Master Pages & Themes

UNIT - 5

Master Page

- ASP.NET master pages allow you to create a consistent layout for the pages in your application.
- 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.
- A master page defines both the static page layout and the regions tha t can be edited by the ASP.NET pages that use the master page.

Master Page

- These content editable regions are indicated by ContentPlaceHolder cont rol, which can be seen within the content
- Our master page has a single ContentPlaceHolder (MainContent), but m aster page's may have multiple ContentPlaceHolders.
- The **@ Master** directive defines it as a master page.
  The master page contains a placeholder tag for individual content.
- The **id** attribute identifies the placeholder, allowing many placeholders in the same master page.
- The master page can also contain code, allowing dynamic content.
  - <%@ Master Language="VB" AutoEventWireup="true" CodeFile="Site.master.vb" Inherits="Site" %>
  - <asp:contentplaceholder id="MainContent" runat="server">

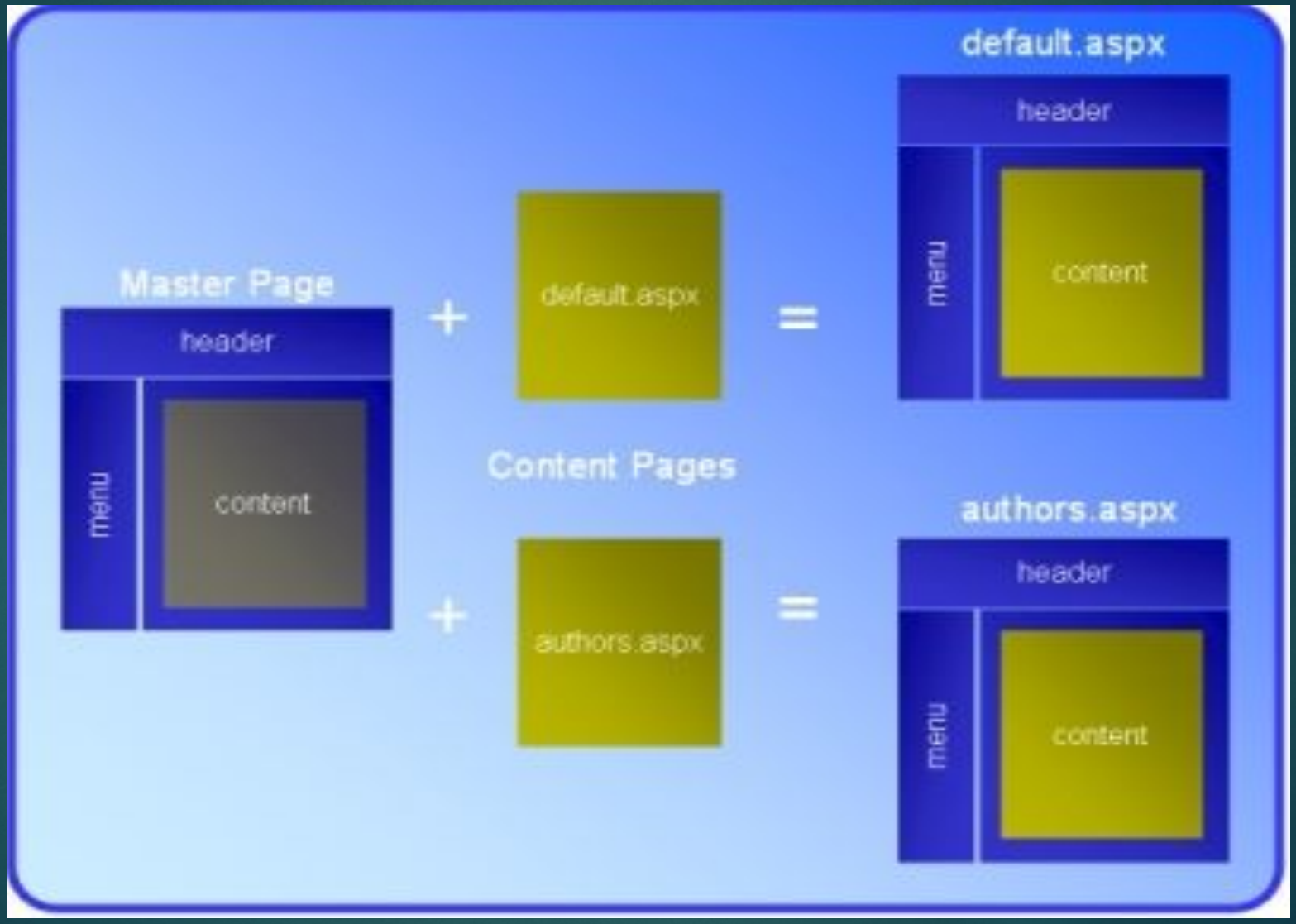
Master Page

- In Content Page,
- @Page directive there's a reference to the master page file used (Master PageFile="~/Site.master"), and
  the ASP.NET page's markup contains a Content control for each of the ContentPlaceHolder controls defined in the
  master page, with the control's ContentPlaceHolderID mapping the Con tent control to a specific ContentPlaceHolder.
- Content control is where you place the markup you want to appear in th e corresponding ContentPlaceHolder.
  - <%@ Page Language="VB" MasterPageFile="~/Site.master" AutoEventWireup="true" CodeFile="Default.aspx.vb" Inherits="_Default" Title="Untitled Page" %>
  - <asp:Content ID="Content1" ContentPlaceHolderID="MainContent" Runat="Server">

    //content

    </asp:Content>

Master Page


default.aspx
header
menu
content
Master Page
header
menu
content
+
default.aspx
=
Content Pages
authors.aspx
header
menu
content
+
authors.aspx
=

Requirement Of a Master Page

- The master pages can be used to accomplish the following:
- Creating a set of controls that are common across all the web pages and attaching them to all the web pages.
- A centralized way to change the above created set of controls which will effectively change all the web pages.
- Dynamically changing the common UI elements on master page from content pages based on user preferences.

MasterPage

- The extension of MasterPage is '**.master**'.
- MasterPage cannot be directly accessed from the client because it just acts as a template for the other Content Pages.
- In a MasterPage we can have content either inside **ContentPlaceHolder** or outside it. Only content inside the ContentPlaceHolder can be customized in the Content Page.
- We can have multiple masters in one web application.
- A MasterPage can have another MasterPage as Master to it.

MasterPage

- The content page content can be placed only inside the content tag.
