
ASP.NET
Reputation
CRM
Quality
Goals
Team
Finance
Customers
Costs
Income
Plan
Marketing
Management
Service

Jump2Learn
The Online Learning Place

ASP.NET

CHAPTER 2 : SERVER CONTROL

AUTHORS

Dr. Ashoksinh V. Solanki
M.C.A., Ph.D.
COLLEGE OF APPLIED SCIENCE AND PROFESSIONAL STUDIES, CHIKHLI

Mr. Yatin Solanki
M.C.A., Ph.D.(Pursuing)
DOLAT USHA INSTITUTE OF APPLIED SCIENCES, VALSAD

Mr. Chetan Parmar
M.C.A., NET, SET
NARMADA COLLEGE OF SCIENCE & COMMERCE, ZADESHWAR

Website : www.jump2learn.com | **Email :** info@jump2learn.com | **Instagram :** www.instagram.com/jump2learn
Facebook : www.facebook.com/Jump2Learn | **Whatsapp :** +91-909-999-0960 | **YouTube :** Jump2Learn

❖ AUTOMATIC POSTBACKS

- Windows developers have long been accustomed to a rich event model that lets your code react to mouse movements, key presses, and the minutest control interactions.
- But in ASP.NET, client actions happen on the client side, and server processing takes place on the web server.
- once the page is posted back, ASP.NET can fire other events at the same time
- ASP.NET web controls extend this model with an *automatic postback* feature. With this feature, input controls can fire different events, and your server-side code can respond immediately.
- To use automatic postback, you simply need to set the AutoPostBack property of a web control.

❖ DATA BINDING CONTROLS IN ASP.NET

Data-bound web server controls are controls that can be bound to a data source control to make it easy to display and modify data in your web application. All of these controls provide a variety of properties that you can set to control the appearance of the UI that they generate. For scenarios where you need greater control over a control's UI or how it processes input, some of these controls let you specify the generated HTML directly using templates. A template is a block of HTML markup that includes special variables that you use to specify where and how the bound data is to be displayed. When the control is rendered, the variables are replaced with actual data and the HTML is rendered to the browser.

Data-bound web server controls are composite controls that combine other ASP.NET web controls, such as Label and TextBox controls, into a single layout.

Every ASP.NET web form control inherits the DataBind method from its parent Control class, which gives it an inherent capability to bind data to at least one of its properties. This is known as **simple data binding** or **inline data binding**.

Simple data binding involves attaching any collection (item collection) which implements the IEnumerable interface, or the DataSet and DataTable classes to the DataSource property of the control.

On the other hand, some controls can bind records, lists, or columns of data into their structure through a DataSource control. These controls derive from the BaseDataBoundControl class. This is called **declarative data binding**.

The data source controls help the data-bound controls implement functionalities such as, sorting, paging, and editing data collections.

ASP.NET support following data binding controls

- GridView
- DataList
- DetailsView
- FormView
- ListView
- Repeater Control

GRIDVIEW DATA BINDING CONTROL

The GridView control is the successor to the DataGrid and extends it in a number of ways. With this GridView control, you could display an entire collection of data, easily add sorting and paging, and perform inline editing. In addition to just displaying data, the GridView can be used to edit and delete the displayed data as well.

The GridView comes with a pair of complementary view controls: DetailsView and FormView. By combining these controls, you can easily set up master-detail views using very little code and sometimes no code at all.

You can use the GridView control to do the following:

- Automatically bind to and display data from a data source control.
- Select, sort, page through, edit, and delete data from a data source control.

Additionally, you can customize the appearance and behavior of the GridView control by doing the following:

- Specifying custom columns and styles.
- Utilizing templates to create custom user interface (UI) elements.
- Adding your own code to the functionality of the GridView control by handling events.

