UNIT IV WEB BASED APPLICATION DEVELOPMENT ON .NET

Programming Web Applications with Web Forms using ASP.NET - ASP.NET controls - Understand Data Binding and various Data Sources in ASP.NET - Understand the creation of Master Pages and themes - Configuration of web applications using IIS configurations - State management in ASP.NET- Programming Web Services.

ASP.NET

ASP.NET is a web application framework developed and marketed by Microsoft to allow programmers to build dynamic web sites.ASP.Net is a part of Microsoft .Net platform. ASP.Net applications could be written in either of the following languages:

- C#
- Visual Basic .Net
- Jscript
- J#

ASP.Net Environment Setup:

- The key development tool for building ASP.Net applications and front ends is Visual Studio.
- Visual Studio is an integrated development environment for writing, compiling and debugging the code. It provides a complete set of development tools for building ASP.Net web applications, web services, desktop applications and mobile applications.
- When you start a new web site, ASP.NET provides the starting folders and files for the site, including two files for the first web form of the site.
- The file named Default.aspx contains the HTML and asp code that defines the form, and the file named Default.aspx.cs (for C# coding) or the file named Default.aspx.vb (for vb coding) contains the code in the language you have chosen and this code is responsible for the form's works.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="WebApplication1._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>SAMPLE APPLICATION</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

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

ASP.NET CONTROLS

Web Server Controls

Web controls fall into the following categories

a.1. Input
a.2. Display
a.3. Action
a.4. Selection
a.5. Databound
a.6. Validation

1. Input Controls

Textbox is the input control available in ASP.NET.Text box controls are typically used to accept input from the user. A text box control can accept one or more lines to text depending upon the setting of the TextMode attribute.

Basic syntax for text box:

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

Common Properties of the Text Box are given below:

Property	Description
TextMode	Specifies the type of text box. SingleLine creates a standard text box, MultiLIne creates a text box that accepts more than one line of text and the Password causes the characters that are entered to be masked. The default is SingleLine.
Text	The text content of the text box
MaxLength	The maximum number of characters that can be entered into the text box.
Wrap	It determines whether or not text wraps automatically for multi-line text box; default is true.
ReadOnly	Determines whether the user can change the text in the box; default is false, i.e., the user can change the text.

The TextBox control can then be accessed programmatically in .cs file with a code fragment like:

```
Textbox1.Text = "Hello ASP.NET"
```

2.Display Controls

Display controls simply render text or images to the browser. Following are the Display Controls which are widely used.

- Label
- Image

Label Control :

Use the Label control to display text in a set location on the page. Unlike static text, you can customize the displayed text through the Text property.

Syntax for Label is as follows

```
<asp:label id="Label1" Text = "Welcome" runat="Server" />
```

Image Control:

The image control is used for displaying images on the web page, or some alternative text, if the image is not available.

Basic syntax for an image control:

```
<asp:Image ID="Image1" imageurl="aspnet.gif" runat="server">
```

It has the following important properties:

Property	Description
AlternateText	Alternate text to be displayed
ImageAlign	Alignment options for the control
ImageUrl	Path of the image to be displayed by the control

3. Action Controls

Action controls allow users to perform some action on that page, such as navigating to a different URL, submitting a form, resetting a form's values, or executing a client script. The following Table lists the action controls.

- Button

- Hyperlink

Button Control:

ASP .Net provides three types of button controls: buttons, link buttons and image buttons. As the names suggest a button displays text within a rectangular area, a link button displays text that looks like a hyperlink. And an Image Button displays an image.

When a user clicks a button control, two events are raised Click and Command.

Basic syntax for button controls:

```
<asp:Button ID="Button1" runat="server"   onclick="Button1_Click"
Text="Click" />
```

Common Properties of the Button control:

Property	Description
Text	The text displayed by the button. This is for button and link button controls only.
ImageUrl	For image button control only. The image to be displayed for the button.
AlternateText	For image button control only. The text to be displayed if the browser can't display the image.
CausesValidation	Determines whether page validation occurs when a user clicks the button. The default is true.
PostBackUrl	The URL of the page that should be requested when the user clicks the button.

