Validation control

Validation is the process of checking something against the creation. There are several situations in which the validations are required. Consider an example where user fills the form for which the minimum age is 18 years. In such a situation, when the user submits the form, the age is validated against the criteria. If criteria are satisfied, the form is accepted. Such type of implementation on a website can be implemented by the ASP.NET validation controls.

ASP.NET provides the following validation controls that can be attached to the input controls to validate the values that are entered by the user.

1. RequiredFieldValidator Control
2. RegularExpressionValidator Control
3. CompareValidator Control
4. RangeValidator Control
5. ValidationSummary Control
6. CustomValidator Control

1. RequiredFieldValidator Control

The control is used to check whether a server control, added to the web form has a value or not. For example, you can use this validation control to check whether the user has entered any value in the password textbox.

The RequiredFieldValidator control inherits the BaseValidator's properties and exposes those properties. The list of properties of the control are:

1. ControlToValidate: It specifies the ID of the control to be validated
2. ErrorMessage: It specifies the error message to be displayed when the validation condition fails
3. Text: It specifies the error message to be displayed
4. InitialValue: It is used to describe the initial value of the ControlToValidate

The following code demonstrates the control in ASP.NET as follows:

Code:

```
<form id="form1" runat="server">
  <div>
    <asp:label id="Label1" runat="server" text="Name"></asp:label>
    <asp:textbox id="Text1" runat="server"></asp:textbox>
    <asp:requiredfieldvalidator id="RequiredFieldValidator1" runat="server" errormessage="The field
cannot be null" backcolor="Red" controltovalidate="Text1" display="Dynamic"
enableclientscript="false"></asp:requiredfieldvalidator>
    <br/>
    <asp:button id="btn1" runat="server" text="Show"/>
  </div>
</form>
```

The output for the code is as shown below:

2. RegularExpressionValidator Control

The control is used to check whether the server control added to the web form matches with the specified regular expression or not. The regular expression can be the format of a telephone number or an email address.

The list of some properties of the control is as shown below:

1. ValidationExpression: It specifies the regular expression, when the validation is performed
2. ControlToValidate: The ID of the control to be validated is defined
3. ErrorMessage: It specifies the error message to be displayed when the validation fails

The following code snippet demonstrates the control on a web form.

Code:

```
<form id="form1" runat="server">
  <div>
    <asp:label id="label1" runat="server" text="Email id"></asp:label>
    <asp:textbox id="TextBox1" runat="server"></asp:textbox>
    <asp:regularexpressionvalidator id="RegularExpression1Validator1" runat="server"
errormessage="Not a valid email address" controltovalidate="TextBox1" validationexpression='"^[\w-\.]{1,}@([\w-]+\.)+[\w-]{2,3}$"' backcolor="Aqua">
    </asp:regularexpressionvalidator>
  </div>
</form>
```

The output for the code is as shown below:

3. CompareValidator Control

The control is used to compare the entered value with another value. The other value can be a number or a value entered into another control. The following list displays some of the properties of the control.

1. ControlToCompare: It specifies the ID of the control that will be used to compare values.
2. ControlToValidate: It specifies the ID of the control to be validated

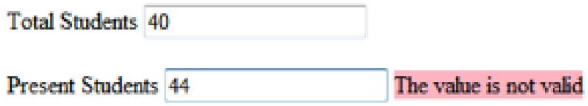
3. ErrorMessage: It specifies the error message to be displayed when the validation condition fails
4. ValueToCompare: It specifies the ID of the control to be compared

The following code snippet is used to demonstrate the control.

