Class:- T.Y.B.C.A SEM-V

Course: 505: ASP .NET

Unit 1. Introduction to ASP.NET

1.1 What is ASP.NET

- **What is ASP.NET?(ActiveX Server Pages)**
- Microsoft ASP.NET is a server side technology that enables programmers to build dynamic Web sites, web applications, and XML Web services.
- It is a part of the .NET based environment and is built on the Common Language Runtime (CLR).
- So programmers can write ASP.NET code using any .NET compatible language.

How ASP.NET Works ?

1. When a browser requests an asp.net file, IIS passes the request to the ASP.NET Engine on the server.
2. Then asp.net engine read the file line by line and execute the scripts in the file.
3. Finally the ASP.NET file is returned to the browser as a plain HTML file.

ESOFT

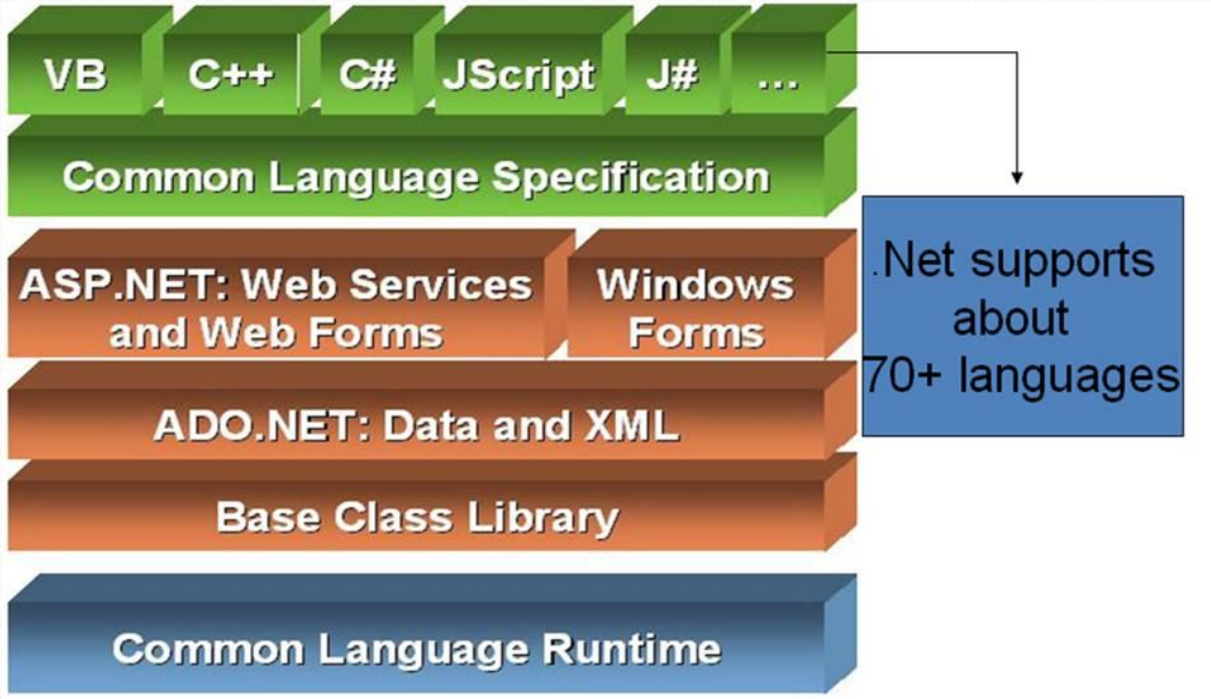
Version of .NET Framework:

Version	Release Year	Visual Studio
.Net Framework 1.0	2002	Visual Studio .Net
.Net Framework 1.1	2003	Visual Studio .Net 2003
.Net Framework 2.0	2005	Visual Studio 2005
.Net Framework 3.0	2006	
.Net Framework 3.5	2007	Visual Studio 2008
.Net Framework 4.0	2010	Visual Studio 2010
.Net Framework 4.5	2012	Visual Studio 2012
.Net Framework 4.6	2015	Visual Studio 2015
.Net Framework 4.7	2017	Visual Studio 2017
.Net Framework 4.8	2019	Visual Studio 2019

Link for download SQL Express

- https://www.microsoft.com/en-in/sql-server/sql-server-downloads

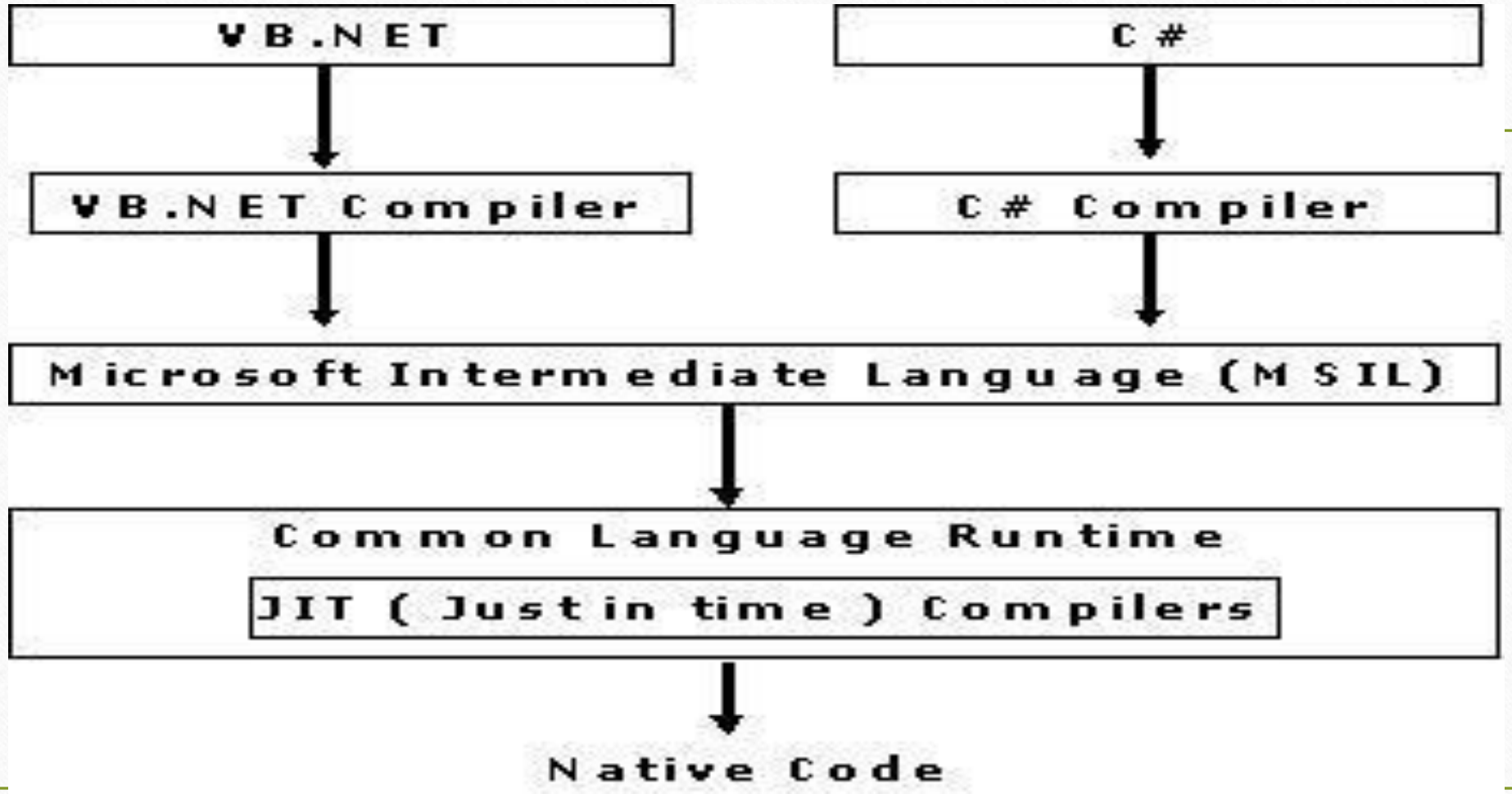
1.2 .NET framework 2.0



.NET Framework
mainly contains two components,

1. Common Language Runtime (CLR)
2. .NET Framework Class Library.

1. **Common Language Runtime** (CLR)



2. **.NET Framework Class Library** (FCL)

- The following are different types of applications that can make use of .net class library.
  1. Windows Application.
  2. Console Application
  3. Web Application.
  4. XML Web Services.
  5. Windows Services.

And many more classes also like ADO .NET Databases and etc.

The .NET Framework includes a set of standard class libraries. A class library is a collection of methods and functions that can be used for the core purpose.

For example, there is a class library with methods to handle all file-level operations. So there is a method which can be used to read the text from a file. Similarly, there is a method to write text to a file.

Most of the methods are split into either the System.* or Microsoft.* namespaces. (The asterisk * just means a reference to all of the methods that fall under the System or Microsoft namespace)

A namespace is a logical separation of methods. We will learn these namespaces more in detail in the subsequent chapters.

3. **Common Type System** (CTS)

- It describes set of data types that can be used in different .Net languages in common. (i.e), CTS ensures that objects written in different .Net languages can interact with each other.

  For Communicating between programs written in any .NET compatible language, the types have to be compatible on the basic level.

4. **Common Language Specification** (CLS)

- It is a sub set of CTS and it specifies a set of rules that needs to satisfied by all language compilers targeting CLR. It helps in cross language inheritance and cross language debugging.

- **Common language specification Rules:**

  It describes the minimal and complete set of features to produce code that can be hosted by CLR. It ensures that products of compilers will work properly in .NET environment.

  Sample Rules:

  1. Representation of text strings

  2. Internal representation of enumerations

  3. Definition of static members and this is a subset of the CTS which all .NET languages are expected to support.

  4. Microsoft has defined CLS which are nothing but guidelines that language to follow so that it can communicate with other .NET languages in a seamless manner.

1.3 Compile Code

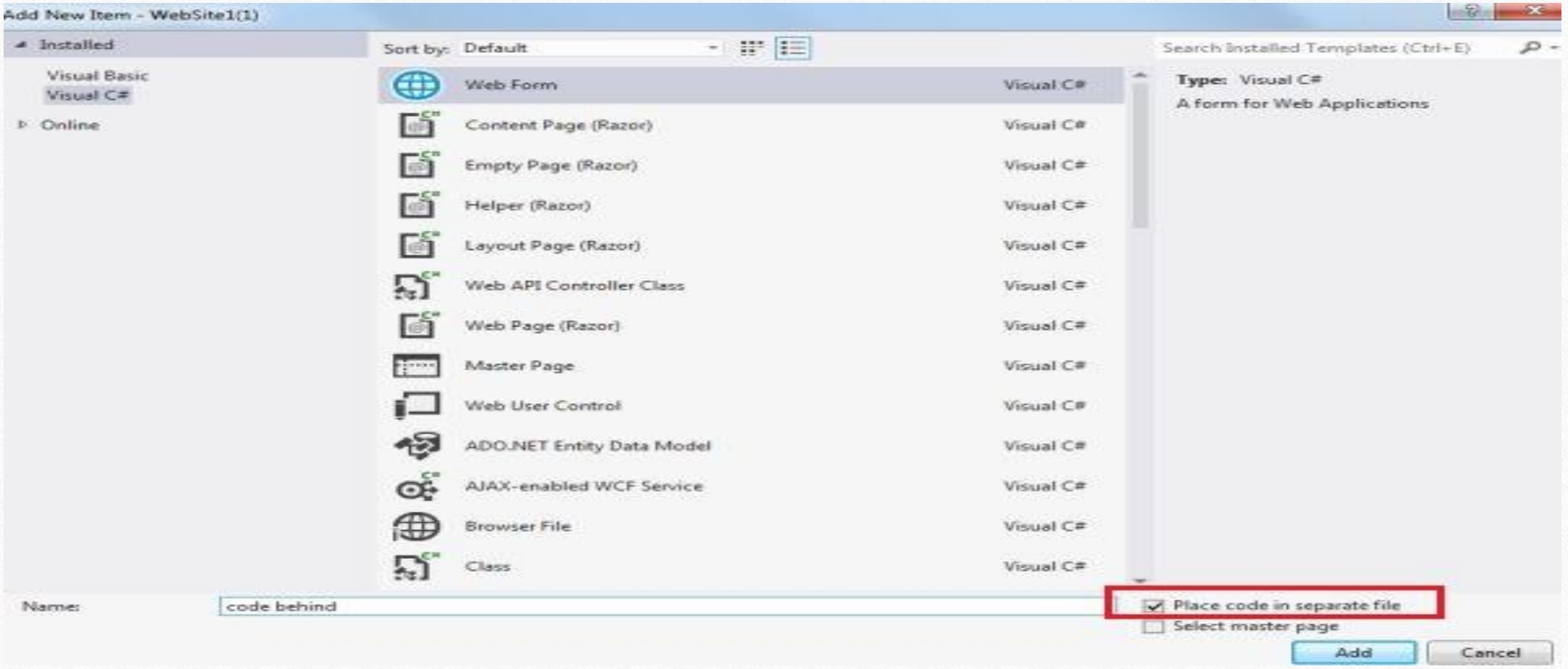
- *Compiled code* is a set of files that must be linked together and with one master list of steps in order for it to run as a program.

