide the page the look as required.
- In **ASP.Net 1.1**, the developers made use of **user controls** and included these user controls in the pages that required to have common elements presented. However, the problem with this option was that when you worked in the Design view, it appeared as grey boxes in the IDE. This made the development of pages very difficult, as you could not visualize the page as to how it would look after it was built.
- As a result, the **ASP.Net team** came up with **Master pages** to take care of the issues posed by **include files** and **user controls**. A **Master page** enables you to share the same content among multiple content pages in a website. You can use a **Master page** to create a common page layout. For example, if you want all the pages in your website to share a three-column layout, you can create the layout once in a **Master page** and apply the layout to multiple content pages.
- You also can use **Master pages** to display common content in multiple pages. For example, if you want to display a standard header and footer in each page in your website, then you can create the standard header and footer in a Master Page.
- By taking advantage of **Master pages**, you can make your website easier to maintain, extend, and modify. If you need to add a new page to your website that looks just like the other pages in your website, then you simply need to apply the same Master page to the new content page. If you decide to completely modify the design of your website, then you do not need to change every content page. You can modify just a single Master page to dramatically change the appearance of all the pages in your application.
- Also the **Master pages** live outside the application, while **user controls** live within your application and are destined to get duplicated.
- In this section, you will learn how to create Master pages and apply Master pages to content pages.

For example, try ordering a book on Amazon. While you search, click through the links, and then head to your shopping cart, you'll always see a continuous user interface with a common header at the top, a set of navigation links on the left, and a footer at the bottom.

3.2: Working with Master Pages

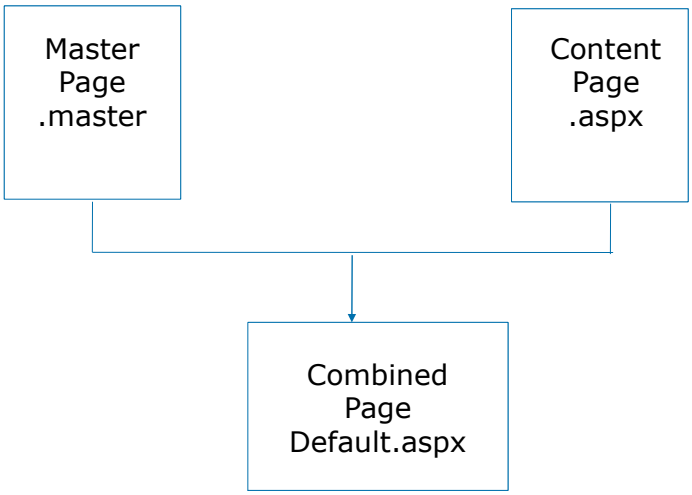
Methodology

- A master file has to be created. It is the template referenced by a subpage or content page
- Master Pages use a .master extension, and the content page uses a .aspx extension

Working with Master Pages:
Master pages are an easy way to provide a template that can be used by any number of ASP.Net pages in your application.
You create a Master page by creating a file that ends with the .master extension. You can locate a Master page file at any place within an application. The master file is the template, which is referenced by a subpage or content page. A content page is a .aspx page. In the .master file, you can place anything you like. This can include the header, navigation, and footer sections, which will be used across the web application.
The content page contains all the page content except for the master page elements. At runtime, the ASP.Net engine combines these elements into a single page for the end user.

3.2: Working with Master Pages

Methodology (Contd…)



Working with Master Pages:
The diagram on the above slide depicts how the master page and content page is put together by the ASP.Net engine.
While working with Master pages you can visually see the template in the IDE which makes the development easy for the content pages. While you are working on the content page, all the templated items are shaded grey and are not editable. On the template page, only some areas are editable which are known as content areas. The content areas are originally defined in the master page itself. The content areas are essentially the areas of the page that the content pages can use. You can have more than one content area in the master page.
Companies and organizations find master pages ideal, as the technology closely models their typical business needs. This process makes it quite easy for the company to keep a consistent look and feel across their web applications like intranet.

3.3: Coding a Master Page

The Process

- You create a Master page in Visual Web Developer by selecting the Website menu option, then Add New Item, and then selecting the Master Page item
- Include the Master directive and specify the language.
  - <%@ Master Language="C#" %>
- You can use server controls, raw HTML, images and so on, like a .aspx page, to build a master page.