HyperLink Control:

Displays a hyperlink text that navigates from one page to another when clicked.

Syntax for Hyperlink :

```
<asp:HyperLink ID="HyperLink1" runat="server" NavigateUrl="Default.aspx"
Text="Clicke here"> </asp:HyperLink>
```

It has the following important properties:

Property	Description
ImageUrl	Path of the image to be displayed by the control
NavigateUrl	Target link URL
Text	The text to be displayed as the link
Target	The window or frame which will load the linked page.

4. Selection Controls

Selection controls allow the user to select one or more values from a list. They include both the CheckBox and RadioButton controls, which are designed to work in a group. The RadioButton control allows you to select only one option out of the group, whereas the CheckBox control allows you to select zero or more options

- Checkbox
- Radiobutton
- Listbox
- Dropdownlist
- Radiobuttonlist
- Checkbox list

Check Boxes and Radio Buttons:

A check box displays a single option that the user can either check or uncheck and radio buttons present a group of options from which the user can select just one option.

To create a group of radio buttons, you specify the same name for the GroupName attribute of each radio button in the group. If more than one group is required in a single form specify a different group name for each group.

If you want a check box or radio button to be selected when it's initially displayed, set its Checked attribute to true. If the Checked attribute is set for more than one radio button in a group, then only the last one will be selected.

Basic syntax for check box:

```
<asp:CheckBox ID= "chkoption" runat= "Server">  </asp:CheckBox>
```

Basic syntax for radio button:

```
<asp:RadioButton ID= "rdboption" runat= "Server"> </asp: RadioButton>
```

Common Properties of the Check Boxes and Radio Buttons:

Property	Description
Text	The text displayed next to the check box or radio button.
Checked	Specifies whether it is selected or not, default is false.
GroupName	Name of the group the control belongs to.

The checkbox can be accessed programmatically in cs file as follows

In a default.aspx file put a Label control and two CheckBox control. When someone check the CheckBox1 control the Label control show the message that he checked the CheckBox1. The second CheckBox is automatically checked and when he checked the first CheckBox.

```
if (CheckBox1.Checked == true)
        {
            Label1.Text = "Member";
            CheckBox2.Checked = true;
        }
        else
        {
            Label1.Text = "Not a Member";
            CheckBox2.Checked = false;

        }
```

Similarly radiobutton can also be accessed as

MyRadioButton1.Checked = False

Listbox and DropdownList

List boxes and drop-down list contain one or more list items. These lists could be loaded either by code or by the ListItem Collection Editor.

Basic syntax for list box control:

```
<asp:ListBox
        ID="ListBox1"
```

```
        runat="server"
        AutoPostBack="True"
        OnSelectedIndexChanged="ListBox1_SelectedIndexChanged">

    </asp:ListBox>
```

Basic syntax for a drop-down list control:

```
<asp:DropDownList
        ID="DropDownList1"
        runat="server"
        AutoPostBack="True"
        OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged">

</asp:DropDownList>
```

Common Properties of List box and Drop-down Lists:

Property	Description
Items	The collection of ListItem objects that represents the items in the control. This property returns an object of type ListItemCollection.
Rows	Specifies the number of items displayed in the box. If actual list contains more rows than displayed then a scroll bar is added.
SelectedIndex	The index of the currently selected item. If more than one item is selected, then the index of the first selected item. If no item is selected, the value of this property is -1.
SelectedValue	The value of the currently selected item. If more than one item is selected, then the value of the first selected item. If no item is selected, the value of this property is an empty string("").
SelectionMode	Indicates whether a list box allows single selections or multiple selections.

It is important to notes that:

- To work with the items in a drop-down list or list box, you use the Items property of the control. This property returns a ListItemCollection object which contains all the items of the list.
- The SelectedIndexChanged event is raised when the user selects a different item from a drop-down list or list box.