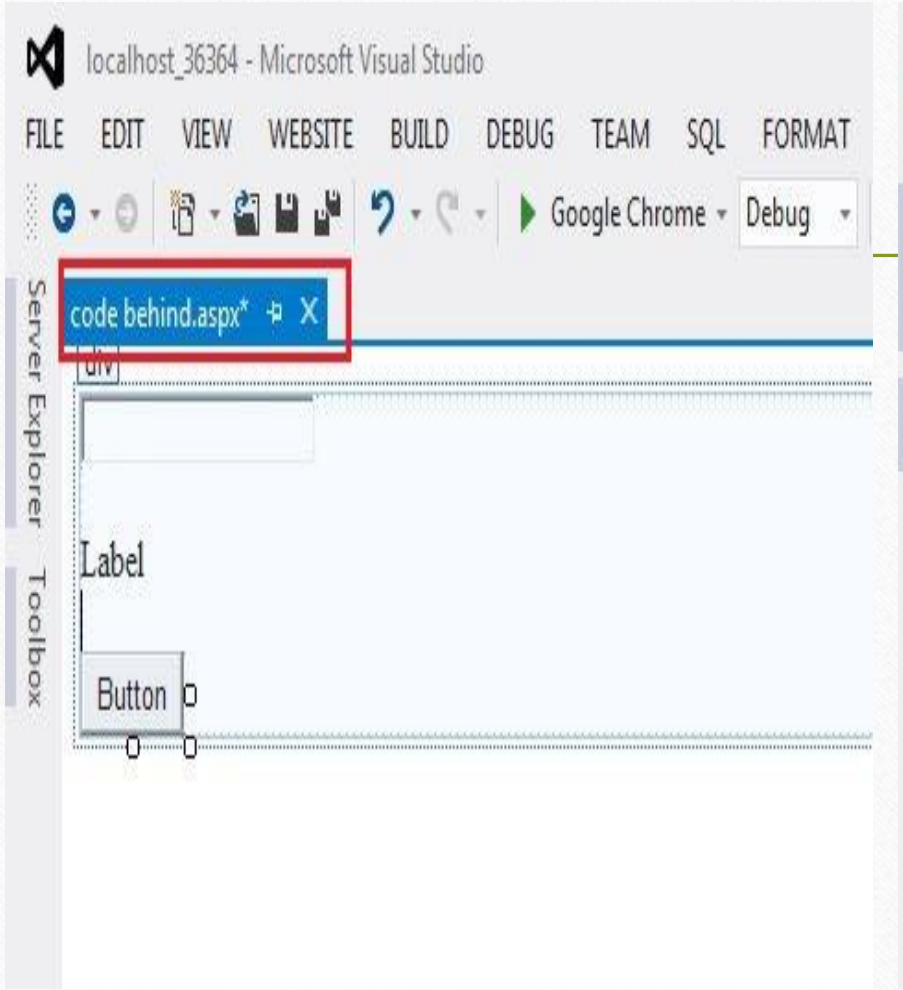
1.3.1 Code Behind and Inline Coding

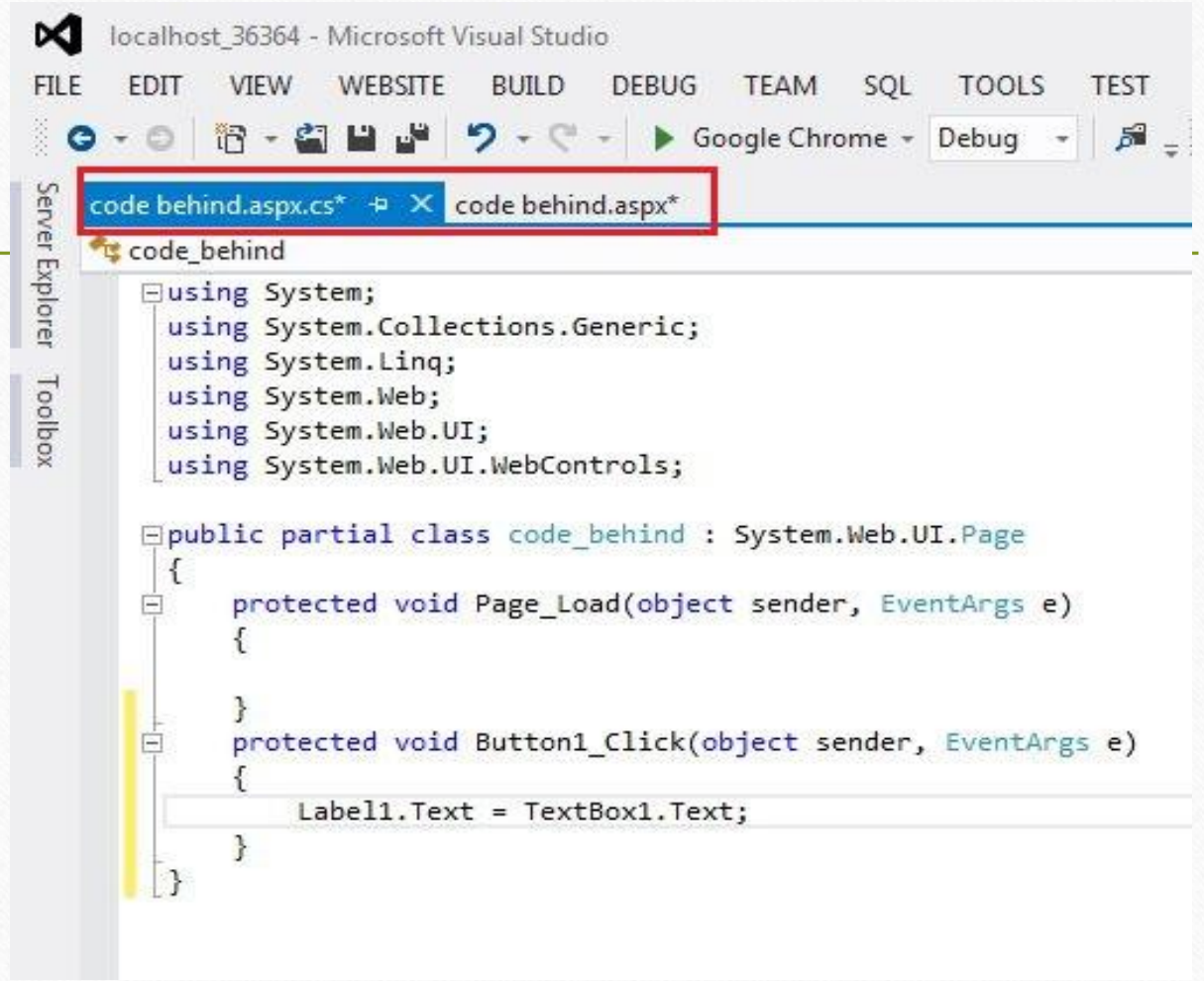
- **Code Behind**
- Code Behind refers to the code for an ASP.NET Web page that is written in a separate class file that can have the extension of .aspx.cs or .aspx.vb depending on the language used. Here the code is compiled into a separate class from which the .aspx file derives. You can write the code in a separate .cs or .vb code file for each .aspx page.
- One major point of Code Behind is that the code for all the Web pages is compiled into a DLL file that allows the web pages to be hosted free from any Inline Server Code.

- **Inline Code**
- Inline Code refers to the code that is written inside an ASP.NET Web Page that has an extension of .aspx. It allows the code to be written along with the HTML source code using a <Script> tag. It's major point is that since it's physically in the .aspx file it's deployed with the Web Form page whenever the Web Page is deployed.

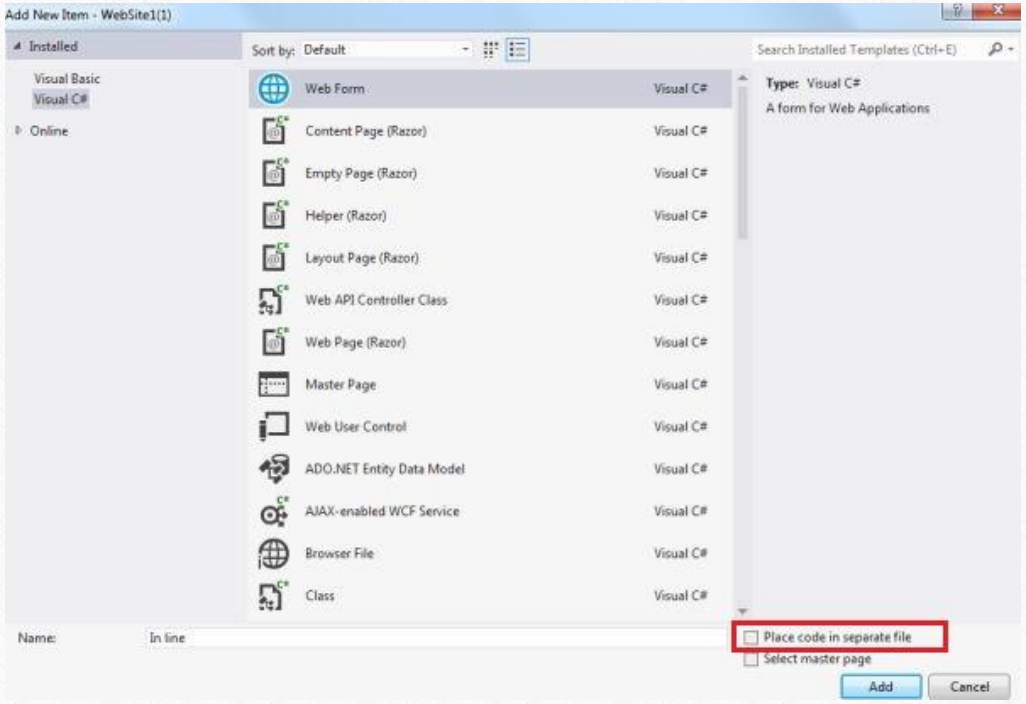
Code Behind

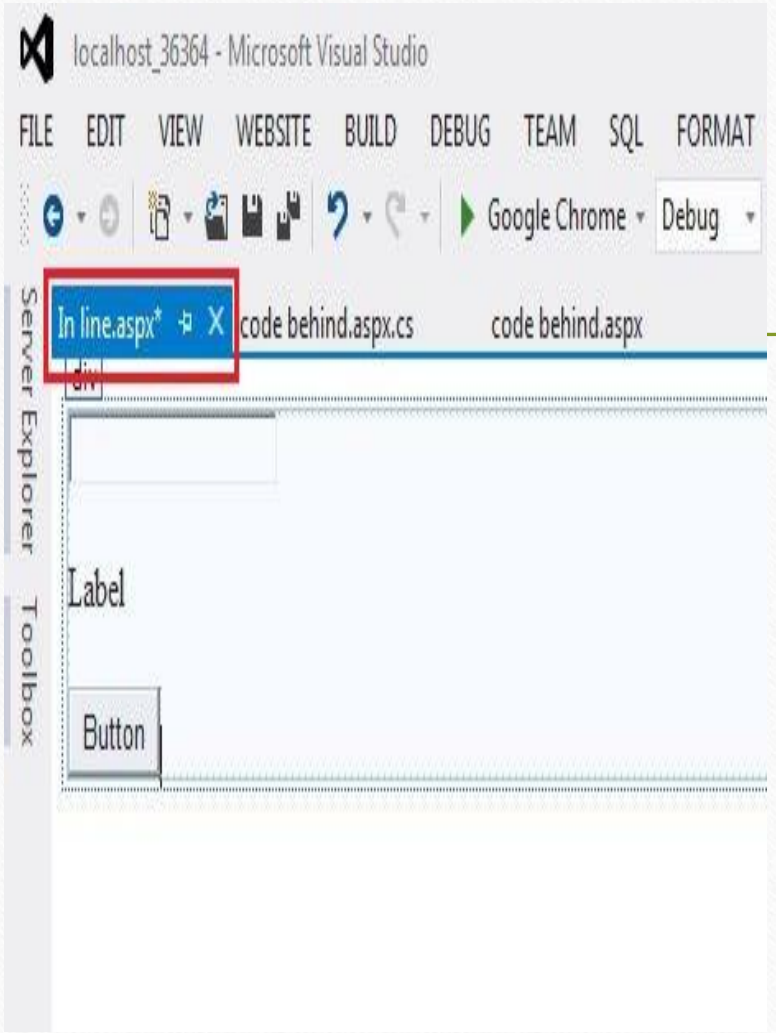

Add New Item - WebSite1(1)
Installed
Visual Basic
Visual C#
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
Web Form
Visual C#
Content Page (Razor)
Empty Page (Razor)
Helper (Razor)
Layout Page (Razor)
Web API Controller Class
Web Page (Razor)
Master Page
Web User Control
ADO.NET Entity Data Model
AJAX-enabled WCF Service
Browser File
Class
Type: Visual C#
A form for Web Applications
Name:
code behind
Place code in separate file
Select master page
Add
Cancel

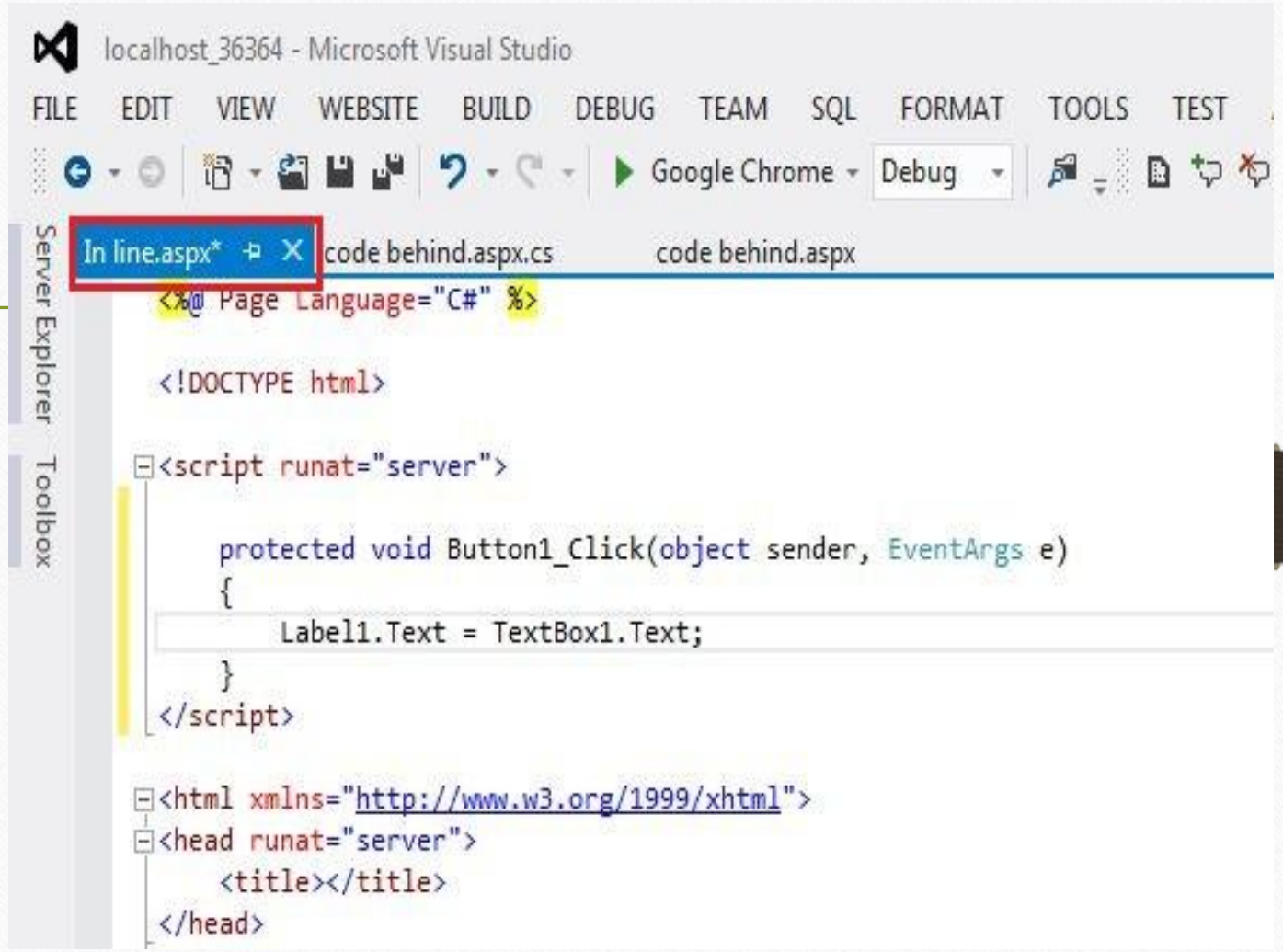

localhost_36364 - Microsoft Visual Studio
FILE EDIT VIEW WEBSITE BUILD DEBUG TEAM SQL FORMAT
Google Chrome
Debug
code behind.aspx*
Server Explorer
Toolbox
Label
Button