Code:

```
<form>
  <div>
    <asp:label id="Label2" runat="server" text="Total students"></asp:label>
    <asp:textbox id="TextBox2" runat="server"></asp:textbox>
    <br/>
    <br/>
    <asp:label id="Label3" runat="server" text="Present students></asp:label>
    <asp:textbox id="TextBox3" runat="server"></asp:textbox>
    <asp:comparevalidator id="CompareValidator1" runat="server" controltocompare="TextBox3"
controltovalidate="TextBox2" errormessage="The value is not valid" operator="GreaterThan"
type="Integer" backcolor="Pink">
    </asp:comparevalidator>
  </div>
</form>
```

The output for the code is as shown below:



4. RangeValidator Control

The control is used to check whether the value of a particular web form field is within the range specified by the control. The MinimumValue and MaximumValue properties are used for the validation. The properties can contain values as string, date, number, and currency amounts.

The list of properties of the control is as mentioned below:

1. ControlToValidate: It specifies the ID of the control to be validated
2. ErrorMessage: It specifies the error message to be displayed when the validation fails
3. MinimumValue: It specifies the minimum value
4. MaximumValue: It specifies the maximum value

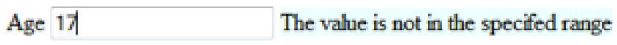
The following code is used to demonstrate the control.

Code:

```
<form>
  <div>
    <asp:label id="Label4" runat="server" text="Age"></asp:label>
    <asp:textbox="TextBox4" runat="server"></asp:textbox>
```

```
    <asp:rangevalidator=”RangeValidator1” runat=”server” controltovalidate=”TextBox4”
backcolor=”Azure” minimumvalue=”10” maximumvalue=”16” errormessage=”The value is not in the
specified range”></asp”rangevalidator>
    <br/>
  </div>
</form>
```

The output for the control is as shown below:



5. ValidationSummary Control

The control is used to summarize the errors and display the error list at a location specified by the user on the web page. The list of properties of the control is as shown below:

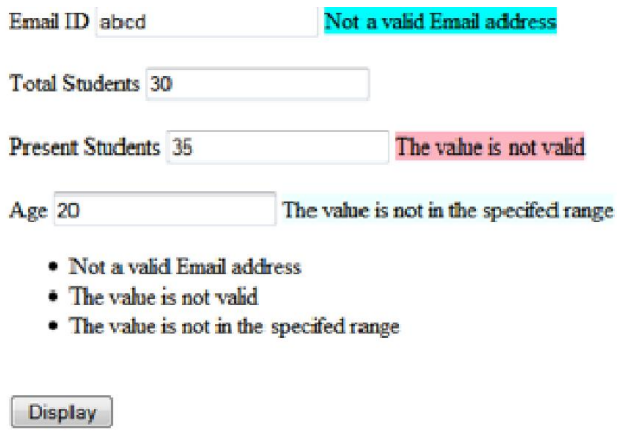
1. HeadText: It gets or sets the text to be displayed at the top of the summary
2. ShowMessageBox: It displays the error in a pop – up message box when the value of the property is true
3. ShowSummary: It enables or disables the summary of error messages

The code snippet to demonstrate the control is shown below:

Code:

```
<form>
  <div>
    <asp:validationsummary id=”Validation1” runat=”server” displaymode=”BulletList”
showsummary=”true”></asp:validationsummary>
    <br/>
    <asp:button id=”btn1” runat=”server” text=”Display”></asp:button>
  </div>
</form>
```

The output for the code is as shown below:



6. CustomValidator Control

The CustomValidator control is used to perform user defined validations that cannot be performed by standard validation controls. The list of properties of the control is as shown below:

1. ControlToValidate: The ID of the control to be validated is displayed
2. ErrorMessage: It specifies the error message to be displayed
3. OnServerValidate: It defines the function to be validated

The following code snippet demonstrates the control.

Code:

```
<asp:label id=”label5” runat=”server” text=”Number”></asp:label>
<asp:textbox id=”TextBox5” runat=”server”></asp:textbox>
<asp:button id=”btn3” runat=”server” text=”Output”></asp:button>
<asp:customvalidator id=”custom1” runat=”server” onservervalidate=”validateNumber”
errormessage=”Length is not valid” backcolor=”Yellow” enableclientscript=”false” display=”Dynamic”>
</asp:customvalidator>
```

The output for the code is as shown below:

Adrotator control

The AdRotator control randomly selects banner graphics from a list, which is specified in an external XML schedule file. This external XML schedule file is called the advertisement file.