Radio Button list and Check Box list

A radio button list presents a list of mutually exclusive options. A check box list presents a list of independent options. These controls contain a collection of ListItem objects that could be referred to through the Items property of the control.

Basic syntax for radio button list

```
<asp:RadioButtonList
        ID="RadioButtonList1"
        runat="server"
        AutoPostBack="True"
        OnSelectedIndexChanged="RadioButtonList1_SelectedIndexChanged">
 </asp:RadioButtonList>
```

Basic syntax for check box list:

```
<asp:CheckBoxList
        ID="CheckBoxList1"
        runat="server"
        AutoPostBack="True"
        OnSelectedIndexChanged="CheckBoxList1_SelectedIndexChanged">
</asp:CheckBoxList>
```

Common Properties of Check Box and Radio Button Lists:

Property	Description
RepeatLayout	This attribute specifies whether the table tags or the normal html flow to use while formatting the list when it is rendered. The default is Table
RepeatDirection	It specifies the direction in which the controls to be repeated. The values available are Horizontal and Vertical. Default is Vertical
RepeatColumns	It specifies the number of columns to use when repeating the controls; default is 0.

5.Databound Controls

- List controls
- GridView
- FormView

List Controls

List controls include the ListBox, DropDownList, CheckBoxList, RadioButtonList, and BulletedList. Each of these controls can be data bound to a data source. They use one field in the data source as the display text and can optionally use a second field as the value of an item. Items can also be added statically at design-time, and you can mix static items and dynamic items added from a data source.

To data bind a list control, add a data source control to the page. Specify a SELECT command for the data source control and then set the DataSourceID property of the list control to the ID of the data source control. Use theDataTextField and DataValueField properties to define the display text and the value for the control

GridView

The GridView control allows for tabular data display and editing using a declarative approach and is the successor to the DataGrid control. The following features are available in the GridView control.

- Binding to data source controls, such as SqlDataSource.
- Built-in sorting capabilities.
- Built-in updating and deleting capabilities.
- Built-in paging capabilities.
- Built-in row selection capabilities.
- Programmatic access to the GridView object model to dynamically set properties, handle events, and so on.
- Multiple key fields.
- Multiple data fields for the hyperlink columns.
- Customizable appearance through themes and styles.

FormView

The FormView control is used to display a single record from a data source. It is similar to the DetailsView control, except it displays user-defined templates instead of row fields. Creating your own templates gives you greater flexibility in controlling how the data is displayed. The FormView control supports the following features:

- Binding to data source controls, such as SqlDataSource and ObjectDataSource.
- Built-in inserting capabilities.
- Built-in updating and deleting capabilities.
- Built-in paging capabilities.
- Programmatic access to the FormView object model to dynamically set properties, handle events, and so on.

- Customizable appearance through user-defined templates, themes, and styles.

6. Validation Controls

ASP.NET removes the hassle of duplicating validation code, a common problem of performing data validation using classic ASP, by neatly encapsulating the standard validations into server controls. You can declaratively relate the validation control to the control whose value needs to be validated, using the ControlToValidate attribute. You can also attach multiple validation controls to a single control. The ASP.NET validation server controls provide server-side validation for all browsers and supply client-side validation via JavaScript for browsers that support JavasScript and DHTML. You can also write your own custom client and/or server-side validation functions, as you'll see in the code example for this section.

One feature that most web programmers would like to have is a summary of the validation errors for the values entered into a page's controls. The ValidationSummary control provides this much-desired feature.

- The RequiredFieldValidator
- The RangeValidator
- The CompareValidator:
- The RegularExpressionValidator

Control	Purpose
CompareValidator	Compares the input in the attached control with a constant value or the property value of another control.
RangeValidator	Checks if the value is between specified upper and lower limits.
RegularExpressionValidator	Checks if the input matches a pattern defined by a regular expression.
RequiredFieldValidator	Ensures that the user can't skip the required

Control	Purpose
	value.

The RequiredFieldValidator:

The RequiredFieldValidator control ensures that the required field is not empty. It is generally tied to a text box to force input into the text box.