localhost_36364 - Microsoft Visual Studio
FILE EDIT VIEW WEBSITE BUILD DEBUG TEAM SQL TOOLS TEST
Google Chrome
Debug
code behind.aspx.cs*
code behind.aspx*
Server Explorer
Toolbox
code_behind
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
public partial class code_behind : System.Web.UI.Page
{
protected void Page_Load(object sender, EventArgs e)
{
}
protected void Button1_Click(object sender, EventArgs e)
{
Label1.Text = TextBox1.Text;
}
}

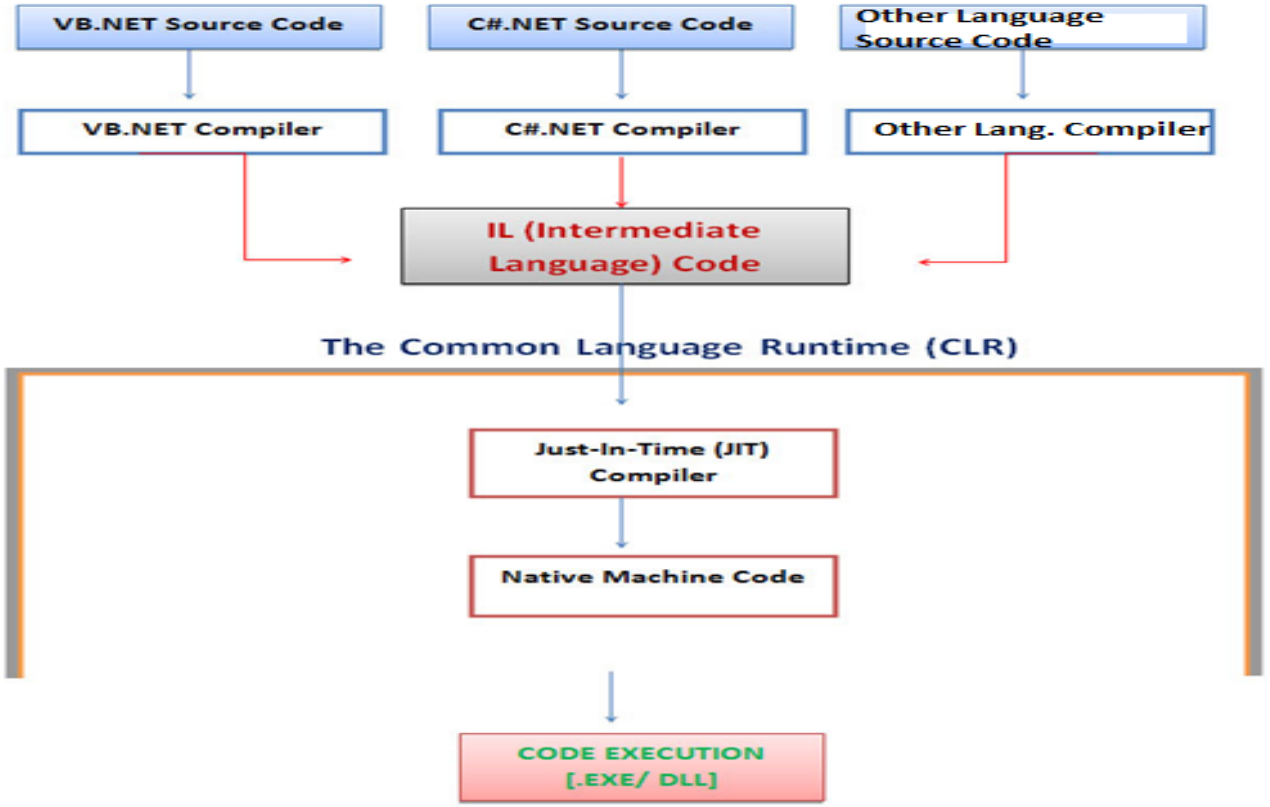
Inline Code


Add New Item - WebSite1(1)
Installed
Visual Basic
Visual C#
Online
Sort by: Default
Search Installed Templates (Ctrl+E)
Web Form
Visual C#
Content Page (Razor)
Empty Page (Razor)
Helper (Razor)
Layout Page (Razor)
Web API Controller Class
Web Page (Razor)
Master Page
Web User Control
ADO.NET Entity Data Model
AJAX-enabled WCF Service
Browser File
Class
Type: Visual C#
A form for Web Applications
Name:
In line
Place code in separate file
Select master page
Add
Cancel


localhost_36364 - Microsoft Visual Studio
FILE EDIT VIEW WEBSITE BUILD DEBUG TEAM SQL FORMAT
Google Chrome
Debug
In line.aspx*
code behind.aspx.cs
code behind.aspx
Server Explorer
Toolbox
Label
Button


localhost_36364 - Microsoft Visual Studio
FILE EDIT VIEW WEBSITE BUILD DEBUG TEAM SQL FORMAT TOOLS TEST
Google Chrome
Debug
In line.aspx*
code behind.aspx.cs
code behind.aspx
Server Explorer
Toolbox
<%@ Page Language="C#" %>
<!DOCTYPE html>
<script runat="server">
protected void Button1_Click(object sender, EventArgs e)
{
Label1.Text = TextBox1.Text;
}
</script>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>

1.4 The Common Language Runtime



The CLR has the following key features:

1)Exception Handling

2) Garbage Collection

3) Type Safety

4) Thread Management

5) Working with Various programming languages

-Language

-Compiler

-Common Language Interpreter

1.5 Object Oriented Concepts

- **OOPS Concepts, Features & Fundamentals**

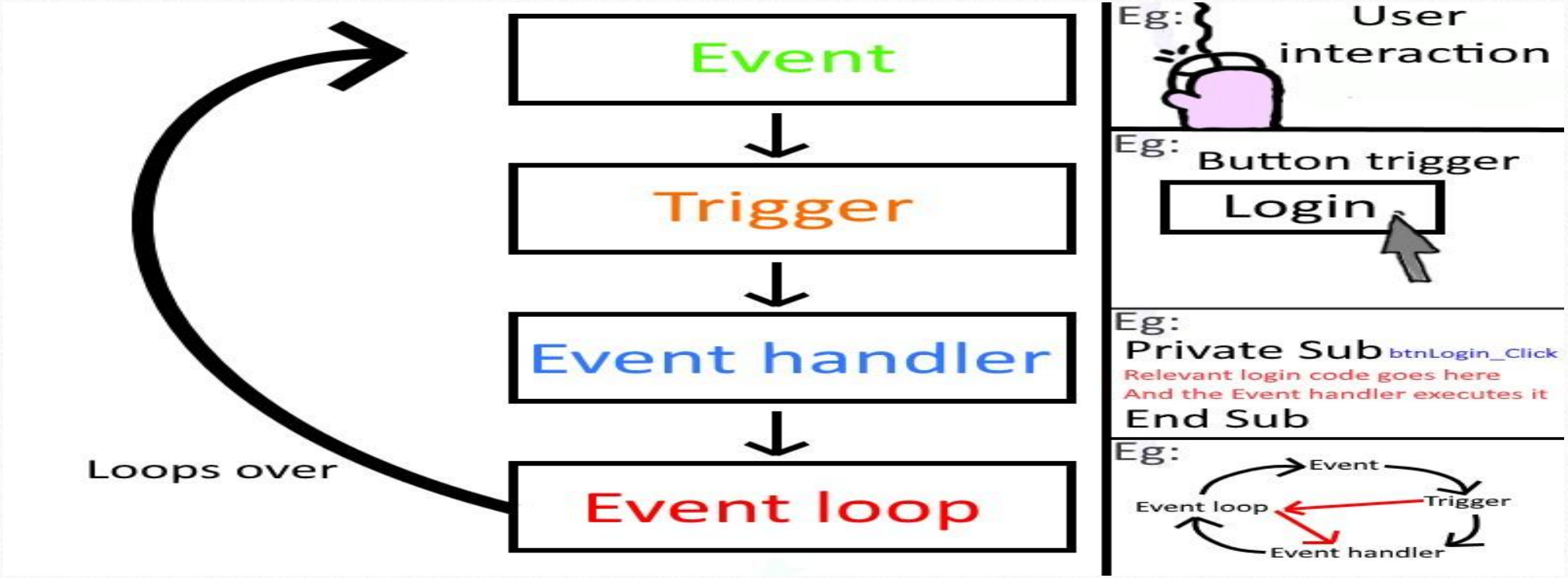
- **Class:-** A class is a collection of objects and represents description of objects that share same attributes and actions.
- **Method**:- Method is an object's behavior. ...
- **Object:-**Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.

- **Encapsulation: -***Binding (or wrapping) code and data together into a single unit are known as encapsulation.*
  For example, a capsule, it is wrapped with different medicines.

Capsule

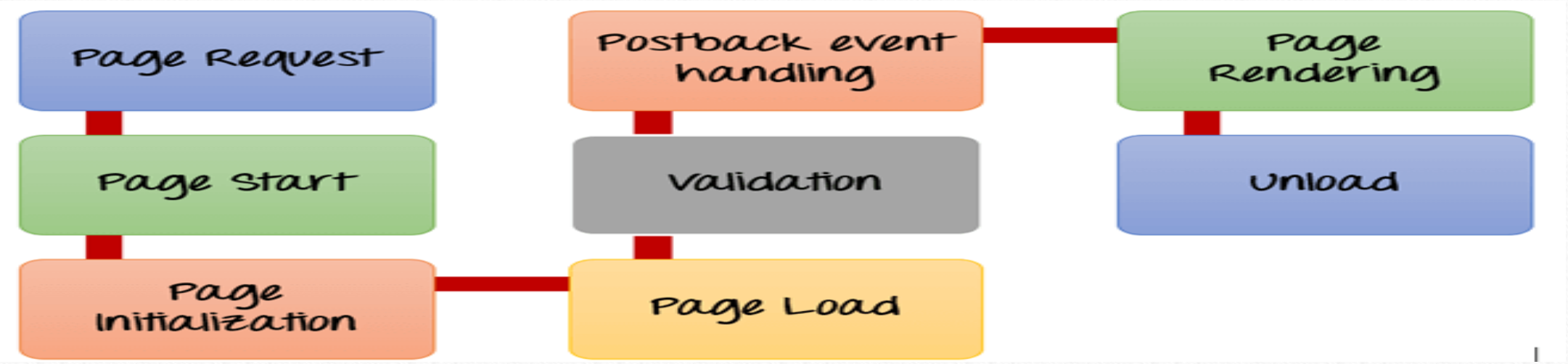
- **Abstraction**:-*Hiding internal details and showing functionality* is known as abstraction. For example phone call, we don't know the internal processing.
- **Inheritance:-***When one object acquires all the properties and behaviors of a parent object*, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.
- **Polymorphism**:-If *one task is performed in different ways*, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc.

1.6 Event Driven Programming



What is ASP.Net Page Lifecycle?

- ✓ When an ASP.Net page is called, it goes through a particular lifecycle. This is done before the response is sent to the user. There are series of steps which are followed for the processing of an ASP.Net page.
- ✓ Let's look at the various stages of the lifecycle of an ASP.Net web page.



Following are the different stages of an ASP.NET page:

•**Page request** - When ASP.NET gets a page request, it decides whether to parse and compile the page, or there would be a cached version of the page; accordingly the response is sent.

•**Starting of page life cycle** - At this stage, the Request and Response objects are set. If the request is an old request or post back, the IsPostBack property of the page is set to true. The UICulture property of the page is also set.

•**Page initialization** - At this stage, the controls on the page are assigned unique ID by setting the UniqueID property and the themes are applied. For a new request, postback data is loaded and the control properties are restored to the view-state values.

•**Page load** - At this stage, control properties are set using the view state and control state values.

•**Validation** - Validate method of the validation control is called and on its successful execution, the IsValid property of the page is set to true.

•**Postback event handling** - If the request is a postback (old request), the related event handler is invoked.

•**Page rendering** - At this stage, view state for the page and all controls are saved. The page calls the Render method for each control and the output of rendering is written to the OutputStream class of the Response property of page.

•**Unload** - The rendered page is sent to the client and page properties, such as Response and Request, are unloaded and all cleanup done.

ASP.NET Page Life Cycle Events

- At each stage of the page life cycle, the page raises some events, which could be coded. An event handler is basically a function or subroutine, bound to the event, using declarative attributes such as Onclick or handle.

1. PreInit:-

- 1.Check the IsPostBack property to determine whether this is the first time the page is being processed.
- 2.Create or re-create dynamic controls.
- 3.Set a master page dynamically.
- 4.Set the Theme property dynamically.