- Controls of MasterPage can be programmed in the MasterPage and content page but a content page control will never be programmed in MasterPage.
- A master page of one web application cannot be used in another web application.

MasterPage

- The **MasterPageFile** property of a webform can be set dynamically and it should be done either in or before the **Page_PreInit** event of the WebForm. **Page.MasterPageFile = "MasterPage.master".** The dynamically set Master Page must have the ContentPlaceHolder whose content has been customized in the WebForm.
- The order in which events are raised: **Load (Page)** a **Load (Master) a LoadComplete (Page)** i.e. if we want to overwrite something already done in Load event handler of Master then it should be coded in the LoadComplete event of the page.

Example:

- Adding a MasterPage to the Project
- Add a new MasterPage file (MainMaster.master) to the Web Application.
- Change the Id of ContentPlaceHolder in <Head> to "cphHead" and the Id "ContentPlaceHolder1" to "cphFirst"
- Add one more ContentPlaceHolder (cphSecond) to Master page.
- To the master page add some header, footer and some default content for both the content place holders.

ContentPlaceHolder

```
<form id="form1" runat="server">
Header...<br />
<asp:ContentPlaceHolder id="cphFirst" runat="server">
        This is First Content Place Holder (Default)
</asp: ContentPlaceHolder>
<br />
<asp:ContentPlaceHolder ID="cphSecond"
runat="server">
This is Second Content Place Holder (Default)
</asp:ContentPlaceHolder>
<br /> Footer...
</form>
```

ContentPlaceHolder

- To the web application add a WebForm (Default.aspx) a Check (Select Master Page) in New Item Dialog
- Note the attribute "MasterPageFile" in @Page directive of the WebForm.
- Delete the <content tag for the ContentPlaceHolderId="cphSecond".
- Run the WebForm - The output rendered includes the Header, Footer, Content of cphSecond in Master and the content of <content tag for ContentPlaceHolderId="cphFirst" in webform.

- Here we understood the importance of ContentPlaceHolder in Master and Content in WebForm.
- Add a Label in the master page (outside ContentPlaceHolder)

  <asp:Label ID="lblMaster" runat="server" Text="In Master"/>
- Add a Button to WebForm (inside content tag)

- <asp:Button ID="btnDemo" runat="server" onclick="btnDemo_Click" Text="Set Label of Master" />

- Handle the Click event of above button and add to it the code as mentioned below.

```
protected void btnDemo_Click(object sender, EventArgs e)
{
    //Get reference to Label control (lblMaster) in the master page.
    Label lbl = (Label)Master.FindControl("lblMaster");
    lbl.Text = "From WebForm Page...";
}
```

- Run the WebForm and Click on Button to see that the text in master page Label has changed.
- To the class in MainMaster.master.cs add the following property.

```
public Label MasterLabel
{
    get { return lblMaster; }
  }
```

-

- To the Default.aspx add the following

  <%@ MasterType VirtualPath="~/MainMaster.master" %>

  Replace the existing code in btnDemo_Click with the following.