Coding a Master Page:
A master page can be added to your application as you would add .aspx page. You create a Master Page in Visual Web Developer by selecting the Website menu option, then Add New Item, and then selecting the Master Page item. The Master page is added to your application and the master page directive is added along with the language for coding. You code a master page like a .aspx page. You can user server controls, raw HTML, images, events, and so on.
A sample master page goes as follows:

```
<%@ Master Language="C# %>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head  runat="server">
<style type="text/css">
        html
        {
          background-color:silver;
          font:14px Arial,Sans-Serif;
        }
        .content
        {
          margin:auto;
          width:700px;
          background-color:white;
          border:Solid 1px black;
        }
```

Coding a Master Page (contd.):

```
.leftColumn
        {
         float:left;
         padding:5px;
         width:200px;
         border-right:Solid 1px black;
         height:700px;
         }
         .rightColumn
         {
          float:left;
          padding:5px;
          }
          .clear
          {
          clear:both;
          }
   </style>
   <title>Simple Master</title> </head>
   <body>
     <form id="form1" runat="server">
     <div class="content">
     <div class="leftColumn">
     <asp:ContentPlaceHolder  id="ContentPlaceHolder1"
runat="server"/>
     </div> <div class="rightColumn">
     <asp:ContentPlaceHolder id="ContentPlaceHolder2"
runat="server"/>
     </div> <br class="clear" />
     </div> </form>
     </body>
</html>
```

- Notice the new Server Control **<asp:ContentPlaceHolder >** control.
- When the Master Page is merged with a particular content page, the content from the content page appears in the areas marked by **ContentPlaceHolder** controls. You can add as many **ContentPlaceHolders** to a Master Page as you need.
- In this example, we have two defined areas where the content page can place content.

3.4: Coding a Content Page

The Process

- Add a new Web form to the application to create a content page, and select the option "Select Master Page"
- The Web form can then be associated with the Master page
- The MasterPageFile attribute within the page directive points to the master page which is inherited by the content page

Coding a Content Page:

Once the master page has been added to the application, this template can be used for any content pages in the application. To add a new Web form to your application, select the option "Select Master Page" in the dialog box unlike just a typical Web form. Once the name for the content page has been provided and you click Add button a new dialog box pops up which allows to select the master page. This will create the content page. The Master page is associated with the content page through the MasterPageFile attribute included in the <%@ Page %> directive. This attribute contains the virtual path to a Master Page.

Notice that the content page does not contain any of the standard opening and closing HTML tags. All these tags are contained in the Master page. This content page appears to be simple. However, if you view the page in Design view , you can see the visually inherited master page.

All the content contained in the content page must be added with Content controls. You must place all the content contained in a content page within the Content controls. If you attempt to place any content outside these controls, then you get an exception.

The Content control includes a ContentPlaceHolderID property. This property points to the ID of a ContentPlaceHolder control contained in the Master Page.

Coding a Content Page:

- Within a **Content control**, you can place anything that you would normally add to an ASP.Net page, including XHTML tags and ASP.Net controls.
- The Content page code will appear as follows once the master page has been included:

```
<%@ Page Language="C#"
MasterPageFile="~/SimpleMaster.master" %>
 <asp:Content ID="Content1"
       ContentPlaceHolderID="ContentPlaceHolder1"
      Runat="Server">
      Content in the first column
    <br />Content in the first column
    <br />Content in the first column
    <br />Content in the first column
    <br />Content in the first column
</asp:Content>
<asp:Content ID="Content2"
       ContentPlaceHolderID="ContentPlaceHolder2"
       Runat="Server">
      Content in the second column
    <br />Content in the second column
    <br />Content in the second column
    <br />Content in the second column
    <br />Content in the second column
</asp:Content>
```

- The **<asp:Content>** server control is defined content area that maps to a specific **<asp:ContentPlaceHolder>** server control on the master page. Within the content page, you do not have to worry about specifying the location of the content because this is already defined within the master page. Therefore a developer's concern is to place the appropriate content within the provided content sections, allowing the master page to work for you.
- Note that a Master page can be applied at **page level** for respective page and **application level** for all the pages. The precedence is always given to the page level master page.
- The Master page can be applied at **page level** by using the page directive:

```
<%@ Page Language="C#"
MasterPageFile="~/SimpleMaster.master" %>
```

- You can also specify the master page in the **web.config** to apply it at **application level**. In **web.config** you can specify it as follows:

```
<configuration>
  <system.web>
      pages masterPageFile="~/Samplemaster.master" />
  </system.web>
</configuration>
```