2. Init

1. This event fires after each control has been initialized.
2. Each control's UniqueID is set and any skin settings have been applied.
3. Use this event to read or initialize control properties.

3. Load

1. The Page object calls the OnLoad method on the Page object, and then recursively does the same for each child control until the page and all controls are loaded. The Load event of individual controls occurs after the Load event of the page.
2. Most code checks the value of IsPostBack to avoid unnecessarily resetting state.
3. You can also create dynamic controls in this method.
4. Use the OnLoad event method to set properties in controls and establish database connections.

4.Control PostBack Event(s)

1. ASP.NET now calls any events on the page or its controls that caused the PostBack to occur.
2. Use these events to handle specific control events, such as a Button control's Click event or a TextBox control's TextChanged event.
3. This is just an example of a control event. Here it is the button click event that caused the postback.

5. Render Method

1. The Render method generates the client-side HTML, Dynamic Hypertext Markup Language (DHTML), and script that are necessary to properly display a control at the browser.

6.UnLoad

1. This event is used for cleanup code.
2. At this point, all processing has occurred and it is safe to dispose of any remaining objects, including the Page object.
3. Cleanup can be performed on:
    - Instances of classes, in other words objects
    - Closing opened files
    - Closing database connections.
4. This event occurs for each control and then for the page.
5. During the unload stage, the page and its controls have been rendered, so you cannot make further changes to the response stream.
6. If you attempt to call a method such as the Response.Write method then the page will throw an exception.

Label control

Label control is used to place a static, non clickable (can't fire onclick event) piece of text on the page. When it is rendered on the page, it is implemented through HTML tag. Its properties like BackColor, ForeColor, BorderColor, BorderStyle, BorderWidth, Height etc. are implemented through style properites of . You can set its Text property either by setting Text properties in the .aspx page or from server side page. (other properties can also be set from both pages)

Following are few properties of the Label that are very useful.

EnableViewState	true/false. If false ViewState will not be maintained.
Visible	true/false. If false control will not be rendered to the page

DEMO : Label

Write something into the TextBox

Ex. Example of Label Control

```
// Label control code
<asp:Label ID="Label2" runat="server" BackColor="Coral"
ForeColor="blue" BorderColor="ActiveBorder"
BorderStyle="dashed" BorderWidth="1" Height="20"
Text="Example of Label Control" Width="200"
></asp:Label>
```

TextBox Control

TextBox control is used to enter data into the form that can be sent to the webserver by posting the form.

DEMO : TextBox

TextMode is Singleline Write some

TextMode is Multiline TextBox w ith TextMode as Multiline

PostBack the form It will Postback the page when cursor leaves this box.

Ex. TextBox value will be written here

```
// Singleline TextBox code
<asp:TextBox ID="TextBox1" runat="Server"
Width="300"></asp:TextBox>
```

Button control

Button control is generally used to post the form or fire an event either client side or server side. When it is rendered on the page, it is generally implemented through <input type=submit> HTML tag. However, if UserSubmitBehavior property is set to false then control will render out as <input type=button>.

Following are some important properties that are very useful.

UserSubmitBehavior	true/false. If true, the button will be used as client browser submit mechanism else asp.net postback mechanism.
CausesValidation	Value can be set as true/false. This indicates whether validation will be performed when a button is clicked.
PostBackUrl	Indicates the URL on which the Form will be posted back.
ValidationGroup	Gets or Sets the name of the validation group that the button belongs to. This is used to validate only a set of Form controls with a Button.
OnClick	Attach a server side method that will fire when button will be clicked.
OnClientClick	Attach a client side (javascript) event that will fire when button will be clicked.

LinkButton control

It implements an anchor <a/> tag that uses only ASP.NET postback mechanism to post the data on the server. Despite being a hyperlink, you can't specify the target URL. There is no UserSubmitBehavior property like Button control with LinkButton control.

Following are some important properties that are very useful.

CausesValidation	Value can be set as true/false. This indicates whether validation will be performed when a button is clicked.
PostBackUrl	Indicates the URL on which the Form will be posted back.
ValidationGroup	Gets or Sets the name of the validation group that the button belongs to. This is used to validate only a set of Form controls with a Button.
OnClick	Attach a server side method that will fire when button will be clicked.
OnClientClick	Attach a client side (javascript) method that will fire when button will be clicked.

ImageButton control

ImageButton control is generally used to post the form or fire an event either client side or server side. When it is rendered on the page, generally it is implemented through <input type=image > HTML tag.

Following are some important properties that are very useful.

ImageUrl	Gets or Sets the location of the image to display.
CausesValidation	Value can be set as true/false. This indicates whether validation should be performed when a button is clicked.
PostBackUrl	Indicates the URL on which the Form will be posted back.
ValidationGroup	Gets or Sets the name of the validation group that the button belongs to. This is used to validate only a set of Form controls with a Button.
OnClientClick	Attach a client side (javascript) method that will fire when button will be clicked.

OnClick	Attach a server side method that will fire when button will be clicked.

Hyperlink control

Hyperlink control is used to jump to another location or to execute the script code. When rendered on the page, it implements an anchor <a/> tag.
Following are some important properties that are useful.

NavigateUrl	Used to specify the location to jump to.
ImageUrl	Used to place an image instead of text as Hyperlink.

DropDownList control

DropDownList control is used to give a single select option to the user from multiple listed items.

You can specify its height and width in pixel by setting its height and width but you will not be able give mutliple select option to the user. When it is rendered on the page, it is implemented through <select/> HTML tag. It is also called as Combo box.
Following are some important properties that are very useful.

SelectedValue	Get the value of the Selected item from the dropdown box.
SelectedIndex	Gets or Sets the index of the selected item in the dropdown box.
SelectedItem	Gets the selected item from the list.
Items	Gets the collection of items from the dropdown box.
DataTextField	Name of the data source field to supply the text of the items. (No need to set when you are adding items directly into .aspx page.)
DataValueField	Name of the data source field to supply the value of the items. (No need to set when you are adding items directly into .aspx page.)
DataSourceID	ID of the datasource component to provide data. (Only used when you have any DataSource component on the page, like SqlDataSource, AccessDataSource etc.)
DataSource	The datasource that populates the items in the dropdown box. (Generally used when you are dynamically generating the items from Database.)
AutoPostBack	true or false. If true, the form is automatically posted back to the server when user changes the dropdown list selection. It will also fire OnSelectedIndexChanged method.
AppendDataBoundItems	true or false. If true, the statically added item (added from .aspx page) is maintained when adding items dynamically (from code behind file) or items are cleared.
OnSelectedIndexChanged	Method name that fires when user changes the selection of the dropdown box. (Fires only when AutoPostBack=true.)

```
<asp:DropDownList ID="DropDownList1" runat="server">
<asp:ListItem Text="Red" Value="red"></asp:ListItem>
<asp:ListItem Text="Blue" Value="blue"></asp:ListItem>
<asp:ListItem Text="Green" Value="green"></asp:ListItem>
</asp:DropDownList>
```

ListBox control

ListBox control is used to give a single or multiple select options to the user from multiple listed items.

All properties and its working resembles DropDownList box. However, ListBox has two extra properties called Rows and SelectionMode. ListBox control is used to give a single or multiple select option to the user (based on the property set) from multiple listed items. You can specify its height and width in pixel by setting its height and width but you will not be able give mutliple select option to the user. When it is rendered on the page, it is implemented through <select/> HTML tag. It is also called as Combo box.

You can add its option items by directly writing into .aspx page directly or dynamically add at run time or bind through database.

Following are some important properties that are very useful.

Rows	No. of rows (items) can be set to display in the List.
SelectionMode	Single or Multiple. If multiple, it allows user to select multiple items from the list by holding Ctrl or Shift key.
SelectedValue	Get the value of the Selected item from the dropdown box.
SelectedIndex	Gets or Sets the index of the selected item in the dropdown box.
SelectedItem	Gets the selected item from the list.
Items	Gets the collection of items from the dropdown box.
DataTextField	Name of the data source field to supply the text of the items. (No need to set when you are adding items directly into .aspx page.)
DataValueField	Name of the data source field to supply the value of the items. (No need to set when you are adding items directly into .aspx page.)
DataSourceID	ID of the datasource component to provide data. (Only used when you have any DataSource component on the page, like SqlDataSource, AccessDataSource etc.)
DataSource	The datasource that populates the items in the listbox box. (Generally used when you are dynamically generating the items from Database.)
AutoPostBack	true or false. If true, the form is automatically posted back to the server when user changes the dropdown list selection. It will also fire OnSelectedIndexChanged method.
AppendDataBoundItems	true or false. If true, the statically added item (added from .aspx page) is maintained when adding items dynamically (from code behind file) or items are cleared.
OnSelectedIndexChanged	Method name that fires when user changes the selection of the dropdown box. (Fires only when AutoPostBack=true.)

```
<asp:ListBox ID="ListBox1" runat="server">
                <asp:ListItem Text="Red" Value="red"></asp:ListItem>
                <asp:ListItem Text="Blue" Value="blue"></asp:ListItem>
                <asp:ListItem Text="Green" Value="green"></asp:ListItem>
</asp:ListBox>
```

CheckBox control

CheckBox control is used to give option to the user.

Following are some important properties that are very useful.

AutoPostBack	Form is automatically posted back when CheckBox is checked or Unchecked.
CausesValidation	true/false. If true, Form is validated if Validation control has been used in the form.
Checked	true/false. If true, Check box is checked by default.
OnCheckedChanged	Fires when CheckBox is checked or Unchecked. This works only if AutoPostBack property is set to true.
ValidationGroup	Used to put a checkbox under a particular validation group. It is used when you have many set of form controls and by clicking a paricular button you want to validate a particular set of controls only.

<asp:CheckBox ID="checkbox2" runat="Server" Text="Click, if Office address is same as Home address" AutoPostBack="True" OnCheckedChanged="PutHomeAddressAsOfficeAddress" BorderColor="brown" BorderWidth="1" CausesValidation="True" />

CheckBoxList control

CheckBoxList control is a single control that groups a collection of checkable list items, all are rendered through an individual <input type=checkbox></input>.

Following are some important properties that are very useful.

SelectedValue	Gets the value of first selected item.
SelectedIndex	Gets or Sets the index of the first selected item.
SelectedItem	Gets the first selected item
TextAlign	Gets or Sets the alignment of the checkbox text.
DataTextField	Name of the data source field to supply the text of the items. (No need to set when you are adding items directly into .aspx page.)
DataValueField	Name of the data source field to supply the value of the items. (No need to set when you are adding items directly into .aspx page.)
DataSourceID	ID of the datasource component to provide data. (Only used when you have any DataSource component on the page, like SqlDataSource, AccessDataSource etc.)
DataSource	The datasource that populates the items in the checkboxlist box. (Generally used when you are dynamically generating the items from Database.)
AutoPostBack	true/false. If true, the form is automatically posted back to the server when user click any of the checkbox. It will also fire OnSelectedIndexChanged method.
AppendDataBoundItems	true/false. If true, the statically added item (added from .aspx page) is maintained when adding items dynamically (from code behind file) or items are cleared.
OnSelectedIndexChanged	Method name that fires when user click any of the checkbox in the list. (Fires only when AutoPostBack=true.)

Items	Gets the colleciton of the items from the list.
RepeatLayout	table/flow. Gets or Sets the layout of the chekboxes when rendered to the page.
RepeatColumns	Gets or Sets the no. of columns to display when the control is rendered.
RepeatDirection	Horizontal/Vertical. Gets or Sets the the value to indicate whether the control will be rendered horizontally or vertically.

```
<asp:CheckBoxList ID="CheckBoxList1" runat="Server">
                <asp:ListItem Text="Red" Value="red"></asp:ListItem>
                <asp:ListItem Text="Blue" Value="blue"></asp:ListItem>
                <asp:ListItem Text="Green" Value="green"></asp:ListItem>
 </asp:CheckBoxList>
```

RadioButton control

RadioButton control is used to give single select option to the user from multiple items.
Following are some important properties that are very useful.

AutoPostBack	Form is automatically posted back when Radio button selection is changed.
CausesValidation	true/false. If true, Form is validated if Validation control has been used in the form.
Checked	true/false. If true, Radio button is selected by default.
OnCheckedChanged	Fires when Radio button selection changes. This works only if AutoPostBack property is set to true.
ValidationGroup	Used to put a radio button under a particular validation group. It is used when you have many set of form controls and by clicking a paricular button you want to validate a particular set of controls only.
GroupName	It is used a group a set of radion buttons so only one of them can be selected at a time.

```
<asp:RadioButton ID="RadioButton7" runat="Server" GroupName="1stGroup" Text="Red"
Checked="True"  />
<asp:RadioButton ID="Radio8" runat="Server" GroupName="1stGroup" Text="Blue"  />
```

RadioButtonList control

RadioButtonList control is a single control that groups a collection of radiobuttons, all are rendered through an individual <input type=radio></input>.
Following are some important properties that are very useful.
(RadioButtonList controls supports the same set of properties as the CheckBoxList control does.

SelectedValue	Get the value first selected item.
SelectedIndex	Gets or Sets the index of the first selected item.
SelectedItem	Gets the first selected item
TextAlign	Gets or Sets the alignment of the radiobutton text.
DataTextField	Name of the data source field to supply the text of the items. (No need to set when you are adding items directly into .aspx page.)
DataValueField	Name of the data source field to supply the value of the items. (No need to set when you are adding items directly into .aspx

	page.)
DataSourceID	ID of the datasource component to provide data. (Only used when you have any DataSource component on the page, like SqlDataSource, AccessDataSource etc.)
DataSource	The datasource that populates the items in the radiobuttonlist. (Generally used when you are dynamically generating the items from Database.)
AutoPostBack	true/false. If true, the form is automatically posted back to the server when user click any of the radiobutton. It will also fire OnSelectedIndexChanged method.
AppendDataBoundItems	true/false. If true, the statically added item (added from .aspx page) is maintained when adding items dynamically (from code behind file) or items are cleared.
OnSelectedIndexChanged	Method name that fires when user click any of the radiobutton in the list. (Fires only when AutoPostBack=true.)
Items	Gets the colleciton of the items from the list.
RepeatLayout	table/flow. Gets or Set the layout of the radiobuttons when rendered to the page.
RepeatColumns	Get or Sets the no. of columns to display when the control is rendered.
RepeatDirection	Horizontal/Vertical. Gets or Sets the the value to indicate whether the control will be rendered horizontally or vertically.

```
<asp:RadioButtonList ID="RadioButtonList1" runat="Server">
                <asp:ListItem Text="Red" Value="red"></asp:ListItem>
                <asp:ListItem Text="Blue" Value="blue"></asp:ListItem>
                <asp:ListItem Text="Green" Value="green"></asp:ListItem>
</asp:RadioButtonList>
```

Image control

Image control is used to place an image on the page.