DATA BINDING WITH THE GRIDVIEW CONTROL

The GridView control provides you with two options for binding to data:

- Data binding using the DataSourceID property, which allows you to bind the GridView control to a data source control. This is the recommended approach because it allows the GridView control to take advantage of the capabilities of the data source control and provide built-in functionality for sorting, paging, and updating.

- Data binding using the DataSource property, which allows you to bind to various objects, including ADO.NET datasets and data readers. This approach requires you to write code for any additional functionality such as sorting, paging, and updating.

When you bind to a data source using the DataSourceID property, the GridView control supports two-way data binding. In addition to the control displaying returned data, you can enable the control to automatically support update and delete operations on the bound data.

FORMATTING DATA DISPLAY IN THE GRIDVIEW CONTROL

You can specify the layout, color, font, and alignment of the GridView control's rows. You can specify the display of text and data contained in the rows. Additionally, you can specify whether the data rows are displayed as items, alternating items, selected items, or edit-mode items. The GridView control also allows you to specify the format of the columns. For information on formatting the GridView control, see the GridView class overview.

EDITING AND DELETING DATA USING THE GRIDVIEW CONTROL

By default, the GridView control displays data in read-only mode. However, the control also supports an edit mode in which it displays a row that contains editable controls such as TextBox or CheckBox controls. You can also configure the GridView control to display a Delete button that users can click to delete the corresponding record from the data source.

The GridView control can automatically perform editing and deleting operations with its associated data source, which allows you to enable editing behavior without writing code. Alternatively, you can control the process of editing and deleting data programmatically, such as in cases where the GridView control is bound to a read-only data source control.

You can customize the input controls that are used when a row is in edit mode using a template. For more information, see the TemplateField class.

DATALIST BINDING CONTROL

The DataList control displays data items in a repeating list, and optionally supports selecting and editing the items. The content and layout of list items in DataList is defined using templates. At a minimum, every DataList must define an ItemTemplate; however, several optional templates can be used to customize the appearance of the list. The following table describes those templates.

TEMPLATE NAME	DESCRIPTION
ItemTemplate	Defines the content and layout of items within the list. Required.
AlternatingItemTemplate	If defined, determines the content and layout of alternating items. If not defined, ItemTemplate is used.
SeparatorTemplate	If defined, is rendered between items (and alternating items). If not defined, a separator is not rendered.
SelectedItemTemplate	If defined, determines the content and layout of the selected item. If not defined, ItemTemplate (AlternatingItemTemplate) is used.
EditItemTemplate	If defined, determines the content and layout of the item being edited. If not defined, ItemTemplate (AlternatingItemTemplate, SelectedItemTemplate) is used.
HeaderTemplate	If defined, determines the content and layout of the list header. If not defined, the header is not rendered.
FooterTemplate	If defined, determines the content and layout of the list footer. If not defined, the footer is not rendered.

DETAILSVIEW BINDING CONTROL

DetailsView is a data-bound user interface control that renders a single record at a time from its associated data source, optionally providing paging buttons to navigate between records. It is similar to the Form View of an Access database, and is typically used for updating and/or inserting new records. It is often used in a master-details scenario where the selected record of the master control (GridView, for example) determines the DetailsView display record.

The DetailsView control gives you the ability to display, edit, insert, or delete a single record at a time from its associated data source. By default, the DetailsView control displays each field of a record on its own line. The DetailsView control is typically used for updating and inserting new records, often in a master/detail scenario where the selected record of the master control determines the record to display in the DetailsView control. The DetailsView control displays only a single data record at a time, even if its data source exposes multiple records.

The DetailsView control relies on the capabilities of the data source control to perform tasks such as updating, inserting, and deleting records. The DetailsView control does not support sorting.

The DetailsView control can automatically page over the data in its associated data source, provided that the data is represented by an object supporting the ICollection interface or that the underlying data source supports paging. The DetailsView control provides the user interface (UI) for navigating between data records. To enable paging behavior, set the AllowPaging property to true.

You select a particular record from the associated data source by paging to that record. The record displayed by the DetailsView control is the current selected record.