The AdRotator control allows you to specify the advertisement file and the type of window that the link should follow in the AdvertisementFile and the Target property respectively.

The basic syntax of adding an AdRotator is as follows:

```
<asp:AdRotator  runat = "server" AdvertisementFile = "adfile.xml"  Target =  "_blank" />
```

Before going into the details of the AdRotator control and its properties, let us look into the construction of the advertisement file.

The Advertisement File

The advertisement file is an XML file, which contains the information about the advertisements to be displayed.

Extensible Markup Language (XML) is a W3C standard for text document markup. It is a text-based markup language that enables you to store data in a structured format by using meaningful tags. The term 'extensible' implies that you can extend your ability to describe a document by defining meaningful tags for the application.

XML is not a language in itself, like HTML, but a set of rules for creating new markup languages. It is a meta-markup language. It allows developers to create custom tag sets for special uses. It structures, stores, and transports the information.

Following is an example of XML file:

```
<BOOK>
  <NAME> Learn XML </NAME>
  <AUTHOR> Samuel Peterson </AUTHOR>
  <PUBLISHER> NSS Publications </PUBLISHER>
  <PRICE> $30.00</PRICE>
</BOOK>
```

Like all XML files, the advertisement file needs to be a structured text file with well-defined tags delineating the data. There are the following standard XML elements that are commonly used in the advertisement file:

Element	Description
Advertisements	Encloses the advertisement file.
Ad	Delineates separate ad.
ImageUrl	The path of image that will be displayed.
NavigateUrl	The link that will be followed when the user clicks the ad.
AlternateText	The text that will be displayed instead of the picture if it cannot be displayed.
Keyword	Keyword identifying a group of advertisements. This is used for filtering.

Impressions	The number indicating how often an advertisement will appear.
Height	Height of the image to be displayed.
Width	Width of the image to be displayed.

Apart from these tags, customs tags with custom attributes could also be included. The following code illustrates an advertisement file ads.xml:

```
<Advertisements>
  <Ad>
    <ImageUrl>rose1.jpg</ImageUrl>
    <NavigateUrl>http://www.1800flowers.com</NavigateUrl>
    <AlternateText>
      Order flowers, roses, gifts and more
    </AlternateText>
    <Impressions>20</Impressions>
    <Keyword>flowers</Keyword>
  </Ad>

  <Ad>
    <ImageUrl>rose2.jpg</ImageUrl>
    <NavigateUrl>http://www.babybouquets.com.au</NavigateUrl>
    <AlternateText>Order roses and flowers</AlternateText>
    <Impressions>20</Impressions>
    <Keyword>gifts</Keyword>
  </Ad>

  <Ad>
```

```
    <ImageUrl>rose3.jpg</ImageUrl>
    <NavigateUrl>http://www.flowers2moscow.com</NavigateUrl>
    <AlternateText>Send flowers to Russia</AlternateText>
    <Impressions>20</Impressions>
    <Keyword>russia</Keyword>
  </Ad>

  <Ad>
    <ImageUrl>rose4.jpg</ImageUrl>
    <NavigateUrl>http://www.edibleblooms.com</NavigateUrl>
    <AlternateText>Edible Blooms</AlternateText>
    <Impressions>20</Impressions>
    <Keyword>gifts</Keyword>
  </Ad>
</Advertisements>
```

Properties and Events of the AdRotator Class

The AdRotator class is derived from the WebControl class and inherits its properties. Apart from those, the AdRotator class has the following properties:

Properties	Description
AdvertisementFile	The path to the advertisement file.
AlternateTextFeild	The element name of the field where alternate text is provided. The default value is AlternateText.
DataMember	The name of the specific list of data to be bound when advertisement file is not used.
DataSource	Control from where it would retrieve data.
DataSourceID	Id of the control from where it would retrieve data.