Following are some important properties that are very useful.

ImageUrl	Url of image location.
AlternetText	Appears if image not loaded properly or if image is missing in the specified location.
Tooltip	Text message Appearing on mouse over the image
ImageAlign	Used to align the Text beside image.

```
<asp:Image ID="Image2" runat="Server" ImageUrl="~/images/Dot.gif" AlternateText="Dot
Logo"ImageAlign="textTop" ToolTip="Go to Dot Home page" />
```

ImageMap control

ImageMap control is used to create an image that contains clickable hotspot region.

Following are some important properties that are very useful.

ImageUrl	Url of image location.
AlternetText	Appears if image not loaded properly
Tooltip	Appears when on mouse over the image

ImageAlign	Used to align the Text beside image.
HotSpotMode	PostBack/Navigate .... When Navigate, the user is navigated to a different URL. In case of PostBack, the page is posted back to the server.
OnClick	Attach a server side event that fires after clicking on image when HostSpotMode is PostBack.
PostBackValue	You can access it in the server side click event through ImageMapEventArgs. (eg. e.PostBackValue)

```
<asp:ImageMap ID="ImageMap1" runat="Server" ImageUrl="controldata/gotocontrols.gif"
OnClick="FireImageMapClick">
 <asp:RectangleHotSpot AlternateText="Label" Left="10" Top="33" Right="75" Bottom="10"
NavigateUrl="~/tutorials/controls/label.aspx" />
 <asp:RectangleHotSpot AlternateText="Button" Left="80" Top="33" Right="150"
Bottom="10" NavigateUrl="~/tutorials/controls/button.aspx" />
 <asp:RectangleHotSpot AlternateText="ImageButton" Left="155" Top="33" Right="275"
Bottom="10" NavigateUrl="~/tutorials/controls/imagebutton.aspx" />
            </asp:ImageMap>
```

Asp: Table control

Table control is used to structure a web pages. In other words to divide a page into several rows and colums to arrange the information or images.
Table control is used to structure a web pages. In other words to divide a page into several rows and colums to arrange the information or images. When it is rendered on the page, it is implemented through <table> HTML tag.

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

We can simply use HTML <table> control instead of using asp:Table control. However many of one benefits of using asp:Table control is we can dynamically add rows or columns at the runtime or change the appearance of the table.
You can skip ID property of the TableRow or TableCell, however it is advisable to write these property otherwise you will not be able to play with these controls.

Following are some important properties that are very useful.

BackImageUrl	Used to Set background image of the table
Caption	Used to write the caption of the table.

```
<asp:Table ID="Table2" runat="Server" CellPadding="2" CellSpacing="1"
BorderColor="CadetBlue" Caption="Demo of asp:Table control" BorderWidth="1"
BorderStyle="Dashed">
<asp:TableRow ID="TableRow2" runat="Server" BorderWidth="1">
<asp:TableCell ID="TableCell4" runat="Server" BorderWidth="1">
Row 1 - Cell 1 </asp:TableCell>
<asp:TableCell ID="TableCell5" runat="Server">
Row 1 - Cell 2 </asp:TableCell> </asp:TableRow>
<asp:TableRow ID="TableRow3" runat="Server">
<asp:TableCell ID="TableCell6" runat="Server">
Row 2 - Cell 1 </asp:TableCell>
```

<asp:TableCell ID="TableCell7" runat="Server">
Row 2 - Cell 2 </asp:TableCell> </asp:TableRow> </asp:Table>

BulletedList control

BulletedList control is used to display the data in a list prefixed with bullet characters.
Following are some important properties that are very useful.

DisplayMode	HyperLink/LinkButton/Text. Determines how to display the items.
FirstBulletNumber	Sets a starting number for Bulleted list when BulletStyle is set to Numbering.
Items	Gets the colleciton of the items in the list control.
BulletStyle	Circle/CustomImage/Disc/LowerAlpha/LowerRoman/Numbered/Square/UpperAlpha/UpperRoman. Determines the style of the bullet.
AppendDataBoundItems	Determines whether statically defined items should remain and shown when adding items dynamically.
DataTextField	Name of the field to set as items text. Used when DisplayMode is Hyperlink or LinkButton.
DataValueField	Name of the field to set as items value. Used when DisplayMode is Hyperlink or LinkButton.
BulletImageUrl	Used to set the Bullet Image when BulletStyle is CustomImage.

```
<asp:BulletedList ID="BulletedList3" runat="Server" BorderColor="Blue" BorderWidth="1">
        <asp:ListItem Text="Item 1"></asp:ListItem>
        <asp:ListItem Text="Item 2"></asp:ListItem>
        <asp:ListItem Text="Item 3"></asp:ListItem
</asp:BulletedList>
```

Literal control

Literal control is the rarely used control which is used to put static text on the web page.
Ideally Literal control is the rarely used control which is used to put static text on the web page.
When it is rendered on the page, it is implemented just as a simple text.
Unlike asp:Label control, there is no property like BackColor, ForeColor, BorderColor, BorderStyle, BorderWidth, Height etc. of Literal control. That makes it more powerful, you can even put a pure
HTML contents into it.
Select color to change the background color the cell Ex. Just a text inside Literal Control

```
// CODE BEHIND
 // Fires when Button is clicked
protected void ChangeBackColor(object sender, EventArgs e)
{  Literal1.Text = " bgcolor='" + dropStatic.SelectedValue + "'";
litText.Text = "<div style='background-color:white;color:#000000'>Literl Control is powerful</div>";}
```

Calendar control

Calendar control is used to display one month calendar and allows to navigate backword & forward through dates, and months.

There are many properties of Calendar control to customize the functionality and appearance. However, these are some important properties that are very useful.

Properties	Description
Caption	Gets or sets the caption for the calendar control.
CaptionAlign	Gets or sets the alignment for the caption.
CellPadding	Gets or sets the number of spaces between the data and the cell border.
CellSpacing	Gets or sets the space between cells.
DayHeaderStyle	Gets the style properties for the section that displays the day of the week.
DayNameFormat	Gets or sets format of days of the week.
DayStyle	Gets the style properties for the days in the displayed month.
FirstDayOfWeek	Gets or sets the day of week to display in the first column.
NextMonthText	Gets or sets the text for next month navigation control. The default value is >.
NextPrevFormat	Gets or sets the format of the next and previous month navigation control.
OtherMonthDayStyle	Gets the style properties for the days on the Calendar control that are not in the displayed month.
PrevMonthText	Gets or sets the text for previous month navigation control. The default value is <.
SelectedDate	Gets or sets the selected date.
SelectedDates	Gets a collection of DateTime objects representing the selected dates.
SelectedDayStyle	Gets the style properties for the selected dates.
SelectionMode	Gets or sets the selection mode that specifies whether the user can select a single day, a week or an entire month.
SelectMonthText	Gets or sets the text for the month selection element in the selector column.
SelectorStyle	Gets the style properties for the week and month selector column.
SelectWeekText	Gets or sets the text displayed for the week selection element in the selector column.
ShowDayHeader	Gets or sets the value indicating whether the heading for the days of the week is displayed.
ShowGridLines	Gets or sets the value indicating whether the gridlines would be shown.
ShowNextPrevMonth	Gets or sets a value indicating whether next and previous month navigation elements are shown in the title section.
ShowTitle	Gets or sets a value indicating whether the title section is displayed.
TitleFormat	Gets or sets the format for the title section.
Titlestyle	Get the style properties of the title heading for the Calendar control.
TodayDayStyle	Gets the style properties for today's date on the Calendar control.
TodaysDate	Gets or sets the value for today's date.
UseAccessibleHeader	Gets or sets a value that indicates whether to render the table header <th> HTML element for the day headers instead of the table data <td> HTML element.
VisibleDate	Gets or sets the date that specifies the month to display.
WeekendDayStyle	Gets the style properties for the weekend dates on the Calendar

	control.

The Calendar control has the following three most important events that allow the developers to program the calendar control. They are:

Events	Description
SelectionChanged	It is raised when a day, a week or an entire month is selected.
DayRender	It is raised when each data cell of the calendar control is rendered.
VisibleMonthChanged	It is raised when user changes a month.

Panel control

Panel control is generally used to keep a set of controls into it.

Following are some important properties that are very useful.

GroupingText	Its used to set the caption of the group of controls inside the panel.
Visible	true/false. Used to hide or show the panel.

Login control

Login control provides a ready to use user interface that can be used as a Login interface in the web site.

Following are some important properties that are very useful.

Properties of the Login Control	
TitleText	Indicates the text to be displayed in the heading of the control.
InstructionText	Indicates the text that appears below the heading of the control.
UserNameLabelText	Indicates the label text of the username text box.
PasswordLabelText	Indicates the label text of the password text box.
FailureText	Indicates the text that is displayed after failure of login attempt.
UserName	Indicates the initial value in the username text box.
LoginButtonText	Indicates the text of the Login button.
LoginButtonType	Button/Link/Image. Indicates the type of login button.
DestinationPageUrl	Indicates the URL to be sent after login attempt successful.
DisplayRememberMe	true/false. Indicates whether to show Remember Me checkbox or not.
VisibleWhenLoggedIn	true/false. If false, the control is not displayed on the page when the user is logged in.
CreateUserUrl	Indicates the url of the create user page.
CreateUserText	Indicates the text of the create user link.
PasswordRecoveryUrl	Indicates the url of the password recovery page.
PasswordRecoveryText	Indicates the text of the password recovery link.
Style of the Login Control	
CheckBoxStyle	Indicates the style property of the Remember Me checkbox.
FailureStyle	Indicates the style property of the failure text.
TitleTextStyle	Indicates the style property of the title text.
LoginButtonStyle	Indicates the style property of the Login button.

TextBoxStyle	Indicates the style property of the TextBox.
LabelStyle	Indicates the style property of the labels of text box.
HyperLinkStyle	Indicates the style property of the hyperlink in the control.
InstructionTextStyle	Indicates the style property of the Instruction text that appears below the heading of the control.
Events of the Login Control	
LoggingIn	Fires before user is going to authenticate.
LoggedIn	Fires after user is authenticated.
LoginError	Fires after failure of login attempt.
Authenticate	Fires to authenticate the user. This is the function where you need to write your own code to validate the user.


```
// Login Control ////////////////////////////
<asp:Login ID="Login1" runat="server" BackColor="#F7F6F3" BorderColor="#E6E2D8"
BorderPadding="4"
   BorderStyle="Solid" BorderWidth="1px" Font-Names="Verdana" Font-Size="0.8em"
ForeColor="#333333" OnAuthenticate="Login1_Authenticate"
OnLoginError="Login1_LoginError">
   <TitleTextStyle BackColor="#5D7B9D" Font-Bold="True" Font-Size="0.9em"
ForeColor="White" />
    <LoginButtonStyle BackColor="#FFFBFF" BorderColor="#CCCCCC" BorderStyle="Solid"
BorderWidth="1px"
      Font-Names="Verdana" Font-Size="0.8em" ForeColor="#284775" />
</asp:Login>
```

LoginView control

LoginView control is very simple yet very powerful and customizable. It allows user to customize its view for both anonymous user and logged in user.

```
LoginView Control ////////////////////////////
<asp:LoginView ID="LoginView1" runat="Server">
   <AnonymousTemplate>
      <span style="font-family: Arial; font-size: 10pt;">Welcome, Guest
         <asp:LoginStatus ID="LoginStatus1" runat="Server" />
      </span>
   </AnonymousTemplate>
```

```
<LoggedInTemplate>
   Welcome,
   <asp:LoginName ID="LoginName1" runat="Server" />
   <asp:LoginStatus ID="LoginStatus1" runat="Server" />
</LoggedInTemplate>
```

File Upload Control

ASP.NET has two controls that allow users to upload files to the web server. Once the server receives the posted file data, the application can save it, check it, or ignore it. The following controls allow the file uploading:

- **HtmlInputFile** - an HTML server control
- **FileUpload** - and ASP.NET web control

Both controls allow file uploading, but the FileUpload control automatically sets the encoding of the form, whereas the HtmlInputFile does not do so.
In this tutorial, we use the FileUpload control. The FileUpload control allows the user to browse for and select the file to be uploaded, providing a browse button and a text box for entering the filename.

Once, the user has entered the filename in the text box by typing the name or browsing, the SaveAs method of the FileUpload control can be called to save the file to the disk.
The basic syntax of FileUpload is:

```
<asp:FileUpload ID= "Uploader" runat = "server" />
```

The FileUpload class is derived from the WebControl class, and inherits all its members.
Apart from those, the FileUpload class has the following read-only properties:

Properties	Description
FileBytes	Returns an array of the bytes in a file to be uploaded.
FileContent	Returns the stream object pointing to the file to be uploaded.
FileName	Returns the name of the file to be uploaded.
HasFile	Specifies whether the control has a file to upload.
PostedFile	Returns a reference to the uploaded file.

The posted file is encapsulated in an object of type HttpPostedFile, which could be accessed through the PostedFile property of the FileUpload class.
The HttpPostedFile class has the following frequently used properties:

Properties	Description
ContentLength	Returns the size of the uploaded file in bytes.
ContentType	Returns the MIME type of the uploaded file.
FileName	Returns the full filename.
InputStream	Returns a stream object pointing to the uploaded file.

For Example

```
Dim strname, strpath, strfullpath As String
    strname = ""
    If FileUpload1.HasFile Then
```

```
    strname = FileUpload1.FileName
    strpath = Server.MapPath("~/image/")
    strfullpath = strpath + strname
    FileUpload1.SaveAs(strfullpath)
End If
```

Request:

Information or message send by client to server is known as request.