Example

```
<asp:TextBox id="Text1"
            Text="Enter a value"
            runat="server"/>

      <asp:RequiredFieldValidator
            id="RequiredFieldValidator1"
            InitialValue="Enter a value"
            ControlToValidate="Text1"
            ErrorMessage="Required field!"
            runat="server" />
```

The RangeValidator:

The RangeValidator control verifies that the input value falls within a predetermined range.The range can be within a pair of numbers,characters or dates.A rangevalidator will display its Errormessage if the value in the control is not between the values specified by the minimumvalue and Maximimvalue attributes.

It has three specific properties:

Properties	Description
Type	it defines the type of the data; the available values are: Currency, Date, Double, Integer and String
MinimumValue	it specifies the minimum value of the range
MaximumValue	it specifies the maximum value of the range

Example

```
      <asp:TextBox id="TextBox3"
            runat="server"/>

      <asp:RangeValidator id="Range1"
            ControlToValidate="TextBox1"
            MinimumValue="1"
            MaximumValue="10"
            Type="Integer"
            EnableClientScript="false"
            Text="The value must be from 1 to 10!"
            runat="server"/>
```

The CompareValidator:

The CompareValidator control compares a value in one control with a fixed value, or, a value in another control.It has the following specific properties:

Properties	Description
Type	it specifies the data type
ControlToCompare	it specifies the value of the input control to compare with
ValueToCompare	it specifies the constant value to compare with
Operator	it specifies the comparison operator, the available values are: Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual and DataTypeCheck

Example :

A classic use of this technique might be to ask a user to enter his password twice and then validate that both entries are identical.

The common scenario is that the user is asked to enter a new password.For security reasons,when the user enter the password,the text is disguised with asterisk or dots.The password entered needs to be validated.This is usually done by reentering the password and then validating across the first password entered.The comparevalidator is the perfect control for doing this job.

```
<asp:TextBox id="TextBox3" runat="server"/>
<asp:TextBox id="TextBox4" runat="server"/>

<asp:CompareValidator
      id="Compare1"
      ControlToValidate="TextBox1"
      ControlToCompare="TextBox2"
      Type="String"
      ErrorMessage=”Passwords do not match”
      Operator=”Equal”
      Text="*"
      runat="server"/>
```

The RegularExpressionValidator

The RegularExpressionValidator allows validating the input text by matching against a pattern against a regular expression. For example to ensure Zipcode is 6 digits,an email address is of the form name@place.com and credit card matches the right format etc.For all

these regular expression validator is used.The regular expression is set in the ValidationExpression property.

The syntax for the control:

```
<asp:RegularExpressionValidator ID="string"
        runat="server"
        ErrorMessage="string"
        ValidationExpression="string"
        ValidationGroup="string">
</asp:RegularExpressionValidator>
```

To validate a server control's input using a RegularExpressionValidator

1. Add a RegularExpressionValidator control to your page.
2. Set the ControlToValidate property to indicate which control to validate.
3. Set the ValidationExpression property to an appropriate regular expression.
4. Set the ErrorMessage property to define the message to display if the validation fails.

The following example shows a **RegularExpressionValidator** control used to validate a name field.