Font	Specifies the font properties associated with the advertisement banner control.
ImageUrlField	The element name of the field where the URL for the image is provided. The default value is ImageUrl.
KeywordFilter	For displaying the keyword based ads only.
NavigateUrlField	The element name of the field where the URL to navigate to is provided. The default value is NavigateUrl.
Target	The browser window or frame that displays the content of the page linked.
UniqueID	Obtains the unique, hierarchically qualified identifier for the AdRotator control.

Following are the important events of the AdRotator class:

Events	Description
AdCreated	It is raised once per round trip to the server after creation of the control, but before the page is rendered
DataBinding	Occurs when the server control binds to a data source.
DataBound	Occurs after the server control binds to a data source.
Disposed	Occurs when a server control is released from memory, which is the last stage of the server control lifecycle when an ASP.NET page is requested
Init	Occurs when the server control is initialized, which is the first step in its lifecycle.
Load	Occurs when the server control is loaded into the Page object.
PreRender	Occurs after the Control object is loaded but prior to rendering.
Unload	Occurs when the server control is unloaded from memory.

Working with AdRotator Control

Create a new web page and place an AdRotator control on it.

```
<form id="form1" runat="server">

  <div>

    <asp:AdRotator ID="AdRotator1" runat="server" AdvertisementFile  ="~/ads.xml"
onadcreated="AdRotator1_AdCreated" />

  </div>

</form>
```

The ads.xml file and the image files should be located in the root directory of the web site.

Try to execute the above application and observe that each time the page is reloaded, the ad is changed.

Login controls

The Login Controls:-

There are following Login controls developed by the Microsoft which are used in ASP.NET Website as given below:-

1. Login
2. LoginView
3. LoginStatus
4. Loginname
5. PasswordRecovery
6. ChangePassword
7. CreateUserWizard

1.) The Login Control:-

The Login control provides a user interface which contains username and password, that authenticate the usernaMe and password and grant the access to the desired services on the the basis of the credentials.

There are used some methods ,properties and events in this Login control,You can check manually after drag and drop this control on your web form as given below:-

2.) The LoginView Control:-

The LoginView Control is a web server control ,Which is used to display two different views of a web page of any website , dependending on whether the any user has logged on to a web page as anonymous user or registered user .If the user is authenticated,the control displays the appropriate to the person with the help of the following views template.

- **Anonymous Template** :- This template (default of all) will be displayed when any user just open the web page but not logged in.
- **LoggedInTemplate**:- This Template (page)will be displayed when the user in logged in.
- **RoleGroups**:- This template will be displayed when user logged in, that is the member of the specific role (defined role group).

You can drag and drop Loginview Control on the web page from toolbox as shown below:-

Note:- You can do so, by adding any server controls such as Label ,Hyperlink and TextBox, to the empty region on the loginview control.

3.) The LoginStatus Control :-

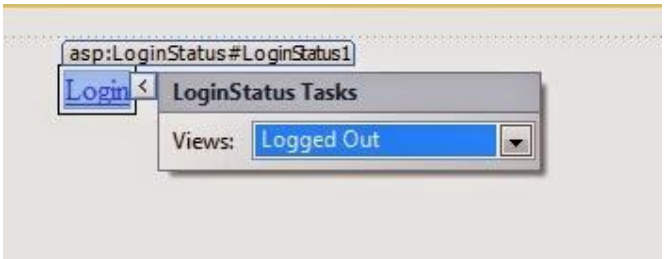
The LoginStatus control specifies whether a particular user has logged on the website or not . When the user is not logged in,this control display the Login text as a hyperlink.When the user is logged in ,this control display the logout text as a hyperlink.

To do this ,Login Status control uses the authentication section of the web.config file.