The request object is an instance of the System.Web.Httprequest class.

This object represents the values and properties of the http request that cause your page to be loaded.

It contains all the URL parameters and all other information sent by a client.

Http request properties:

1. **Application path and Physical path:-**
   Application path gets the ASP.Net applications virtual directory (URL). While physical path gets the real directory.
2. **Browser:-**
   This provides a link to an http browser capabilities object which contains properties describing various browser features, such as supports for activates control, cookies, VB script and frames.
3. **Cookies:-**
   This gets the collection of cookies sent with this request.
4. **Form:-**
   This represents the collection of form variable that were posted back to the page. In almost all cases, you will retrieve this information from control properties instead of using this collection.
5. **IsLocal:-**
   This returns true, if the user is requesting the page from the current computer.
6. **Querystring:-**
   This provides the parameters that were passed along with the Querystring.
7. **URL and URL Reffer:-**
   This provides a URL object that represent the current address for the page and the page were the user is coming from (the previous page that link to this page)
8. **User Host address and User Host name:-**
   This get the IP address and the DNS name of the remote client.
   You could also access this information the server variables collection. However, this information may not always be available.

Response:

Information send by server to client is known as Response.

The response object is a instance of the system.web.httpresponse class and it represents the web server response to a client request.

The http response does till provide important functions namely cookie features and the redirect method. The redirect method allows you to send the user to another page.

Here is an example,

You can redirect to a file in the current directory Response.Redirect(“default2.aspx”)

You can redirect to other website Response.Redirect("http://www.google.com")

The Redirect() method requires a round-trip. Essentially, it sends a message to the browser that instructs it to request a news page.

If you want to transfer the user to another page in the same web application, you can use a faster approach with the Server.Transfer() method.

Http response members:

1. **Cookies:-**
   This is the collection of cookies send with the response. You can use this property to add additional cookies.
2. **IsClientConnected:-**
   This is a Boolean value indicating whether the client is still connected to the server. If it is not, you might want to stop a time consuming operation.
3. **Write(), BinaryWrite() and WriteFile():-**
   This method allows you to write the text or binary content directory to the response string. You can even write the content of a file.
4. **Redirect:-**
   This method transfers the user to another page in your application or a different website.

Server:

The server object is an instance of the System.Web.HttpServerUtility class.

Http server utility methods:

1. **MachineName:-**
   A property representing the computer name of the computer on which the page is running. This is the name of webserver computer. Uses to identify itself to rest of the network.
2. **GetLastError:-**
   Retrieves the exception object for the most recently encountered error, (all or a null reference if there is not one). This error must have occurred while processing the current request and it must not have been handled.

3. **HTML Encode and HTML Decode:-**
   Changes an ordinary string with a legal HTML characters.
4. **URL Encode and URL Decode:-**
   Changes an ordinary string into string with legal URL character.
5. **MapPath():-**
   Returns the physical file path the co-responds to specified virtual file path on the web server.
6. **Transfer():-**
   The transfer execution to another webpage in the current application. This is similar to Response.Redirect(). But, it is faster.

It cannot be used to transfer the page to a site on another web server or to a non ASP.Net page (such as an HTML page or an ASP page)

The transfer method is quickest to redirect user to another page in your application.When you use this method a round-trip is not involved. Instead the ASP.Net engine simply loads the new page and begins processing it.

As a result the URL i.e. displayed in the client browsers won't change.

You can transfer to a file in the current web application.

i.e. Server.Transfer("newpage.aspx")

You can't redirect to another website. This attempt will cause an error.

i.e. Server.Transfer("http://www.google.com")

The MapPath() is another useful method of the server object.

For e.g. Imagine you want to load a file name info.txt from the current virtual directory.

Instead of hard coding path, you can use Request.ApplicationPath to get the current relative virtual directory and Server.MapPath to convert this to an absolute physical path.

Here, is an example

Dim physicalpath as string

Physicalpath=Server.MapPath("~/data/info.txt")

Difference between Server.Transfer and Response.Redirect:

Response.Redirect	Server.Transfer
Response.Redirect involves a round-trip to the server.	Server.Transfer avoids the round-trip. It just changes the focus of the web server to different page and transforms the page processing to a different page.
Response.Redirect can be used for both .aspx and HTML pages.	Server.Transfer can be used only for .aspx page.
Response.Redirect can be used to redirect a user to an external website.	Server.Transfer can be used only on sites running on the same server. You can't use Server.Transfer to redirect the user to a page running on different server.
Response.Redirect changes the URL in the browser. So they can be bookmark.	Serever.Transfer retains the original URL in the browser. It just replaces the content of the previous page with new page.

HTML Server Control:

This are controls which are defined in the namespace System.Web.UI.HtmlControls

There are 20 different HTML server control. They are divided into different catagories based on whether they are input control or container control. Following diagram shows this

hierarchy.

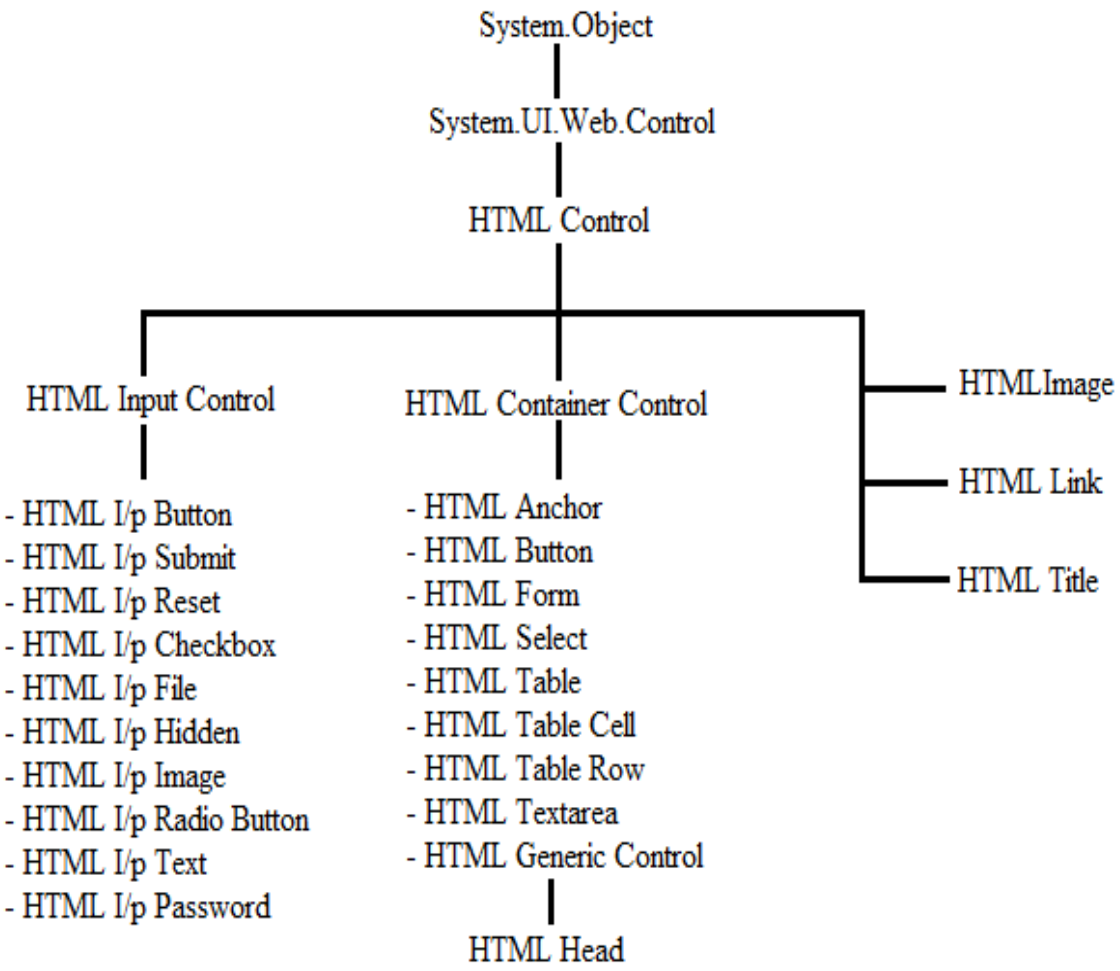


Fig: HTML Server Control

The HTML Control Class:

All the HTML server controls derives from the HTML base class HTML control. The following are set of common properties of HTML control class.

i. **Attribute:-**
   Allow to access or add attribute in the control tag.
ii. **Disabled:-**
   It sets or gets the control disabled state. If true then the control Is usually grayed and not usable.
iii. **Style:-**
   Returns a collection of CSS attributes that are applied to the control.
iv. **Tagname:-**
   Returns the control tag name.

The HTML Container Control Class:

Any HTML tag that has both an opening and closing tag can contain other HTML content or controls i.e. anchor tag <a> which usually wraps text or an image with the text.
<a>…..

There are other tag like …. which is also use as a container tag.

In addition to this we have bold tag. ….

In addition to this we can use this tag to map the HTML server control class by using the attribute runat=”server”.

In this case we can interact with this tag using the HTML generic control.

The following are the 2 main properties of HTML container control:-

i. **InnerHTML:-**
   Returns or sets the HTML tags inside the opening and closing tags. When you use the property, all characters are left as it is. This means you can embedded HTML markup.
ii. **InnerText:-**
   Returns or sets the text inside the opening and closing tags. When you use this property, any characters that would be interacted as special HTML syntax are automatically replaced with the HTML entity equivalents.

The HTML Input Control Class:

The HTML input control class allow for user interaction. It include checkboxes, textboxes, button and list boxes. The type attribute indicate the type of input control as in

<input type=”text”> (a textbox), <input type=”file”> (control for uploading file).

The HTML Input Control properties:

i. Name:-
   Gets the unique identifier name for the HTML input control.
ii. Type:-
   Gets or sets the type of an HTML input control. For e.g. If this property is set to text, the HTML input control is textbox for data entry.
iii. Value:-
   Gets or sets the value associated with input control.

The HTML Server Control Classes:

HTML server controls and the specific properties and events that each one adds to the base class.

Runat=”server” will allow to access particular HTML control at coding file.

HTML server control classes:-

Tag declaration	.Net class	Specific member
<a runat=”server”>	HTML anchor	HREF, target, title, name, server click event.
<button runat=”server”>	HTML Button	CausesValidation, ValidationGroup, Server click event.
<Form runat=”server”>	HTML Form	Name, method, target, DefaultButton, DefaultFocus
<img runat=”server”>	HTML Image	Align, alt, border, height, src,

		width.
<input type=”button” runat=”server”>	HTML input button	Name, type, value, CausesValidation, ValidationGroup, server click event.
<input type=”reset” runat=”server”>	HTML input reset	Name, type, value.
<input type=”submit” runat=”server”>	HTML input submit	Name, type, value, CausesValidation, ValidationGroup, server click event
<input type=”checkbox” runat=”server”>	HTML input checkbox	Check, type, name, value, server click event
<input type=”file” runat=”server”>	HTML input file	Accept,maxlength, name, posted file, size, type, value.
<input type=”hidden” runat=”server”>	HTML input hidden	Name, type, value, server change event.
<input type=”image” runat=”server”>	HTML inputimablege	Align, alt, border, name, src, type, value, CausesValidation, ValidationGroup, sercer click event
<input type=”radio” runat=”server”>	HTML input radio button	Check, type, name, value, server change event
<input type=”text” runat=”server”>	HTML input text	Maxlength, name, type, value, serverChange event
<input type=”password” runat=”server”>	HTML input password	Maxlength, name, type, value, serverChange event
<select runat=”server”>	HTML select	Multiple, selectedindex, size, value, datasource, datatextfield, datavaluefield, items(collection), server change event
<table runat=”server”>	HTML table	Align, bgcolor, border, border-color, cellpadding, cellspacing, height, nowrap, width, rows(colloction).
<th runat=”server”>	HTML table cell	Align, bgcolor, border, colspan, rowspan, nowrap, valign
<tr runat=”server”>	HTML table row	Align, bgcolor, height, valign, cells (collection)
<textarea runat=”server”>	HTML text area	Cols, name, rows, value, server change event.
Any other <html> with runat=”server” attribute	HTML generic control	None.

ImageMap:

ImageMap control is used to create an image that contains clickable hotspot region. When user click on the region, the user is either sent to a URL or a sub program is called. When it is rendered on the page, it is implemented through <img /> HTML tag.

Its properties like *BackColor, ForeColor, BorderColor, BorderStyle, BorderWidth, Height etc.* are implemented through style properites of <img>.

Following are some important properties that are very useful.

ImageUrl	Url of image location.
AlternetText	Appears if image not loaded properly
Tooltip	Appears when on mouse over the image
ImageAlign	Used to align the Text beside image.
HotSpotMode	PostBack/Navigate .... When Navigate, the user is navigated to a different URL. In case of PostBack, the page is posted back to the server.
OnClick	Attach a server side event that fires after clicking on image when HostSpotMode is PostBack.
PostBackValue	You can access it in the server side click event through ImageMapEventArgs. (eg. e.PostBackValue)

Clicking on

```
    <asp:ImageMap ID="ImageMap1" runat="Server"
ImageUrl="controldata/gotocontrols.gif" OnClick="FireImageMapClick">

                        <asp:RectangleHotSpot AlternateText="Label" Left="10"
Top="33" Right="75" Bottom="10" NavigateUrl="~/tutorials/controls/label.aspx" />

                         <asp:RectangleHotSpot AlternateText="Button" Left="80"
Top="33" Right="150" Bottom="10" NavigateUrl="~/tutorials/controls/button.aspx" />

                        <asp:RectangleHotSpot AlternateText="ImageButton"
Left="155" Top="33" Right="275" Bottom="10"
NavigateUrl="~/tutorials/controls/imagebutton.aspx" />

                        <asp:RectangleHotSpot AlternateText="Fires server side
Click Event. Postback value is ListBox" Left="300" Top="40" Right="400" Bottom="0"
NavigateUrl="~/tutorials/controls/listbox.aspx" HotSpotMode="PostBack"
PostBackValue="ListBox" /></asp:ImageMap>
```

Asp Table:

Table control is used to structure a web pages. In other words to divide a page into several rows and columns to arrange the information or images. When it is rendered on the page, it is implemented through <table> HTML tag.