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

    <asp:RegularExpressionValidator
        ID="regexpName"
        runat="server"
        ErrorMessage="This expression does not validate."
        ControlToValidate="txtName"
        ValidationExpression="^[a-zA-Z'.\s]{1,40}$" />
```

The regular expression used in the preceding code example constrains an input name field to alphabetic characters (lowercase and uppercase), space characters, the single quotation mark (or apostrophe) for names such as O'Dell, and the period or dot character. In addition, the field length is constrained to 40 characters.

HANDLING CONTROL EVENTS

One of the most convenient aspects of the new ASP.NET Web Forms model is that it brings event-driven programming,.

Moreover, most server controls expose one or more events for which you can write handlers. The below table shows a list of common events and the controls that support them. These events and controls are in addition to the standard events, such as Init, Load, PreRender, and UnLoad, that are inherited from the base Controlclass.

For example, the Button server control exposes the Click and Command events. These events are both raised when the button is clicked, but while the Click event is usually used simply to handle the event for a single button, the Command event can be used to handle clicking on several buttons (so long as the buttons' CommandName property is set). The CommandName

property, along with an optional CommandArgument property, become properties of the CommandEventArgs object, which is passed as a parameter of the Command event handler. You can then examine the CommandName and CommandArgument properties within the event handler code to determine what action(s) to take.

Event	Event type	Description	Controls
OnAdCreated	Change	Raised after creation of the control and immediately before the page is rendered. If an Advertisement file is provided, OnAdCreated is raised after an ad has been selected from the file. Passes an AdCreatedEventArgs argument.	AdRotator
OnClick	Action	Raised when the user clicks the control. Passes an EventArgs argument.	Button ImageButton LinkButton
OnCommand	Action	Raised when a button containing OnCommand,CommandName, and CommandArgument attributes is clicked. Passes a CommandEventArgs argument containing the CommandName and CommandArgumentattribute values.	Button ImageButton LinkButton
OnSelectedIndexChanged	Change	Raised when the user changes the selection. Passes an EventArgs argument.	CheckBoxList DropDownList ListBox RadioButtonList
OnCheckedChanged	Change	Raised when the user clicks the control.	CheckBox

Event	Event type	Description	Controls
		Passes an EventArgs argument.	RadioButton
OnPageIndexChanged	Change	Raised when the user clicks a page selection element. Passes a DataGridPageChangedEventArgs argument.	DataG

UNDERSTAND THE CREATION OF MASTER PAGES AND THEMES

Master and content Pages

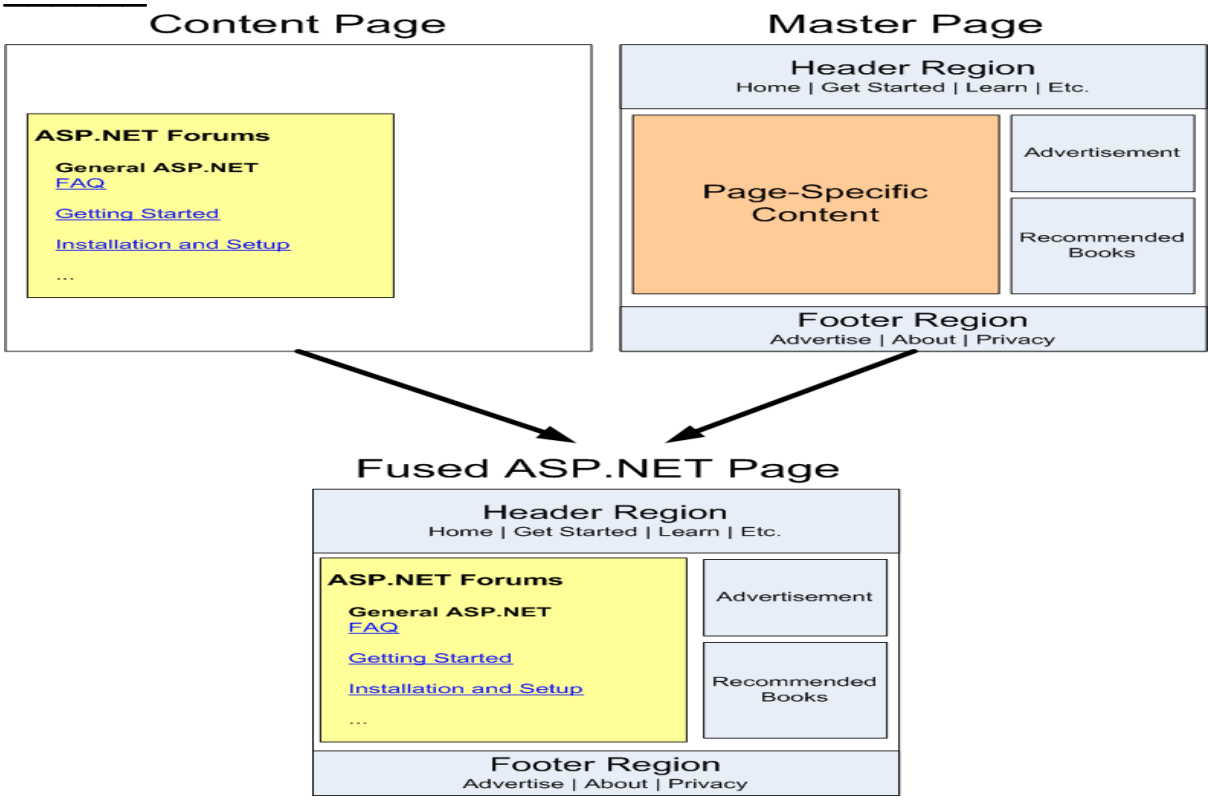
Master page Stores global page elements that occur on every content page. Master pages allow you to create a consistent look and behaviour for all the pages (or group of pages) in your web application.

A master page provides a template for other pages, with shared layout and functionality. The master page defines placeholders for the content, which can be overridden by content pages. The output result is a combination of the master page and the content page.

Extension: ".Master"

Content page Stores page-specific elements that are put into the master.The content pages contains the content you want to display.When users 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.

Extension: ".aspx"



Master Page Example