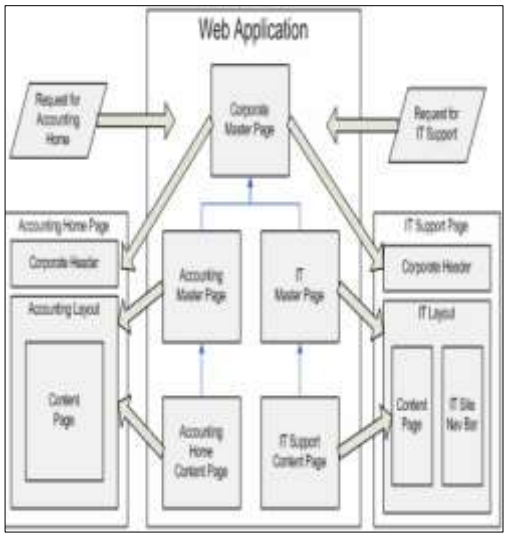
Demo

- Creating and applying Master Pages

3.5: Nesting of Master Pages

The Process

- Variations are required in master page by different groups.
- In ASP.Net 4.5, you can have master page within a master page
- You can define a global master page and the groups can define their own master pages by inheriting the global master page



Nesting Master Page:

Master pages provide to help you create templated web application. However, many organizations have various groups/departments that may want to have some layout specific to them. In ASP.Net , you can have a master page within a master page.

This technique can be useful when your application is broken into sub-applications that need to inherit a common branded look but still be able to define their own customized layout template within the brand.

Hence you can define a Global master page and every group can inherit this master page and add their own elements in the master page as well.

The following code listing shows the nesting of master pages. You first need to create the top level master page which is globally used by all pages.

```
<%Master Language=“C#”>
<html xmlns="http://www.w3.org/1999/xhtml“>
<head runat=“server”>
      <title>Capgemini Main</title> </head>
<body>
<form id=“form1” runat=“server”>
   <p> <asp:Label ID=“Label1”
runat=“server” BackColor=“LightGray”
   BorderColor=“Black” BorderStyle=“Solid” Font-Size=“XX-
Large”> Capgemini
   </asp:Label> </p>
    <asp:ContentPlaceHolder ID=“ContentPlaceHolder”
runat=“server”>
    </asp:ContentPlaceHolder>
</form></body></html>
```

3.5 Nesting of Master Pages

Nested Master Pages

- Master pages can be nested, with one master page referencing another as its master.
- Nested master pages allow you to create componentized master pages.
- For example, a large site might contain an overall master page that defines the look of the site. Different site content partners can then define their own child master pages that reference the site master and that in turn define the look for that partner's content.
- A child master page has the file name extension .master, as with any master page.
- The child master page typically contains content controls that are mapped to content placeholders on the parent master page.

In this respect, the child master page is laid out like any content page. However, the child master page also has content placeholders of its own to display content supplied by its own child pages.

Nesting Master Page (contd.):

- The top level master page has the **ContentPlaceHolder** server control with the ID of ContentPlaceHolder1.
- Now let us define a **submaster page**, that is Capgemini-ILA.master:
- creating a **submaster page,** add a **master page** to your application and remove all the code except the directive line. Then you create a **Content server control**. By default, Visual Studio does not allow to create submaster pages. The **ContentPlaceHolderId** attribute of the Content control ties this content area to content area **ContentPlaceHolder1** which is defined in the main master page.
- You can define as many content areas as you like even in the submaster page.
- Now, define the **content page** that uses the **submaster page**:

```
<Master MasterPageFile="~/CapgeminiMain.master" %>
<asp:Content ID="Content1"
ContentPalceHolderId="ContentPlaceHolder1" runat="server">
    <asp:Label ID="Label1" runat="server" BackColor="#E0E0E0"
BorderColor="Black"
     BorderStyle="Dotted" Font-Size="Large">
     Capgemini-ILA</asp:Label><br /><hr />
     <asp:ContentPlaceHolder ID="ContentPlaceHolder2"
runat="server">
     </asp:ContentPlaceHolder>
</asp:Content>
```

- As you can see in this content page, the value **MasterPageFile** attribute in the **Page directive** is the **submaster page** that you created.
- You can try this example to get a better understanding of nesting of master pages.

```
<%@ Page Language="C#" MasterPageFile="~/Capgemini-ILA.master"
>
  <asp:Conten ID="Content"
ContentPlaceHolderId="ContentPlaceHolder2"
   runat="server">
   Hello World. Welcome to India Learning Academy
   </asp:Content>
```