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

We can simply use HTML <table> control instead of using asp:Table control. However many of one benefits of using asp:Table control is we can dynamically add rows or columns at the runtime or change the appearance of the table. You can skip ID property of the TableRow or TableCell, however it is advisable to write these property otherwise you will not be able to play with these controls.

Following are some important properties that are very useful.

BackImageUrl	Used to Set background image of the table
Caption	Used to write the caption of the table.
Demo of asp:Table control Row 1 - Cell 1 Row 1 - Cell 2 Row 2 - Cell 1 Row 2 - Cell 2	Add One Row and 2 Column Change Table Back Color

```
<asp:Table ID="Table2" runat="Server" CellPadding="2" CellSpacing="1"
        BorderColor="CadetBlue" Caption="Demo of asp:Table control" BorderWidth="1"
BorderStyle="Dashed">
            <asp:TableRow ID="TableRow2" runat="Server" BorderWidth="1">
                <asp:TableCell ID="TableCell4" runat="Server" BorderWidth="1">
                    Row 1 - Cell 1
                </asp:TableCell>
                <asp:TableCell ID="TableCell5" runat="Server">
                    Row 1 - Cell 2
                </asp:TableCell>
            </asp:TableRow>
            <asp:TableRow ID="TableRow3" runat="Server">
                <asp:TableCell ID="TableCell6" runat="Server">
                    Row 2 - Cell 1
                </asp:TableCell>
                <asp:TableCell ID="TableCell7" runat="Server">
                    Row 2 - Cell 2
                </asp:TableCell>
            </asp:TableRow>
        </asp:Table>
```

BulletedList :

BulletedList control is used to display the data in a list prefixed with bullet characters. The item can be statically written or can be bound with the datasource. When it is rendered on the page, it is implemented through <table> HTML tag.

Its properties like *BackColor, ForeColor, BorderColor, BorderStyle, BorderWidth, Height etc.* are implemented generally through style properites of <ul> tag, However it depends on BulletStyle property.

Following are some important properties that are very useful.

DisplayMode	HyperLink/LinkButton/Text. Determines how to display the items.
FirstBulletNumber	Sets a starting number for Bulleted list when BulletStyle is set to Numbering.
Items	Gets the colleciton of the items in the list control.
BulletStyle	Circle/CustomImage/Disc/LowerAlpha/LowerRoman/Numbered/Square/UpperAlpha/ UpperRoman. Determines the style of the bullet.

AppendDataBoundItems	Determines whether statically defined items should remain and shown when adding items dynamically.
DataTextField	Name of the field to set as items text. Used when DisplayMode is Hyperlink or LinkButton.
DataValueField	Name of the field to set as items value. Used when DisplayMode is Hyperlink or LinkButton.
BulletImageUrl	Used to set the Bullet Image when BulletStyle is CustomImage.

Literal:

Ideally Literal control is the rarely used control which is used to put static text on the web page. When it is rendered on the page, it is implemented just as a simple text. Unlike asp:Label control, there is no property like *BackColor, ForeColor, BorderColor, BorderStyle, BorderWidth, Height etc.* of Literal control. That makes it more powerful, you can even put a pure HTML contents into it.

Select color to change the background color the cell Red Change Background Color	Ex. Just a text inside Literal Control

```
                // Set the background color of the cell from server side
event
                <td> <asp:Literal ID="Literal2" runat="Server" />
                    Ex. <asp:Literal ID="Literal3" runat="Server" Text="Just
a text inside Literal Control"></asp:Literal>
                </td>
            </tr>

             // CODE BEHIND
            // Fires when Button is clicked

             Literal1.Text = " bgcolor='" + dropStatic.SelectedValue + "'";
 litText.Text = "<div style='background-color:white;color:#000000'>Literl
Control is powerful</div>";
```

Cascading Style Sheets (CSS)

Cascading Style Sheets (CSS) is a style sheet language used to describe the look and formatting of a document written in a markup language

CSS information can be provided by various sources. CSS style information can be either attached as a separate document or embedded in the HTML document. Multiple style sheets can be imported. Different styles can be applied depending on the output device being used.

Priority scheme for CSS sources (from highest to lowest priority):

- Author styles (provided by the web page author), in the form of:
    - Inline styles, inside the HTML document, style information on a single element, specified using the "style" attribute
    - Embedded style, blocks of CSS information inside the HTML itself
    - External style sheets, i.e., a separate CSS file referenced from the document
- User style:
    - A local CSS file the user specifies with a browser option, which acts as an override applied to all document.

The style sheet with the highest priority controls the content display. Declarations not set in the highest priority source are passed on by a source of lower priority such as the user agent style. This process is called *cascading*.

One of the goals of CSS is also to allow *users* greater control over presentation.

- <LINK : The HTML's standard link tag.
- REL="stylesheet" : The link type
- TYPE="text/css" : Advisory content type
- HREF="../CSS/Format.CSS"> : This is our most important element, this is the file name of our CSS file. The '../CSS' is not of particular meaning; it's just the name of the folder inside which our CSS file is stored and which can be anything or even nothing.

What are the commonly used methods of Dataadapter in ADO.NET?

Dataadapter has several methods associated with it.

Most commonly used methods among them are listed below:

• **Fill:** Fill method is used to fetch records from the database and update them into the datatables of dataset. Uses SelectCommand for execution. Syntax for Fill method is : sampleAdapter.Fill(employee,”Employee”);
Here sampleAdapter is a SqlDataAdapter containing select query for Employee, employee is the dataset and Employee is the database table.

• **FillSchema:** FillSchema method is used to create an empty table in dataset containing the same schema as that of a specific table in the database. Constraints of the corresponding database table is also copied and reflected in the datatable of dataset. Uses SelectCommand for execution but copies only the schema of the table and not the data. Syntax for this method is shown below:
sampleAdapter.FillSchema(empDataSet, SchemaType.Source, "Employee");
Here empDataSet is the dataset and Employee is the database table name.

• **Update:** Manipulated records of the dataset are updated back in the database using this method. Records that are inserted, updated and deleted from the dataset are pushed into the database using this method. Uses InsertCommand or UpdateCommand or DeleteCommand for the above mentioned purpose. Syntax for Update method is shown below:
sampleAdapter.Update(employeeTable);
Before this statement, sampleAdapter will include an UpdateCommand. employeeTable is the datatable of the dataset.

• **Dispose:** This method is used to release all resources used by the dataadapter. Here is the syntax:
sampleAdapter.Dispose();

Data Binding

ASP.NET adds a feature that allows you to pop data directly into HTML elements and fully formatted controls. It's called *data binding*.

Types of ASP.NET Data Binding

Two types of ASP.NET data binding exist: single-value binding and repeated-value binding.

Single-value data binding is by far the simpler of the two, whereas repeated-value binding provides the foundation for the most advanced ASP.NET data controls.

Single-Value, or "Simple," Data Binding

You can use *single-value data binding* to add information anywhere on an ASP.NET page. You can even place information into a control property or as plain text inside an HTML tag. Single-value data binding doesn't necessarily have anything to do with ADO.NET. Instead, single-value data binding allows you to take a variable, a property, or an expression and insert it dynamically into a page.

Single-value data binding is really just a different approach to dynamic text. To use it, you add special data binding expressions into your .aspx files. These expressions have the following
format:

<%# expression_goes_here %>

This may look like a script block, but it isn't. If you try to write any code inside this tag, you will receive an error. The only thing you can add is a valid data binding expression.

For example, if you have a public or protected variable named Country in your page, you could write the following:

<%# Country %>

When you call the DataBind() (me.databind()) method for the page, this text will be replaced with the value for Country (for example, Spain).

Repeated-Value, or "List," Binding

Repeated-value data binding allows you to display an entire table (or just a single field from a table). Unlike single-value data binding, this type of data binding requires a special control that supports it. Typically, this will be a list control such as CheckBoxList or ListBox, but it can also be a much more sophisticated control such as the GridView You'll know that a control supports repeated-value data binding if it provides a DataSource property. As with single-value binding, repeated value binding doesn't necessarily need to use data from a database, and it doesn't have to use the ADO.NET objects. For example, you can use repeated-value binding to bind data from a collection or an array.

Although using simple data binding is optional, repeated-value binding is so useful that almost every ASP.NET application will want to use it somewhere. Repeated-value data binding uses one of the special list controls included with ASP.NET. You link one of these controls to a data list source (such as a field in

a data table), and the control automatically creates a full list using all the corresponding values.

To create a data expression for list binding, you need to use a list control that explicitly supports data binding. Luckily, ASP.NET provides a whole collection, many of which you've probably already used in other applications or examples:
ListBox,DropDownList, CheckBoxList, and RadioButtonList: These web controls provide a list for a single-column of information.
GridView,DetailsView, and FormView: These rich web controls allow you to provide repeating lists or grids that can display more than one column (or field) of information
at a time.

How Data Binding Works
Data binding works a little differently depending on whether you're using single-value or repeated-value binding. In single-value binding, a data binding expression is inserted into the HTML markup in the .aspx file (not the code-behind file).

Once you specify data binding, you need to activate it. You accomplish this task by calling the DataBind() method. The DataBind() method is a basic piece of functionality supplied in the Control class. It automatically binds a control and any child controls that it contains.With repeated-value binding, you can use the DataBind() method of the specific list control you're using.

The Page Life Cycle with Data Binding
Data source controls can perform two key tasks:
• They can retrieve data from a data source and supply it to linked controls.
• They can update the data source when edits take place in linked controls.
To use the data source controls, you need to understand the page life cycle. The following
steps explain the sequence of stages your page goes through in its lifetime. The two steps in bold (4 and 6) are the steps where the data source controls will spring into action:
1. The page object is created (based on the .aspx file).
2. The page life cycle begins, and the Page.Init and Page.Load events fire.
3. All other control events fire.
4. The data source controls performing updates. If a row is being updated, the Updating and Updated events fire. If a row is being inserted, the Inserting and Inserted events fire. If a row is being deleted, the Deleting and Deleted events fire.
5. The Page.PreRender event fires.
6. The data source controls perform any queries and insert the retrieved data in the linked controls. The Selecting and Selected events fire at this point.
7. The page is rendered and disposed.

DataList

DataList control displays data using user-defined layout. However there are many added advantages in comparison with Repeater control in terms of graphical layout.

One of the main advantage of DataList control is it supports directional rendering (Horizontal/Vertical) also. It has many properties and several events attached. We can say DataList is the advanced version of Repeater control.

Following are some important properties that are very useful.

AlternatingItemTemplate	Template to define the rendering of every alternate item.
FooterTemplate	Template to define how to render the footer.
HeaderTemplate	Template to define how to render the header.
Items	Gets the collection of DataList Items.
ItemTemplate	Template to define how items are rendered.
SeparatorTemplate	Template to define how separator between items will be rendered.

DEMO : DataList

Name : jjh Address : jhjh Phone : jhhjj City : jjkjk	|||	Name : MallaReddy Address : Hyd Phone : 12345 City : Hyd	|||
Name : mkmk Address : ji Phone : eee City : eee	|||	Name : mndsam Address : dmsna Phone : mndsa City : msna	|||
Name : name Address : home Phone : 7006 City :	|||	Name : qqqq Address : 1223 Phone : 115 City : 14545	|||

```
// DataList Control ////////////////////////////
<asp:DataList ID="DataList1" runat="Server"
DataSourceID="SqlDataSource1" DataKeyField="AutoID" Width="100%"
     RepeatColumns="2" RepeatDirection="horizontal"
RepeatLayout="table" CellPadding="2" CellSpacing="1"
     BorderWidth="1">
    <ItemTemplate>
        <table width="100%" style="background-color:#efefef;">
            <tr>
                <td>
                    Name : <%# Eval("Name") %><br />
                    Address : <%# Eval("Address") %><br />
                    Phone : <%# Eval("Phone") %><br />
                    City : <%# Eval("City") %><br />
                </td>
            </tr>
        </table>
    </ItemTemplate>
```

```
    <AlternatingItemTemplate>
        <table width="100%">
            <tr>
                <td>
                    Name : <%# Eval("Name") %><br />
                    Address : <%# Eval("Address") %><br />
                    Phone : <%# Eval("Phone") %><br />
                    City : <%# Eval("City")% ><br />
                </td>
            </tr>
        </table>
    </AlternatingItemTemplate>
    <SeparatorTemplate>
        |||
    </SeparatorTemplate>
</asp:DataList>



// SqlDataSource Control ////////////////////////////
 <asp:SqlDataSource ID="SqlDataSource1" runat="server"
ConnectionString='<%$ ConnectionStrings:ConnStr %>'
 SelectCommand="Select * FROM emp ORDER BY [Name]">
 </asp:SqlDataSource>
```

The DataSet Class

The dataset represents a subset of the database. It does not have a continuous connection to the database. To update the database a reconnection is required. The DataSet contains DataTable objects and DataRelation objects. The DataRelation objects represent the relationship between two tables.

Following table shows some important properties of the DataSet class:

Properties	Description
CaseSensitive	Indicates whether string comparisons within the data tables are case-sensitive.
IsInitialized	Indicates whether the DataSet is initialized.
Relations	Returns the collection of DataRelation objects.
Tables	Returns the collection of DataTable objects.

The following table shows some important methods of the DataSet class:

Methods	Description
AcceptChanges	Accepts all changes made since the DataSet was loaded or this method was called.
BeginInit	Begins the initialization of the DataSet. The initialization occurs at run time.
Clear	Clears data.
Clone	Copies the structure of the DataSet, including all DataTable schemas, relations, and constraints. Does not copy any data.
Copy	Copies both structure and data.
EndInit	Ends the initialization of the data set.
Equals(Object)	Determines whether the specified Object is equal to the current Object.
Finalize	Free resources and perform other cleanups.
GetChanges	Returns a copy of the DataSet with all changes made since it was loaded or the AcceptChanges method was called.
GetChanges(DataRowState)	Gets a copy of DataSet with all changes made since it was loaded or the AcceptChanges method was called, filtered by DataRowState.
GetDataSetSchema	Gets a copy of XmlSchemaSet for the DataSet.
GetObjectData	Populates a serialization information object with the data needed to serialize the DataSet.