  //To set Text of Label in master page using the public property MasterLabelMaster**.MasterLabel**.Text = "From WebForm"**;**

- //The above line would work only if <%@MasterType Directive is added to current page

Themes and Skins

Theme

- 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, across an entire Web application, or across all Web applications on a server.

- 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.

Skins

- A skin file has the file name extension .skin and contains property settings for individual controls such as ButtonA skin file has the file name extension .skin and contains property settings for individual controls such as Button, LabelA skin file has the file name extension .skin and contains property settings for individual controls such as Button, Label, TextBoxA skin file has the file name extension .skin and contains property settings for individual controls such as Button, Label, TextBox, or Calendarcontrols.

- Control skin settings are like the control markup itself, but contain only the properties you want to set as part of the theme. For example, the following is a control skin for a Button control:

- <asp:button runat="server" BackColor="lightblue" ForeColor="black" />

Skin Files

- You create .skin files in the Theme folder.
- 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:
  - **def*ault skins***
  - **and *named skins.***

default skin & control skin

- A default skin automatically applies to all controls of the same type when a theme is applied to a page.
- A control skin is a default skin if it does not have a **SkinID** attribute.
- For example, if you create a default skin for a CalendarFor example, if you create a default skin for a Calendar control, the control skin applies to all CalendarFor example, if you create a default skin for a Calendar control, the control skin applies to all Calendarcontrols on pages that use the theme. (Default skins are matched exactly by control type, so that a ButtonFor example, if you create a default skin for a Calendar control, the control skin applies to all Calendarcontrols on pages that use the theme. (Default skins are matched exactly by control type, so that a Button control skin applies to all ButtonFor example, if you create a default skin for a Calendar control, the control skin applies to all Calendarcontrols on pages that use the theme. (Default skins are matched exactly by control type, so that

named skin

- A named skin is a control skin with a SkinID property set.
- Named skins do not automatically apply to controls by type.
- Instead, you explicitly apply a named skin to a control by setting the control's SkinID property. Creating named skins allows you to set different skins for different instances of the same control in an application.

Cascading Style Sheets

- A theme can also include a cascading style sheet (.css file).
- When you put a .css file in the theme folder, the style sheet is applied automatically as part of the theme.
- You define a style sheet using the file name extension .css in the theme folder.

Scoping Themes

- You can define themes for a single Web application, or as global themes that can be used by all applications on a Web server.
- After a theme is defined, it can be placed on individual pages using the **Theme** or **StyleSheetTheme** attribute of the @ Page attribute of the @ Page directive, or it can be applied to all pages in an application by setting the <pages> attribute of the @ Page directive, or it can be applied to all pages in an application by setting the <pages> element in the application configuration file. If the <pages> element is defined in the Machine.config file, the theme will apply to all pages in Web applications on the server.

Page Themes

- A page theme is a theme folder with control skins, style sheets, graphics files and other resources created as a subfolder of the \App_Themes folder in your Web site.
- Each theme is a different subfolder of the \App_Themes folder.
- The following example shows a typical page theme, defining two themes named BlueTheme and PinkTheme.
- MyWebSite
    - App_Themes
    - **BlueTheme**
        - **Controls.skin**
        - **BlueTheme.css**
        - **PinkTheme**
        - **Controls.skin**
        - **PinkTheme.css**

How to: Define ASP.NET Page Themes

- To create a page theme
- In Solution Explorer, right-click the name of the Web site for which you want to create a page theme, and then click **Add ASP.NET Folder**.
- Click **Theme**.
- If the **App_Themes** folder does not already exist, Visual Web Developer creates it. Visual Web Developer then creates a new folder for the theme as a child folder of the **App_Themes** folder.

Themes and skins

- Type a name for the new folder.
- The name of this folder is also the name of the page theme. For example, if you create a folder named **App_Themes**\FirstTheme, the name of your theme is FirstTheme.
- Add files to your new folder for control skins, style sheets, and images that make up the theme

How to: Apply ASP.NET Themes

To apply a theme to a Web site

- In the application's Web.config file, set the <pages> element to the name of the theme, either a global theme or a page theme, as shown in the following example:
- <configuration>
- <system.web>
- <pages theme="ThemeName" />
- </system.web>
- </configuration>

To apply a theme to an individual page

- Set the **Theme** or **StyleSheetTheme** attribute of the @ Page directive to the name of the theme to use, as shown in the following example:

- <%@ Page Theme="ThemeName" %>
- <%@ Page StyleSheetTheme="ThemeName" %>

Applying Skins to Controls

- To apply a named skin to a control
- Set the control's SkinID property, as shown in the following example:

- <asp:Calendar runat="server" ID="DatePicker" SkinID="SmallCalendar" />