3.5: Nesting of Master Pages

Demo

- Creating Nested Master Pages

3.6: Themes

Concept

- Themes are text-based style definitions in ASP.NET
- Very similar to Cascading Style Sheets in a manner visual styles for the web pages can be defined.
- Themes can be applied at the application, page, or server control level

Themes:
When you build a web application, it usually has a similar look and feel across all its pages. Generally, the applications use similar fonts, colors and server control styles across all the pages.
Your choice could be to apply the styles individually to each and every page and every server control. Or you can alternatively choose to centrally specify these styles and let all pages in the application apply these styles.
ASP.Net 2.0 introduces themes which can allow you to define these styles centrally. Themes are text-based style definitions. A theme is a collection of property settings that allow you to define the look of pages and controls, and then apply the look consistently across pages in a Web application.
Themes are very similar to Cascading Style Sheets in a manner that you can define visual styles for the web pages. Themes can be applied at the application, page, or server control level.
Themes are made up of a set of elements: skins, cascading style sheets (CSS), images, and other resources. At a minimum, a theme will contain skins. Themes are defined in special directories in your Web site or on your Web server.

3.7: Skins

Concept of Skins

- A Skin is a definition of styles applied to the server control in your ASP.Net page
- Skins can work in combination with CSS files or images
- A Skin file is created with a .skin extension

Skins:
A skin file has the file name extension .skin and contains property settings for individual controls such as Button, Label, TextBox, or Calendar controls. Control skin settings are like the control markup itself, but contain only the properties you want to set as part of the theme. Skins can work in combination with CSS files or images.
A .skin file can contain one or more control skins for one or more control types. You can define skins in a separate file for each control or define all the skins for a theme in a single file.
There are two types of control skins:

A default skin automatically applies to all controls of the same type when a theme is applied to a page.
A named skin is a control skin with a SkinID property set.

3.8: Applying a Theme

Methods

- Following options are available for applying a theme:
  - Applying a Theme at page level through page directive:

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

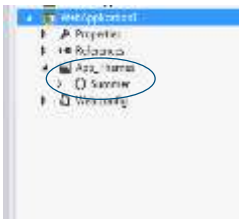
  - Applying a Theme at application level through web.config file:

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

3.9: Creating Themes

The Process

- Follow the given steps to create a Theme:
  - Add a Theme folder to your web application
  - Right-click and add ASP.Net folder. The Theme folder will be displayed in the Solution explorer as follows:



Creating Themes:
As you can see on the slide, we have added the Themes folder which is named as App_Themes. Within this folder we will create a Theme named as Summer. You can add as many Themes in this folder according to requirement.

3.10: Creating a Skin

The Process

- Follow the given steps to create a Skin:
  - Right click the Summer folder and add a new item.
  - From the dialog box, select Skin File.

Creating a Skin:
To add the skin file to the Summer Theme that we have created, right-click the Summer folder and select Add New Item.
In the dialog box, select the Skin file option and provide the name. For our example the file name is Summer.skin.
The code in the Summer.skin file will be as follows:

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

In this code, we have placed definitions for three server controls, namely Label, TextBox, and Button. The control definition must include the runat="server" attribute.

3.11: Applying a Theme to ASP.Net Page

The Process

- You can follow the given steps to apply the theme to an ASP.NET page:
  - Design the ASP.Net page as required
  - Specify the attribute Theme in the page level directive as follows:

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


Applying the Theme to ASP.NET Page:
Design the ASP.Net page layout as required. Apply the Theme to the page by specifying the Theme attribute in the Page level directive as shown on the slide.
Write the following code for the click event of the button:

Execute the application, and you will be able to see that the Label, TextBox, and Button have the style as defined in the skin file.

```
protected void Button1_Click(object sender, EventArgs e)
   {
      Label1.Text = "Welcome  " + TextBox1.Text + "to our portal";
   }
```

3.12: Defining Multiple Skin Options

The Process

- You can define multiple skin options for one control in the skin file using Skin ID attribute

```
<asp:TextBox runat="server" ForeColor="#004000"
Font-Names="Verdana" Font-Size="Small"
BorderStyle="Dotted"
BorderWidth="5px"  BorderColor="#004000" Font-
Bold="False"
SkinID="TextBoxDotted"/>
```