```
<%@ Master %>

<html>
<body>
<h1>Standard Header For All Pages</h1>
<asp:ContentPlaceHolder id="CPH1" runat="server">
</asp:ContentPlaceHolder>
</body>
</html>
```

The master page above is a normal HTML page designed as a template for other pages.

The **@ Master** directive defines it as a master page.

The master page contains a placeholder tag **<asp:ContentPlaceHolder>** for individual content.

The **id="CPH1"** attribute identifies the placeholder, allowing many placeholders in the same master page.

This master page was saved with the name **"master1.master".**

Note: The master page can also contain code, allowing dynamic content.

Content Page Example

```
<%@ Page MasterPageFile="master1.master" %>

<asp:Content ContentPlaceHolderId="CPH1" runat="server">
  <h2>Individual Content</h2>
  <p>Paragraph 1</p>
  <p>Paragraph 2</p>
</asp:Content>
```

The content page above is one of the individual content pages of the web.

The **@ Page** directive defines it as a standard content page.

The content page contains a content tag **<asp:Content>** with a reference to the master page (ContentPlaceHolderId="CPH1").

This content page was saved with the name **"mypage1.aspx"**.

When the user requests this page, ASP.NET merges the content page with the master page.

Note: The content text must be inside the <asp:Content> tag. No content is allowed outside the tag.

Content Page With Controls