THE REPEATER CONTROL

The Repeater control displays data items in a repeating list. Similar to DataList, the content and layout of list items in Repeater is defined using templates. At a minimum, every Repeater must define an ItemTemplate; however, the following optional templates may be used to customize the appearance of the list.

Repeater Control is used to display repeated list of items that are bound to the control and it's same as gridview and datagridview. Repeater control is lightweight and faster to display data when compared with gridview and datagrid. By using this control we can display data in custom format but it's not possible in gridview or datagridview and it doesn't support for paging and sorting.

The Repeater control works by looping through the records in your data source and then repeating the rendering of its templates called item template. Repeater control contains different types of template fields those are

1) itemTemplate 2) AlternatingitemTemplate 3) HeaderTemplate 4) FooterTemplate

5) SeperatorTemplate

ItemTemplate: ItemTemplate defines how the each item is rendered from data source collection.

AlternatingItemTemplate: AlternatingItemTemplates is used to change the background color and styles of AlternatingItems in DataSource collection

HeaderTemplate: HeaderTemplate is used to display Header text for DataSource collection and apply different styles for header text.

FooterTemplate: FooterTemplate is used to display footer element for DataSource collection

SeparatorTemplate: SeparatorTemplate will determine separator element which separates each Item in Item collection. Usually, SeparateTemplate will be
 html element or <hr> html element.

Unlike DataList, Repeater has no built-in layout or styles. You must explicitly declare all HTML layout, formatting, and style tags within the templates of the control. For example, to create a list within an HTML table, you might declare the <table> tag in the HeaderTemplate, a table row (<tr> tags, <td> tags, and data-bound items) in the ItemTemplate, and the </table> tag in the FooterTemplate.

Example of Repeater Control

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb" Inherits="_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:Repeater ID="Repeater1" runat="server" DataSourceID="SqlDataSource1">
<HeaderTemplate>

<table cellspacing="0" rules="all" border="1">
<tr>
<th scope="col" style="width: 80px">
        Roll No
</th>
<th scope="col" style="width: 120px">
        Student
</th>

</tr>

</HeaderTemplate>

<ItemTemplate>
<tr>
<td bgcolor="#CCFFCC">
<asp:Label runat="server" ID="Label1"
        text='<%# Eval("rollno") %>' />
```

```
</td>

<td bgcolor="#CCFFCC">
<asp:Label runat="server" ID="Label2"
                text='<%# Eval("sname") %>' />
</td>
</tr>
</ItemTemplate>

<AlternatingItemTemplate>
<tr>
<td >
<asp:Label runat="server" ID="Label3"
              text='<%# Eval("rollno") %>' />
</td>
<td >
<asp:Label runat="server" ID="Label4"
              text='<%# Eval("sname") %>' />
</td>
</tr>
</AlternatingItemTemplate>

<SeparatorTemplate >
<tr>
<td >
          =========
</td>
</SeparatorTemplate>
<FooterTemplate>
</table>
</FooterTemplate>
</asp:Repeater>
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
       ConnectionString="<%$ ConnectionStrings:ConnectionString %>"
       SelectCommand="SELECT * FROM [stud]"></asp:SqlDataSource>