This control provides the following two views:-

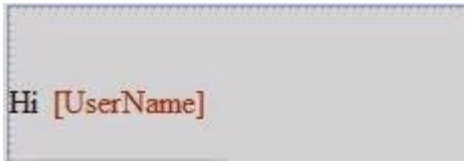
1. **LoggedIn** --> It will be displayed,when the user is not Logged In.
2. **Logout** --> It will be displayed when the user is Logged In.

You can drag and drop this control on the web page as shown below:-



4.) The LoginName Control :-

The LoginName Control is responsible for display the names of all the authenticated users.If no users id logged in ,then this control is not displayed on the page.
This control uses the page .User.Identity.Name namespace to return the value for the user's name.
You can drag and drop Login Name control on the page from the toolbox as given below:-



5.) Passwordrecovery Control:-

The Passwordrecovery control is used to recover or reset the forgotten password of a user . This control does not display the password on the browser,but it sends the password to the respective email address whatever you have specified at the time of registration.
This control has included three views as given below:-

1. **UserName** :- It refers to the view that accepts the username of a user.
2. **Question** :- It accepts the security questions asked from the users.
3. **Success** :- It display a message to the user that retrieved password has been set to the user.

You can easily drag and drop this control on the web a page as shown below.

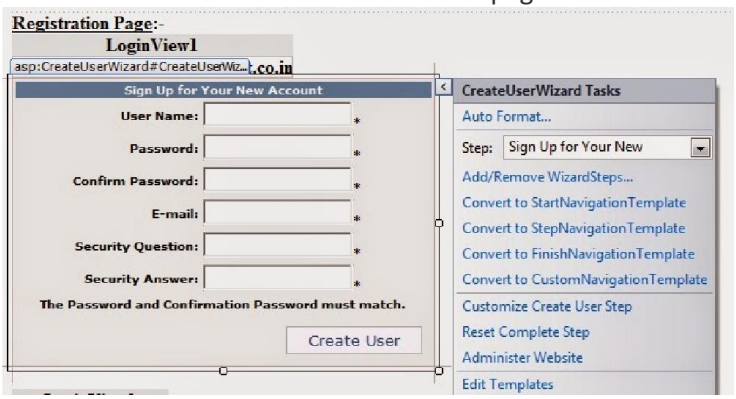
Note :-

- To retrieve and reset password you must set some properties inside the asp.net Membership services.
- You can can learn its methods ,properties and events from PasswordRecovery class yourself.

6.) CreateUserWizard control:-

This control uses the membership service to create a new user in the membership data store.The CreateUserWizard control is provides by the CreateUserWizard class and can be customized by using template and style properties .Using this control any user can easily create an account and login to the web page.