```
<%@ Page MasterPageFile="master1.master" %>

<asp:Content ContentPlaceHolderId="CPH1" runat="server">
  <h2>New Heading</h2>
  <form runat="server">
    <asp:TextBox id="textbox1" runat="server" />
    <asp:Button id="button1" runat="server" text="Button" />
  </form>
</asp:Content>
```

The content page above demonstrates how .NET controls can be inserted into the content page just like an into an ordinary page.

THEMES IN ASP.NET

A theme decides the look and feel of the website. It is a collection of files that define the looks of a page. It can include skin files, CSS files & images.

We can define themes in a special App_Themes folder. Inside this folder is one or more subfolders named Theme1, Theme2 etc. that define the actual themes. The theme property is

applied late in the page's life cycle, effectively overriding any customization you may have for individual controls on your page.

How to apply themes

There are 3 different options to apply themes to our website:

1. Setting the theme at the page level: the Theme attribute is added to the page directive of the page.

   ```
   <%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"Inherits="Default" Theme="Theme1"%>
   ```

2. Setting the theme at the site level: to set the theme for the entire website you can set the theme in the web.config of the website. Open the web.config file and locate the <pages> element and add the theme attribute to it:

   ```
   <pages theme="Theme1">
   ....
   ....
   </pages>
   ```

3. Setting the theme programmatically at runtime: here the theme is set at runtime through coding. It should be applied earlier in the page's life cycle ie. Page_PreInit event should be handled for setting the theme. The better option is to apply this to the Base page class of the site as every page in the site inherits from this class.

   ```
   Page.Theme = Theme1;
   ```

Uses of Themes

1. Since themes can contain CSS files, images and skins, you can change colors, fonts, positioning and images simply by applying the desired themes.

2. You can have as many themes as you want and you can switch between them by setting a single attribute in the web.config file or an individual aspx page. Also you can switch between themes programmatically.

3. Setting the themes programmatically, you are offering your users a quick and easy way to change the page to their likings.

4. Themes allow you to improve the usability of your site by giving users with vision problems the option to select a high contrast theme with a large font size.

To create a page theme

1. 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**.
2. 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.

3. 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.

4. Add files to your new folder for control skins, style sheets, and images that make up the theme.

To add a skin file and a skin to a page theme

1. In Solution Explorer, right-click the name of your theme and then click **Add New Item**.
2. In the **Add New Item** dialog box, click **Skin File.**
3. In the **Name** box, type a name for the .skin file, and then click **Add**.

   The typical convention is to create one .skin file per control, such as Button.skin or Calendar.skin. However, you can create as many or as few .skin files as you need.

4. In the .skin file, add a normal control definition by using declarative syntax, but include only the properties that you want to set for the theme. The control definition must include the runat="server" attribute, and it must not include the ID="" attribute.

   The following code example shows a default control skin for a Button control, defining the color and font for all of the **Button** controls in the theme.

```
<asp:Textbox runat="server"
  BackColor="Red"
  ForeColor="White"
  Font-Name="Arial"
  Font-Size="9px" />
```

   This Textbox control skin does not contain a **skinID** attribute. It will be applied to all of the Textbox controls in the themed application that do not specify the **skinID** attribute.

5. Repeat steps 2 and 3 for each control skin file that you want to create.

   You can define only one default skin per control. Use the **SkinID** attribute in the skin's control declaration to create named skins for the same type of control.

To add a cascading style sheet file to your page theme

1. In Solution Explorer, right-click the name of your theme and then click **Add New Item**.
2. In the **Add New Item** dialog box, click **Style Sheet**.
3. In the **Name** box, type a name for the .css file, and then click **Add**.

STATE MANAGEMENT IN ASP.NET

A new instance of the Web page class is created each time the page is posted to the server. In traditional Web programming, this would typically mean that all information associated with

the page and the controls on the page would be lost with each round trip. For example, if a user enters information into a text box, that information would be lost in the round trip from the browser or client device to the server.
To overcome this inherent limitation of traditional Web programming, ASP.NET includes several options that help you preserve data on both a per-page basis and an application-wide basis. These features are as follows:

- View state
- Control state
- Hidden fields
- Cookies
- Query strings
- Application state
- Session state
- Profile Properties

View state, control state, hidden fields, cookies, and query strings all involve storing data on the client in various ways. However, application state, session state, and profile properties all store data in memory on the server. Each option has distinct advantages and disadvantages, depending on the scenario.PostBack is the name given to the process of submitting an ASP.NET page to the server for processing.During every postback data is lost.To preserve the data several state management techniques are used.We will see some of the key techniques used widely.

	Viewstate	QueryString	Cookies	Session
Stores data for a page	Yes	Yes	Yes	Yes
Stores data across Pages	No	Targeted Page	All	All
Where it gets stored	In a Viewstate Variable	In URL	On client machine	On server

1)Viewstate

- If a site happens to not maintain a ViewState, then if a user has entered some information in a large form with many input fields and the page is refreshes, then the values filled up in the form are lost.
- The same situation can also occur on submitting the form. If the validations return an error, the user has to refill the form.
- Thus, submitting a form clears up all form values as the site does not maintain any state called ViewState.
- In ASP .NET, the ViewState of a form is maintained with a built-in state management technique keeps the state of the controls during subsequent postbacks by a particular user.
- The ViewState option can be disabled by including the directive <%@ Page EnableViewState="false"%> at the top of an .aspx page
- If a ViewState of a certain control has to be disabled, then set EnableViewState="false".

If you want to add one variable in **View State**,