</div>
</form>
</body>
</html>
```


FORMVIEW

FormView is a data-bound user interface control that renders a single record at a time from its associated data source, optionally providing paging buttons to navigate between records. It is similar to the DetailsView control, except that it requires the user to define the rendering of each item using templates, instead of using data control fields.

The main difference between DetailsView and FormView is that DetailsView has a built-in tabular rendering, whereas FormView requires a user-defined template for its rendering. The FormView and DetailsView object model are very similar otherwise. The following example demonstrates a FormView control bound to an ObjectDataSource. The ItemTemplate property of FormView contains data-bound Image, Label and HyperLink controls.

The FormView control is typically used for updating and inserting new records. It is often used in master/detail scenarios where the selected record of the master control determines the record to display in the FormView control. For more information and an example, see Modifying Data Using a FormView Web Server Control.
The FormView control relies on the capabilities of the data source control to perform tasks such as updating, inserting, and deleting records. The FormView control displays only a single data record at a time, even if its data source exposes multiple records.

❖ WEB SERVER CONTROLS

❖ HTML SERVER CONTROLS (BASIC HTML SERVER CONTROL)

❖ VALIDATION CONTROLS

ASP.NET INCLUDES SIX VALIDATION CONTROLS.

- <asp:RequiredFieldValidator>
- <asp:RangeValidator>
- <asp:CompareValidator>
- <asp:RegularExpressionValidator>
- <asp:CustomValidator>
- <asp:ValidationSummary>

- you can use more than one validator for the same control.
- you can't validate RadioButton or CheckBox controls
- you can validate the TextBox(the most common choice) and other controls such as ListBox, DropDownList, RadioButtonList,HtmlInputText, HtmlTextArea, and HtmlSelect.

THE VALIDATION PROCESS

- You can use the validation controls to verify a page automatically when the user submits it or to verify it manually in your code.
- Every button has a CausesValidation property, which can be set to true or false.
  - If false, ASP.NET will ignore the validation controls, the page will be posted back, and your event-handling code will run normally.
  - If true (the default), ASP.NET will automatically validate the page when the user clicks the button.

THE BASEVALIDATOR CLASS

- The validation control classes are found in the System.Web.UI.WebControls namespace and inherit from the BaseValidator class.
- This class defines the basic functionality for a validation control.
- BaseValidator Members
  - ControlToValidate
  - EnableClientScript
  - Display
  - Enabled
  - ErrorMessage
  - ValidationGroup

THE REQUIREDFIELDVALIDATOR CONTROL

- It's work is to ensure that the associated control is not empty
- E.g.
  - <asp:TextBox runat="server" ID="Name" />

<asp:RequiredFieldValidator runat="server"

ControlToValidate="Name" ErrorMessage="Name is required" Display="dynamic">

*

</asp:RequiredFieldValidator>

THE RANGEVALIDATOR CONTROL

- The RangeValidator control verifies that an input value falls within a predetermined range.
- It has three specific properties: Type, MinimumValue, and MaximumValue.
  - *Type* defines the type of the data that will be typed into the input control and validated.
  - The available values are Currency, Date, Double, Integer, and String.
- E.g.

```
<asp:TextBox runat="server" ID="DayOff" />
 <asp:RangeValidator runat="server" Display="dynamic"
    ControlToValidate="DayOff" Type="Date"
    ErrorMessage="Day Off is not within the valid interval"
    MinimumValue="08/05/2002" MaximumValue="08/20/2002">
   *
</asp:RangeValidator>
```

THE COMPAREVALIDATOR CONTROL

- The CompareValidator control compares a value in one control with a fixed value or, more commonly, a value in another control.
- For example, this allows you to check that two text boxes have the same data or not.
- E.g.-1

```
<asp:TextBox runat="server" ID="Age" />
    <asp:CompareValidator runat="server" Display="dynamic"
    ControlToValidate="Age" ValueToCompare="18"
        ErrorMessage="You must be at least 18 years old"
  Type="Integer" Operator="GreaterThanEqual">  *
   </asp:CompareValidator>
```

- E.g.-2

```
<asp:TextBox runat="server" TextMode="Password" ID="Password" />

<asp:TextBox runat="server" TextMode="Password“

ID="Password2" />

<asp:CompareValidator runat="server“ ControlToValidate="Password2"
    ControlToCompare= "Password“ ErrorMessage="The passwords don't match"

 Type="String" Display="dynamic">

  <img src="imgError.gif" border="0" alt="The passwords don't match">

</asp:CompareValidator>
```

THE REGULAREXPRESSIONVALIDATOR CONTROL

- It allows you to validate text by matching against a pattern defined in a regular expression.
- You simply need to set the regular expression in the ValidationExpression property.
- For example, the following control checks that the text input in the text box is a valid e-mail address:

```
<asp:TextBox runat="server" ID="Email" />

<asp:RegularExpressionValidator runat="server"

 ControlToValidate="Email" ValidationExpression= ".*@.{2,}\..{2,}“ ErrorMessage=
    "E-mail is not in a valid format" Display="dynamic"> *

</asp:RegularExpressionValidator>
```

- Metacharacters for Matching Single Characters
- Metacharacters for Matching Types of Characters
- Quantifiers
- E-mail address : * \S+@\S+\.\S+

THE CUSTOM VALIDATOR CONTROL

- if you need more advanced or customized validation, then the CustomValidator control is what you need.
- TheCustomValidator allows you to execute your custom client-side and server-side validation routines.
- It takes two parameters: a reference to the validator and a custom argument object.
- This object provides a Value property that contains the current value of the associated input control (the value you have to validate) and an IsValid property through which you specify whether the input value is valid.

THE VALIDATIONSUMMARY CONTROL

- The ValidationSummary control doesn't perform any validation. Instead, it allows you to show a summary of all the errors in the page.
- This summary displays the ErrorMessage value of each failed validator.
- The summary can be shown in a client-side JavaScript message box (if theShowMessageBox property is true) or on the page (if the ShowSummary property is true).
- Youcan set both ShowMessageBox and ShowSummary to true to show both types of summaries.
- E.g.

```
<asp:ValidationSummary runat="server" ID="ValidationSum"