Defining Multiple Skin Options:
Skins defined in your theme apply to all control instances in the application or pages to which the theme is applied.
In some cases, you might want to apply a specific set of properties to an individual control. You can do that by creating a named skin (an entry in a .skin file that has a SkinID property set) and then applying it to individual controls. In the skin file, include the multiple definitions for a control as follows:

```
<asp:TextBox runat="server" ForeColore="#004000"
Font-Names="Verdana" Font-Size="Small" BorderStyle="Solid"
BorderWidth="1px"
BorderColor="#004000" Font-Bold="True" />
<asp:TextBox runat="server" ForeColore="#004000"
Font-Names="Verdana" Font-Size="Small" BorderStyle="Dotted"
BorderWidth="5px"  BorderColor="#004000" Font-Bold="False"
SkinID="TextBoxDotted"/>
<asp:TextBox runat="server" ForeColore="#004000"
Font-Names="Verdana" Font-Size="Small" BorderStyle="Dashed"
BorderWidth="3px"  BorderColor="#004000" Font-Bold="False"
SkinID="TextBoxDashed"/>
```

Notice that in the skin file, we have three different definitions for the textbox control. The last two have the SkinID attribute specified. The value of the SkinID attribute can be anything to uniquely identify the style.

3.12: Defining Multiple Skin Options

The Process (Contd…)

- Notice the skinID attribute included
- Specify the skinID attribute for the control to which you would want to apply the specific style like

```
<asp:TextBox ID="TextBox1" runat="server"
    SkinId="TextBoxDotted">TextBox1</asp:TextBox>
```


Defining Multiple Skin Options (contd.):

- Now the skin can be applied to the control. By specifying the **SkinId**, we can associate the control with the specific style.
- The **SkinId** attribute is specified for **TextBox1**. However, we have not specified any **SkinId** for the **TextBox2**. This indicates that it will be using the default style definition.
- If the page theme does not include a control skin that matches the SkinID property, then the control uses the default skin for that control type.
- **Page-level Theming** can be enabled or disabled by setting the EnableTheming attribute to True or False.

```
<%@ Page Language="C#" EnableTheming="True"%>
```

Control-level Theming: Even if the Theming is enabled at the page-level, one can restrict the controls from beiing applied with the Theme by setting the EnableTheming property to true or false For example,

```
<asp:TextBox runat="server" ID="txt_name" EnableTheming="false">
</asp:TextBox>
```

```
<asp:TextBox ID="TextBox1" runat="server"
    SkinId="TextBoxDotted">TextBox1</asp:TextBox>
<asp:TextBox ID="TextBox2" runat="server"> TextBox2</asp:TextBox>
```

3.12.1 Demo on Themes and Skins

Demo

- Creating Themes and applying it at:
  - Page Level
  - Application Level

3.13: Dynamically Changing Themes

The Process

- To dynamically assign a Theme to your page,use the following code:

```
Protected void Page_PreInit(object sender, EventArgs e)
{
 Page.Theme = Request.QueryString[“ChangedTheme”];
}
```

3.14: Dynamically Assigning Skins to Control

The Process

- To dynamically change the skin for a control, use the following code:

```
Protected void Page_PreInit(object sender, EventArgs e)
{
  TextBox1.SkinID="TextBoxDashed";
}
```

3.14.1 Demo on Dynamically Changing Themes

Demo

- Creating themes and dynamically changing them at run time

3.15: Best Practices and Guidelines

Master Pages

- Use master pages to centralize the common functionality (for example: menus) of all pages so that updates can be done in just one place
- While working with elements on master pages, use server controls instead of HTML controls
  - This gives programmability of controls in the content pages
    - For example: Setting label on master page from content pages
  - It gives ease of maintenance, as well.

3.15: Best Practices and Guidelines

Themes

- Use themes and skins for ASP.Net application, wherever possible
  - It allows defining the layout of server controls and making them look coherent to the overall application.
- Protect the global and application theme directories with proper access control settings
  - Only trusted users should be allowed to write files to the theme directories

3.15: Best Practices and Guidelines

Themes (Contd…)

- Do not use themes from an un-trusted source
  - Always examine any themes from outside your organization for malicious code before using them on you Web site
- Do not expose the theme name in query data
  - Malicious users can use this information to use themes that are unknown to the developer and thereby expose sensitive information
- Disable theme, when a control or page already has a predefined look that should not be overridden by theme

Summary

- In this lesson, you have learnt:
  - Working with Master Pages
  - Themes and Skins

Review – Questions

- Question 1: Which of the following Page Events can be used to dynamically apply a Style:
  - Page_Load
  - Page_Init
  - Page_PreInit
  - Page_Render
- Question 2: Named skin will have:
  - SkinName
  - SkinId
  - SkinType
  - SkinSource