You can drag and drop CreateUserWizared control on the web page as shown below:-



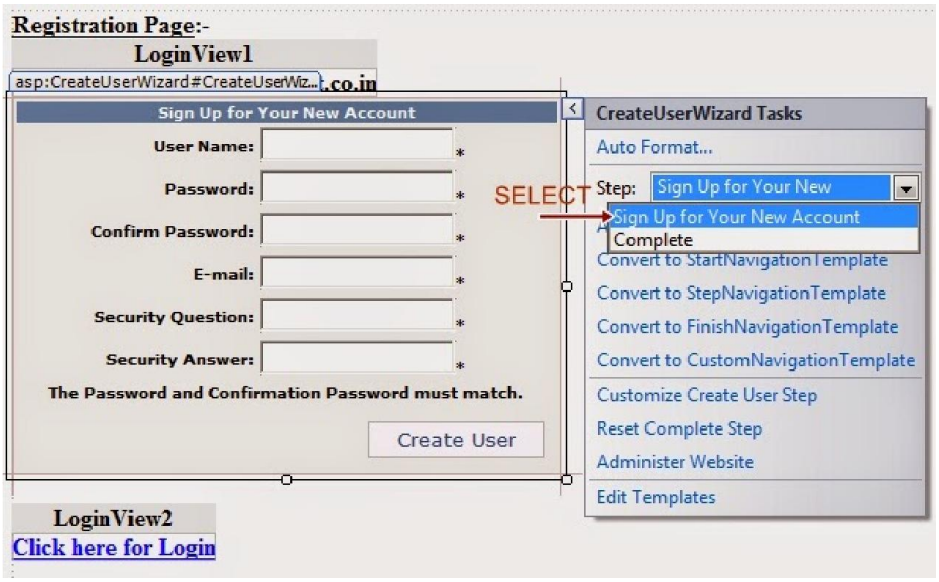
7.) The ChangePassword Control:-

Using this control ,user can easily change your existing password (old password) on the ASP.NET Website.This control prompts uses to provide the current password first and then set the new password first and then set the new password.If the old password is not correct then new password can't be set. This is also helps to send the email to the respective users about the new password.This control is used ChangePassword class.

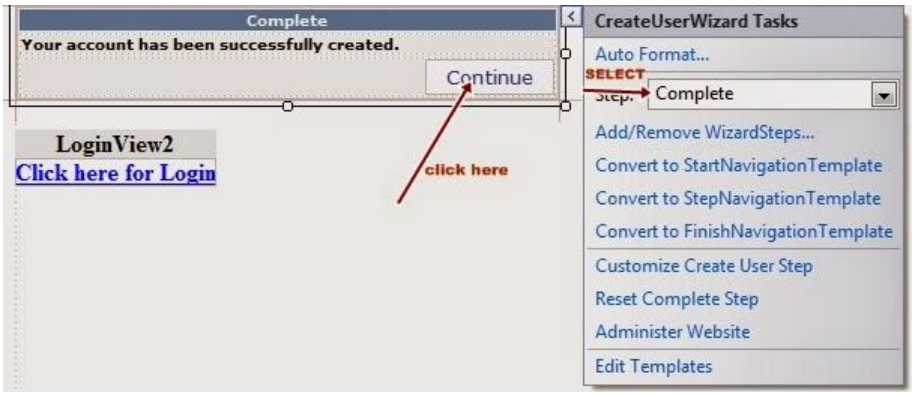
There are some steps to use Login controls concepts in ASP.NET Application as given below:-

Step 1 :- First open your visual studio -->File -->New -->Select ASP.NET Empty website --> OK -->Open Solution Explorer -->Add a New web form (login.aspx) -->Now drag and Drop Login control and and LoginView control on the page from toolbox --> Add a Hyperlink control in LoginView 's blank space as shown below:-

Step 2 :- Now open Solution Explorer --> Add a New web Form (Registrationpage.aspx) --> Drag and drop CreateUserWizard and LoginView controls on the page --> Put a Hyperlink control inside blank space in LoginView control as shown below:-



- Now select **Complete** from createUserWizard Tasks as shown above --> Now double click on Continue button and write the following c# codes for Navigation as given below:-



Master page

Master pages provide templates for other pages on your web site.

Master Pages

Master pages allow you to create a consistent look and behavior 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.

The content pages contain 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.