ShowSummary="true" DisplayMode="BulletList"    HeaderText="<b>Please review
  the following errors:</b>" />
```

❖ NAVIGATION CONTROLS

Navigation controls are very important for websites.Navigation controls are basically used to navigate the user through webpage .It is more helpful for making the navigation of pages easier .There are three controls in ASP.NET ,Which are used for Navigation on the webpage.

1. TreeView control
2. Menu Control

TREEVIEW CONTROL

The TreeView control is used to display hierarchical data, such as a table of contents or file directory, in a tree structure.

A Tree View control displays a hierarchical list of items using lines to connect related items in a hierarchy. Each item consists of a label and an optional bitmap. Windows Explorer uses a Tree View control to display directories. You can use the Tree View control in any situation in which you need to display hierarchical data.

The TreeView control is used for logically displaying the data in a hierarchical structure. We can use this navigation control for displaying the files and folders on the webpage.

TREEVIEW NODE TYPES

The TreeView control is made up of one or more nodes. Each entry in the tree is called a node. The following table describes the three different node types.

TreeView control node types

NODE TYPE	DESCRIPTION
Root	A node that has no parent node and one or more child nodes.
Parent	A node that has a parent node and one or more child nodes.
Leaf	A node that has no child nodes.

Although a typical tree has only one root node, the TreeView control allows you to add multiple root nodes to your tree structure. This is useful when you want to display item listings without displaying a single main root node, as in a list of product categories.

Each node has a Text property and a Value property. The value of the Text property is displayed in the TreeView control, while the Value property is used to store any additional data about the node, such as data passed to the postback event that is associated with the node.

MENU Control

The ASP.NET menu control allows you to develop both statically and dynamically displayed menus for your ASP.NET Web pages.

The Menu control has two modes of display: static and dynamic. Static display means that the Menu control is fully expanded all the time. The entire structure is visible, and a user can click on any part. In a dynamically displayed menu, only the portions you specify are static, while their child menu items are displayed when the user holds the mouse pointer over the parent node.

The Menu control is used to create a menu of hierarchical data that can be used to navigate through the pages. The Menu control conceptually contains two types of items. First is StaticMenu that is always displayed on the page, Second is DynamicMenu that appears when opens the parent item.

Its properties like BackColor, ForeColor, BorderColor, BorderStyle, BorderWidth, Height etc. are implemented through style properites of <table, tr, td/> tag.

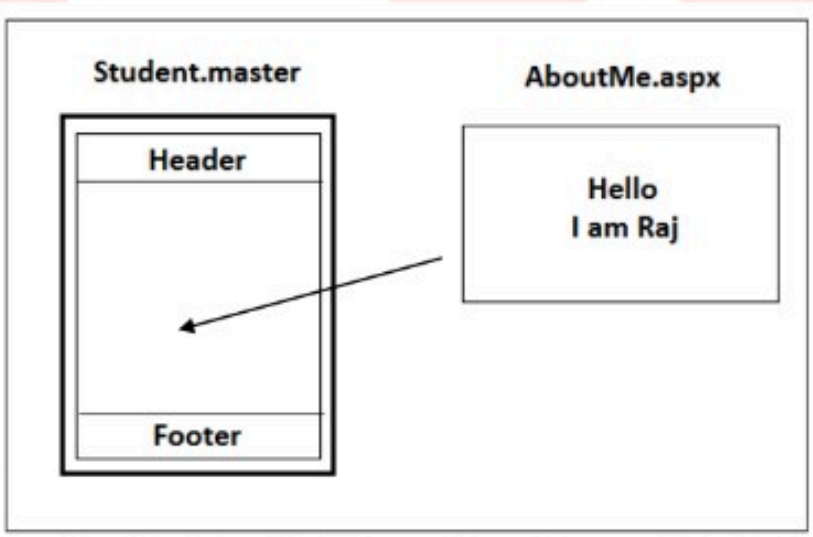
Following are some important properties that are very useful.

PROPERTIES OF MENU CONTROL	
DataSourceID	Indicates the data source to be used (You can use .sitemap file as datasource).
Text	Indicates the text to display in the menu.
Tooltip	Indicates the tooltip of the menu item when you mouse over.
Value	Indicates the nondisplayed value (usually unique id to use in server side events)
NavigateUrl	Indicates the target location to send the user when menu item is clicked. If not set you can handle MenuItemClick event to decide what to do.
Target	If NavigationUrl property is set, it indicates where to open the target location (in new window or same window).
Selectable	true/false. If false, this item can't be selected. Usually in case of this item has some child.
ImageUrl	Indicates the image that appears next to the menu item.
ImageToolTip	Indicates the tooltip text to display for image next to the item.
PopOutImageUrl	Inidcates the image that is displayed right to the menu item when it has some subitems.
Target	If NavigationUrl property is set, it indicates where to open the target location (in new window or same window).
STYLES OF MENU CONTROL	
StaticMenuStyle	Sets the style of the parent box in which all menu items appears.