```
ViewState["myviewstate"] = myValue;
```

For Retrieving information from View State

```
string Test = ViewState["myviewstate"];
```

Advantages:

- Simple for page level data
- Encrypted
- Can be set at the control level

Disadvantages:

- Overhead in encoding View State values
- Makes a page heavy

2)Querystring

Query strings are usually used to send information from one page to another page. They are passed along with URL in clear text. Now that cross page posting feature is back in asp.net 2.0, Query strings seem to be redundant. Most browsers impose a limit of 255 characters on URL length. We can only pass smaller amounts of data using query strings. Since Query strings are sent in clear text, we can also encrypt query values. Also, keep in mind that characters that are not valid in a URL must be encoded using Server.UrlEncode

Now I have one page which contains one textbox and button control I need to send textbox value to another page when we click on button control for that we need to write the code like this

```
Response.Redirect("Default2.aspx?UserId=" + txtUserId.Text);
```

In case if we need to send multiple parameters to another page we need to write code like this

```
Response.Redirect("Default2.aspx?UserId=" + txtUserId.Text +
"&UserName=" + txtUserName.Text);
```

Now we need to get these values in another page (here I mentioned Default2.aspx) by using QueryStringParameter values with variable names or index values that would be like this

```
lblUserId.Text = Request.QueryString["UserId"];
lblUserName.Text = Request.QueryString["UserName"];
```

Advantages:

- Simple to Implement

Disadvantages:

- Human Readable
- Client browser limit on URL length
- Cross paging functionality makes it redundant
- Easily modified by end user

3) Cookies

Cookies is a small piece of information stored on the client machine. This file is located on client machines "C:\Document and Settings\Currently_Login user\Cookie" path. Its is used to store user preference information like Username, Password,City and PhoneNo etc on client machines. We need to import namespace called Systen.Web.HttpCookie before we use cookie.

Type of Cookies

1. Persist Cookie - A cookie has not have expired time Which is called as Persist Cookie
2. Non-Persist Cookie - A cookie has expired time Which is called as Non-Persist Cookie

How to create a cookie?

Its really easy to create a cookie in the Asp.Net with help of Response object or HttpCookie

Example 1:

```
HttpCookie userInfo = new HttpCookie("userInfo");
userInfo["UserName"] = "Stevejobs";
userInfo["UserColor"] = "Black";
userInfo.Expires.Add(new TimeSpan(0, 1, 0));
Response.Cookies.Add(userInfo);
```

Example 2:

```
Response.Cookies["userName"].Value = "Stevejobs";
Response.Cookies["userColor"].Value = "Black";
```

How to retrieve from cookie?

Its easy way to retrieve cookie value form cookes by help of Request object.

Example 1:

```
string User_Name = string.Empty;
 string User_Color = string.Empty;
 User_Name = Request.Cookies["userName"].Value;
 User_Color = Request.Cookies["userColor"].Value;
```

Example 2:

```
string User_name = string.Empty;
string User_color = string.Empty;
HttpCookie reqCookies = Request.Cookies["userInfo"];
if (reqCookies != null)
{
    User_name = reqCookies["UserName"].ToString();
    User_color = reqCookies["UserColor"].ToString();
}
```

4) Session

Web is stateless, which means a new instance of a web page class is re-created each time the page is posted to the server. As we all know, HTTP is a stateless protocol, it can't hold client information on a page. If the user inserts some information and move to the next page, that data will be lost and the user would not be able to retrieve that information. What do we need here? We need to store information. **Session** provides a facility to store information on server memory. It can support any type of object to store along with our own custom objects. For every client,**session** data is stored separately, which means **session** data is stored on a per client basis. Storing and retrieving values in **session** are quite similar to that in ViewState.

The following code is used for storing a value to **session**:

```
Session["UserName"] = txtUser.Text;
```

Now, let's see how we can retrieve values from **session**:

```
Textbox1.text = Session["UserName"];
```