Master Page Example

```
<%@ Master %>

<html>
<body>
<h1>Standard Header From Masterpage</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".**

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.

Content Page With Controls

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

<asp:Content ContentPlaceHolderId="CPH1" runat="server">
  <h2>W3Schools</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.

Advantages

- They allow you to centralize the common functionality of your pages so that you can make updates in just one place.
- They make it easy to create one set of controls and code and apply the results to a set of pages. For example, you can use controls on the master page to create a menu that applies to all pages.
- They give you fine-grained control over the layout of the final page by allowing you to control how the placeholder controls are rendered.
- They provide an object model that allows you to customize the master page from individual content pages.

Themes

Themes are the way to define the formatting details for various controls and can be reused in multiple pages. Later, by applying minor changes on the themes, the complete appearance of website can be changed with maintaining consistency.

Why another formatting feature? CSS is a very convenient formatting feature but it is limited to the generic formatting details (fonts, borders, background etc) but sometimes it's required to achieve control specific formatting. Asp.net 2.0 provides a new feature, "Themes", to fill this gap.

It's different to CSS in several manners:
1. Themes are control based not Html Based.
2. Themes are implemented on the server side while in case of style sheets; client receives both page and css and combines them at client side.
3. Themes can be applied through configuration files.
4. Themes provide the flexibility to retain the css feature instead of blindly overriding it.

Creating Themes Folder:
Create a folder (with specific name) under App_Theme folder, which is under application main directory then add one or more skin files (text file with .skin extension) under the created theme folder. Application may contain many themes and all of them should be defined under different theme folders. But only one theme can be active on a page at a time.

Creating Skin Files:
Skin file is a text file with .skin extension. It contains a set of control tags with standardized properties as:

<asp:TextBox runat="server" BackColor="Orange"/>

The runat="server" portion is always required. Everything else is optional while id attribute is not allowed in a theme.

It's also possible to place multiple skin files under the theme folder, but ASP.NET treats them all as a part of single theme definition. It's more useful in case if separate skin files are written for complex controls (Calender, Grid View) to distinguish their formatting from other simple controls.

Note: Visual studio doesn't provide any design time support for creating themes, so it's better approach to copy the tag from the web page and modify it as per theme definition.

Applying Themes:
To apply the theme in a web page, you need to set the Theme attribute of the Page directive to the folder name for your theme. (ASP.NET will automatically scan all the skin files in that theme.)
<%@ Page Language="C#" AutoEventWireup="true" ... Theme="MyTheme" %>

Once a theme is applied to a page, ASP.NET considers each control on that web page and checks the registered skin files to see if they define any properties for that control. If ASP.NET finds a matching tag in the skin file, the information from the skin file overrides the current properties of the control.

But sometimes, it's required to override some of theme properties with self defined properties. This can be achieved by using StyleSheetTheme attribute in place of Theme. It will restrict the theme settings for those control properties which are already set and the other properties will take configuration from themes (The same thing happens when we apply css classes).
<%@ Page Language="C#" AutoEventWireup="true" ... StyleSheetTheme="MyTheme" %>

There is also an option (EnableTheming = 'False') to restrict themes for all the properties under the specific control.
<asp: TextBox ID="TextBox1" runat="server" ... EnableTheming="false" />

Creating multiple skins for the same control:
If more than one theme is defined for the same control, ASP.NET gives build error. But the situation can be handled by supplying SkinID attribute to the conflicting themes for the same control.
<asp:TextBox runat="server" BackColor="Orange" />
<asp:TextBox runat="server" BackColor="Red" SkinID="Dramatic"/>

Correspondingly, set the particular SkinId for the web control.
<asp: TextBox ID=" TextBox 1" runat="server" ... SkinID="Dramatic" />

Note: It's possible to use same skinID for several themes which are associated to different control.
<asp:TextBox runat="server" BackColor="Red" SkinID="Dramatic"/>
<asp:ListBox runat="server" BackColor="Red" SkinID="Dramatic"/>

Using CSS in a theme:
Themes can be further standardized using css classes. To do so, place the css class under the respective theme folder. ASP.Net searches for all css classes under this folder and dynamically binds them to the web page (that uses this particular theme) through link attribute (It's only possible if runat="server" attribute is setfor the web page <head> section)
<head runat="server">

Applying themes through configuration files:
A particular theme can be applied throughout the web site using Web.config file. As:
<Configuration>

```
<system.web>
<pages theme="MyTheme" />
OR
<pages styleSheetTheme="MyTheme" />
</system.web>
</Configuration>
```

Applying themes dynamically:
Use Page.Theme or Page.StyleSheet property(Page.Theme=”MyTheme”) at the Pre_Init event. So if it’s required to change the theme, the pre_init event should be triggered again. This can be achieved by forcefully performing page postback(eg, page redirection to same page).

Similiary, the control’s SkinID property can also be set dynamically, in case of named skins.

One of the limitations of applying themes dynamically is that you can't apply theme within a user control or master page as neither of them has a Pre_Init event.