DynamicMenuStyle	Sets the style of the parent box in which dynamic menu items appears.
StaticMenuItemStyle	Sets the style of the individual static menu items.
DynamicMenuItemStyle	Sets the style of the individual dynamic menu items.
StaticSelectedStyle	Sets the style of the selected static items.
DynamicSelectedStyle	Sets the style of the selecdted dynamic items.
StaticHoverStyle	Sets the mouse hovering style of the static items.
DynamicHoverStyle	Sets the mouse hovering style of the dynamic items (subitems).

- **LOGIN CONTROL**
- **MASTER PAGE**

- MasterPages allows you to create a consistent look and behavior for all the pages (or group of pages) in your web application.

- MasterPage means not HomePage.
- A MasterPage provides a template for other pages, with shared layout and functionality.
- If you want to disply standard header, footer and menu in each page in your website, you can create the standard header, footer and menu in a MasterPage.
- Microsoft has used a concept from PowerPoint application with Master pages. In powerpoint you can create a master slide that acts as a template for all other slides. The background and common text on the master slide is automatically used in other slides.
- Likewise the Master Page is used to create the look and feel of ASP.NET web application.
- Master Pages help us build consistent and maintainable user interface.
- Remember that It is not related with color-combination concept; it is related with **Layout** – like frameset of HTML.
- They allow you to layout a page and use it over and over.
- The MasterPage defines placeholders for the content, which can be overridden by content pages. The output result is a combination of the master page and content page.
- The content page contains the content you want to display.



- When user request the content page, ASP.NET merges the pages to produce output that combines the layout of the Master Page with the content of the content page.

Characteristics:-

- Extension of MasterPage is '.master' it also has a code behind file.
- MasterPage can not be run directly.

- MasterPage can be nested.
- MasterPage allows centralizing the common functionality of the pages so that we can make updates at one place only.
- Website can contain many Master Pages and you can create different set of content pages for each Master Page.
- A MasterPage has 2 parts:
    1. Content that appears on each page that inherits the master page.
    2. Regions that can be customized by the pages inheriting the master page.
- A Master Page need to specify both the parts common to all pages and the parts that are customizable.
- Use ContentPlaceHolder control to specify a region that can be customize.
- Master Page should contain at least one CntentPlaceHolder.

ContentPlaceHolder:-

- When a new Master Page is initially added that time you will see ContentPlaceHolder1 on the webpage.
- ContentPlaceHolders are used to allocate places on the MasterPage that will be changed on the content pages.
- ContentPlaceHolder is a control in MasterPage. The content from the content page appears in the area made by ContentPlaceHolder.
- There are two ContentPlaceHolder on any MasterPage.
    1. In the Head section.
    2. In the between <form> </form>
- In the head you specify certain Meta fields such as the page description and keywords. These vary from page to page.
- In the form section the actual webpage contents will presents.
- You can have multiple ContentPlaceHolders on the same page.

Content Pages:-

- The content pages contain the content you want to display.
- For example – Home.aspx, AboutUs.aspx, ContactUs.aspx, etc... are content pages.
- You define the content for the master page's placeholder controls by creating individual content pages, which are ASP.NET pages that are bound to a specific Master Page.
- When users request the content pages, they merge with the MasterPage to produce output that combines the layout of the MasterPage with the contents from the content page.

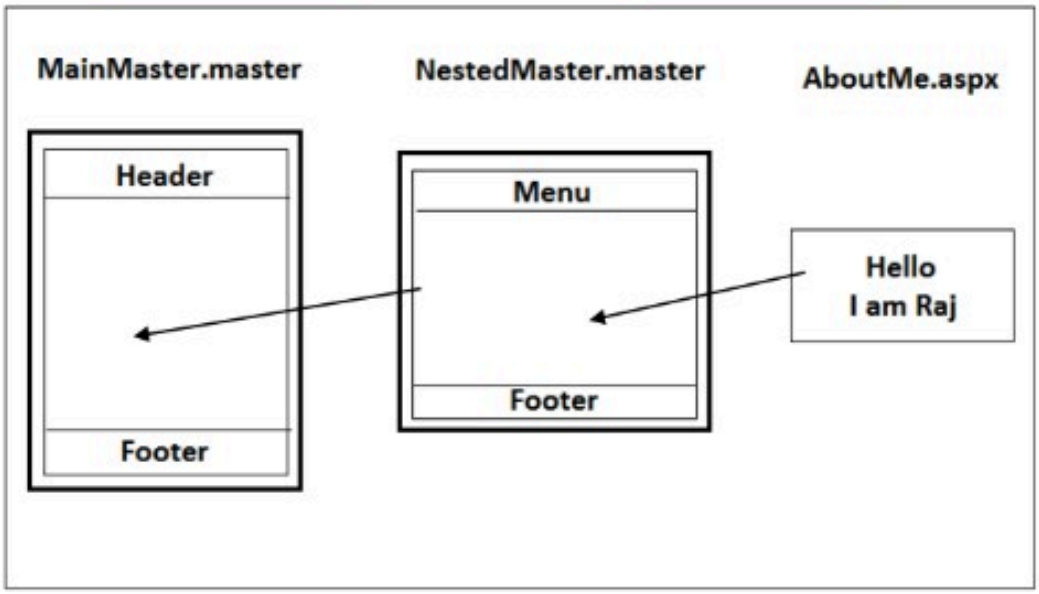
Run-time Behavior of Master Pages:-

- Users request a page by typing the URL of the content page.
- ASP.NET fetches the page.

- When the page is fetched, the @page directive is read. If the directive references a Master Page, the Master Page is read as well. If this is the first time the pages have been requested, both pages are compiled.
- ASP.NET merges the content into the ContentPlaceHolders on the MasterPage.
- The MasterPage with the updated content is merged into the control tree of the content page.
- ASP.NET renders result to the browser.

Nested MasterPage

- When one Master Page references another as its master, it is said to be a Nested Master Page.
- For example- you can create one Master Page for the entire website and then nest Master Pages below the site Master Page for individual sectons.



- A number of nested masters can be componentized into a single master.
- There is no architectural limitations to the number of child master that can be created.
- The depth of nesting also does not impact on performance significantly.
- The advantages of this kind of structuring is that a number of child masters can be created to be subordinated to the overall look and feel of the site defined by the parent master, while the child master give the child pages some uniqueness. While a parent master defines the overall layout of the pages header, body and footer, the child master expands the body for a group of pages.
- Just like the parent master page child masters have the extension '.master' too.
- It contains all the controls that are mapped to ContentPlaceHolders on the parent master page.

- Child masters also have ContentPlaceHolders of their own to display contents of its child pages.

❖ THEMES & CSS

- A Master Page enables user to share content across multiple pages in a website, while a Theme enables you to control the appearance of the content.
- Like the concept of changing the Theme in "Orkut".
- 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 application on a server.
- The concept is when one user is visiting site and seeing it, another user can be view the same site, but gets a completely different experience.
- Themes can be thought of as a container where you store your style sheets, images, skin files, etc.
- For theme here is am App_Theme folder, we can add this folder by right clicking on web project add new folder.
- Themes are applied on the server: so we can change it runtime.
- A theme folder can contain a variety of different types of files:
    - Skins
    - Cascading Style Sheets (CSS)
    - Images and other resources.
- Themes are control-based, not HTML-based: One of the biggest advantages of themes is that unlike CSS, it allows you to define and reuse almost any control property.
- Themes can be applied through configuration files.
- Only one theme can be applied to each page.

SKIN FILE:-

- A skin generally contains visual properties (for look) for one or more kinds of ASP.NET controls (server control)
- A skin file has the file name extension '.skin'
- It contains property settings for individual controls such as Button, Label, TextBox, Calendar, etc. server side controls.
- You can define skins in a separate file for each control or define all the skins for a theme in a single file.

DEFAULT SKIN:-

- A default skin automatically applies to all controls of the same type when a theme is applied to a page.
- The SkinId is not defined.

- Only one default control skin per control type is allowed in the same theme.
- For example, if you create a default skin for a TextBox control, the control skin applies to all TextBox controls on pages that use the theme.

NAMED SKIN:-

- It is a control skin with a **SkinID** property set.
- Named skin do not automatically apply to controls by type.
- In Named skin, user can decide when you want to apply the Skin.
- The SkinID should be uniquely defined because duplicate SkinID's per control type are not allowed in the same theme.

CSS FILE:-

- A Cascading Style Sheet (CSS) may be added to a theme by placing it under the named theme subdirectory.
- Extension of the CSS file is '.css'.
- When you put a '.css' file in the theme directory, the style sheet is applied automatically as part of the theme.
- The CSS will be applied to all pages with that theme applied.

POINTS TO REMEMBER:-

- Each and every control has property **Enable Theme**, user can set either true or false.
- It indicates whether the control can be themed or not.
- If the user set it as a false then the effect never comes. By default it's True.