GetType	Gets the type of the current instance.
GetXML	Returns the XML representation of the data.
GetXMLSchema	Returns the XSD schema for the XML representation of the data.
HasChanges()	Gets a value indicating whether the DataSet has changes, including new, deleted, or modified rows.
Merge()	Merges the data with data from another DataSet. This method has different overloaded forms.
ReadXML()	Reads an XML schema and data into the DataSet. This method has different overloaded forms.
ReadXMLSchema(0)	Reads an XML schema into the DataSet. This method has different overloaded forms.
RejectChanges	Rolls back all changes made since the last call to AcceptChanges.
WriteXML()	Writes an XML schema and data from the DataSet. This method has different overloaded forms.
WriteXMLSchema()	Writes the structure of the DataSet as an XML schema. This method has different overloaded forms.

DataSet Vs DataReader
or
Connectionless object and Connection Oriented Object

Asp.net developer uses **DataSet** and **DataReader** to fetch data from the data source while developing asp.net application. But most of them **don't** know exactly what are the main difference between **DataSet** and**DataReader** and what to use and when to use out of these two.

Both **DataSet** and **DataReader** are widely used in asp.net applications for the same purpose i.e. to get/fetch the data from the database. But one has to know the best practices in developing fast, reliable and scalable application. The **DataSet** and **DataReader** which are as follows:

DataSet Vs DataReader

DataReader	Dataset
DataReader is Connection Oriented object.	Data SET is connectionless object
DataReader is used to retrieve read-only (cannot update/manipulate data back to datasource) and forward-only (cannot read backward/random) data from a database.	Dataset is used to manipulate data
DtaReader is like a forward only recordset. It fetches one row at a time so very less network cost compare to **DataSet**	**DataSet** which fetches all the rows at a time i.e. it fetches all data from the datasource at a time to its memory area.
As one row at a time is stored in memory in **DataReader** it increases application performance and reduces system overheads while there is more system overheads in **DataSet** .	**DataSet** as it fetches all the data from the datasource at a time in memory so it has more system overhead.
As **DataReader** is forward only, we can't fetch random records as we can't move back and forward	In **DataSet** we can move back and forward and fetch records randomly as per requirement.
DataReader fetches data from a single table	**DataSet** can fetch more the one table in it.
As **DataReader** can have data from a single table so no relationship can be maintained.	While relationship between multiple tables can be maintained in **DataSet**.
DataReader is read only so no transaction like insert, update and delete is possible	While inert, update, delete transactions are possible in **DataSet**.
DataReader is require small memory compare dataset object	**DataSet** is a bulky object that requires lot of memory space as compared to **DataReader**

DataReader is a connected architecture: The data is available as long as the connection with database exists	while **DataSet** is a disconnected architecture that automatically opens the connection, fetches the data into memory and closes the connection when done.
DataReader requires connection to be open and close manually in code	While DataSet automatically handles it.
DataReader can't be serialized so we can not store in Session.	**DataSet** can be serialized and represented in XML so it can easily store in session.
DataReader will be the best choice where we need to show the data to the user which requires no manipulation.	While **DataSet** is best suited where there is possibility of manipulation on the data.
DataReader can only be read once so it can be bound to a single control and requires data to be retrieved for each control.	When you need to navigate through the data multiple times then **DataSet** is better choice e.g. we can fill data in multiple controls

DataTable

Introduction

DataTable is a central object in the ADO.NET library. If you are working with ADO.NET - accessing data from database, you can not escape from DataTable. Other objects that use DataTable are DataSet and DataView. In this tutorials, I will explain how to work with DataTable. I have tried to cover most of the frequently used activity in the DataTable, I hope you will like it.

Creating a DataTable

To create a DataTable, you need to use System.Data namespace, generally when you create a new class or page, it is included by default by the Visual Studio. Lets write following code to create a DataTable object. Here, I have pased a string as the DataTable name while creating DataTable object.

```
// instantiate DataTable
Dim dTable As New DataTable("Emp")
```

Creating Columns in the DataTable

To create column in the DataTable, you need to use DataColumn object. Instantiate the DataColumn object and pass column name and its data type as parameter. Then call add method of DataTable column and pass the DataColumn object as parameter.

```
' create columns for the DataTable

Dim auto As New DataColumn("AutoID", GetType(System.Int32))

dTable.Columns.Add(auto)

' create another column

Dim name As New DataColumn("Name", GetType(String))

dTable.Columns.Add(name)

' create one more column

Dim address As New DataColumn("Address", GetType(String))

dTable.Columns.Add(address)
```

Using DataRow object

Look at the code below, I have created a DataRow object above the loop and I am assiging its value to the dTable.NewRow() inside the loop. After specifying columns value, I am adding that row to the DataTable using dTable.Rows.Add method.

```
' populate the DataTable using DataRow object
Dim row As DataRow = Nothing
For i As Integer = 0 To 4
      row = dTable.NewRow()
      row("AutoID") = i + 1
      row("Name") = i & " - Ram"
      row("Address") = "Ram Nagar, India - " & i
      dTable.Rows.Add(row)
Next
```

Properties

Name	Description
CaseSensitive	Indicates whether string comparisons within the table are case-sensitive.
ChildRelations	Gets the collection of child relations for this **DataTable**.
Columns	Gets the collection of columns that belong to this table.
Constraints	Gets the collection of constraints maintained by this table.
DataSet	Gets the **DataSet** to which this table belongs.
DefaultView	Gets a customized view of the table that may include a filtered view, or a cursor position.
IsInitialized	Gets a value that indicates whether the **DataTable** is initialized.
ParentRelations	Gets the collection of parent relations for this **DataTable**.
PrimaryKey	Gets or sets an array of columns that function as primary keys for the data table.
Rows	Gets the collection of rows that belong to this table.
TableName	Gets or sets the name of the **DataTable**.

Methods

Name	Description
AcceptChanges	Commits all the changes made to this table since the last time AcceptChanges was called.
BeginInit	Begins the initialization of a **DataTable** that is used on a form or used by another component. The initialization occurs at runtime.
BeginLoadData	Turns off notifications, index maintenance, and constraints while loading data.
Clear	Clears the **DataTable** of all data.
Clone	Clones the structure of the **DataTable**, including all **DataTable** schemas and constraints.
Copy	Copies both the structure and data for this **DataTable**.
Dispose	Overloaded. Releases the resources used
EndInit	Ends the initialization of a **DataTable** that is used on a form or used by another component. The initialization occurs at runtime.
Equals	Overloaded. Determines whether two Object instances are equal.
GetChanges	Overloaded. Gets a copy of the **DataTable** containing all changes made to it since it was last loaded, or since **AcceptChanges** was called.
GetType	Gets the Type of the current instance.
Merge	Overloaded. Merge the specified **DataTable** with the current **DataTable**.
NewRow	Creates a new **DataRow** with the same schema as the table.
ReadXml	Overloaded. Reads XML schema and data into the **DataTable**.
RejectChanges	Rolls back all changes that have been made to the table since it was loaded, or the last time **AcceptChanges** was called.
Reset	Resets the **DataTable** to its original state.
Select	Overloaded. Gets an array of **DataRow** objects.
WriteXml	Overloaded. Writes the current contents of the **DataTable** as XML.

DetailsView

DetailsView control is a data-bound control that renders a single record at a time. It can provide navigation option also. It can insert, update and delete the record also.

Following are some important properties that are very useful.

Behavior Properties of the DetailsView Control	
AllowPaging	true/false. Indicate whether the control should support navigation.
DataSource	Gets or sets the data source object that contains the data to populate the control.
DataSourceID	Indicate the bound data source control to use
AutoGenerateEditButton	true/false. Indicates whether a separate column with edit link/button should be added to edit the record.
AutoGenerateDeleteButton	true/false. Indicates whether a separate column with delete link/button should be added to delete the record.
AutoGenerateRows	true/false. Indicate whether rows are automatically created for each field of the data source. The default is true.
DefaultMode	read-only/insert/edit. Indicate the default display mode.
Style Properties of the DetailsView Control	
AlternatingRowStyle	Defines the style properties for every alternate row in the DetailsView.
EditRowStyle	Defines the style properties for the row in EditView (When you click Edit button for a row, the row will appear in this style).
RowStyle	Defines the style properties of the rows of the DetailsView.
PagerStyle	Defines the style properties of Pager of the DetailsView. (If AllowPaging=true, the page number row appears in this style)
EmptyDataRowStyle	Defines the style properties of the empty row, which appears if there is no records in the data source.
HeaderStyle	Defines the style properties of the header of the DetailsView. (The column header appears in this style.)
FooterStyle	Defines the style properties of the footer of DetailsView.

Appearance Properties of the DetailsView Control	
CellPadding	Indicates the amount of space in pixel between the cells and the border of the DetailsView.
CellSpacing	Indicates the amount of space in pixel between cells.
GridLines	Both/Horizontal/Vertical/None. Indicates whether GrdiLines should appear or not, if yes Horizontal, Vertical or Both.
HorizontalAlign	Indicates the horizontal alignment of the DetailsView.
EmptyDataText	Indicates the text to appear when there is no record in the data source.
BackImageUrl	Indicates the location of the image that should display as a background of the DetailsView.
Caption	Gets or sets the caption of the DetailsView.
CaptionAlign	left/center/right. Gets or sets the horizontal position of the DetailsView caption.
State Properties of DetailsView Control	
Rows	Gets the collection of objects that represent the rows in the DetailsView.
FooterRow	Returns a DetailsViewRow object that represents the footer of the DetailsView.
HeaderRow	Returns a DetailsViewRow object that represents the header of the DetailsView.
PageCount	Gets the number of the pages required to display the records of the data source.
PageIndex	Gets or sets the 0-based page index.
DataKeyNames	Gets an array that contains the names of the primary key field of the currently displayed rows in the DetailsViewRow.
DataKeys	Gets a collection of DataKey objects that represent the value of the primary key fields set in DataKeyNames property of the DetailsViewRow.
Events of the DetailsView Control	
ItemCommand	Fires when any clickable element on the control is clicked.
ItemCreated	Fires after DetailsView fully creates all rows of the record.
ItemDeleting, ItemDeleted	Both event fires when current record is deleted. The first one fires before and other fires after record is deleted.
ItemInserting, ItemInserted	Both event fires when an item is inserted. The first one fires before and second after the item is created.

ItemUpdating, ItemUpdated	Both event fires when an item is updated. The first one fires before and second fires after the record is updated.
ModeChanging, ModeChanged	Both event fires when DetailsView change its display mode. The first one fires before and second fires after display mode is changed.
PageIndexChanging, PageIndexChanged	Both event fires when the DetailsView move to another record. The first one fires before and second fires after page is changed.

Syntax for Details View

```
<asp:DetailsView ID="DetailsView1" runat="server" AllowPaging="True"
            AutoGenerateRows="False" >
            <Fields>
```

[Bound columns]

```
</Fields>
        </asp:DetailsView>
```

Difference between Details view and form view

the DetailsView and the FormView. Both show a single record at a time but can include optional pager buttons that let you step through a series of records (showing one per page). The difference between the DetailsView and the FormView is their support for templates. The DetailsView is built from field objects, in the same way the GridView is built from column objects. On the other hand, the FormView is based on templates that work in the same way as a GridView template column, which requires a little more work but gives you much more flexibility. Now that you understand the features of the GridView, you can get up to speed with the DetailsView and the FormView quite quickly. That's because both borrow a portion of the GridView model.

What is Exception?
An exception is unexpected error or problem.

What is Exception Handling?
Method to handle error and solution to recover from it, so that program works smoothly.

Error Handling

Visual Basic .NET supports two different ways to keep an unexpected error from terminating an application: Unstructured Error Handling and Structured Error Handling. These events are run-time errors, also called exceptions.

Unstructured Error Handling

Unstructured Error Handling is implemented with the **On Error** statement, which is placed at the beginning of a code block to handle all possible exceptions that occur during the execution of the code. You should place only one **On Error** statement in each procedure, because additional statements disable all previous handlers that are defined in that procedure.

On Error Statement

The **On Error** statement is used to enable an error-handling routine, disable an error handling routine, or specify where to branch the code in the event of an error.
On Error { GoTo [line | 0 | -1] | Resume Next }

GoTo *line*

Used to enable the error-handling routine, starting at the location that is specified by the ***line*** argument. The ***line*** argument can be either a line label or a line number that is located within the closing procedure.

To avoid unexpected behavior, place an **Exit Sub** statement, an **Exit Function** statement, or an **Exit Property** statement just before the line label or line number. This prevents the error-handling code from running when no error has occurred.
GoTo 0

Disables the enabled error handler that is defined within the current procedure and resets it to **Nothing**.

GoTo -1

Disables the enabled exception that is defined within the current procedure and resets it to **Nothing**.

Resume Next

Moves the control of execution to the statement that follows immediately after the statement that caused the run-time error to occur, and continues the execution

from this point forward. This is the preferred form to use to access objects, rather than using the **On Error GoTo** statement.

Structured Error Handling

Structured Error Handling also provides the programmers with a simpler way to create robust applications that are easier to maintain.

Structured error handling is implemented in Visual Basic. NET with a **Try...Catch...Finally** block of statements.

- **Try**: A Try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more Catch blocks.
- **Catch**: A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The Catch keyword indicates the catching of an exception.
- **Finally**: The Finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.
- **Throw**: A program throws an exception when a problem shows up. This is done using a Throw keyword.

Try...Catch...Finally Statements

The following code demonstrates the structure of a **Try...Catch...Finally** statement.

```
 Try
    'do something
Catch [Optional Filters]
    'The code runs if any of the statements
    'in the Try block fails and filter is evaluated as true.
   [Additional Catch Blocks]
Finally
    'Code executed after Try and Catch statement.
End Try
```

For Example

```
  Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

      Try
         Dim int1 As Integer
              Dim int2 As Integer = 10
         int1 = int2 / 0
      Catch ex As Exception
         MsgBox(ex.Message)
      Finally
         MsgBox("Exception is handled")
      End Try
```

```
End Sub
```

Introduction

When errors occur in an ASP.NET application, they either get handled or propagates unhandled to higher scopes. When an unhandled exception propagates, the user may be redirected to an error page using different ASP.NET configuration settings.

Error handling in ASP.NET therefore, may be divided into two separate logics:

- Redirecting the user to an error page when errors go unhandled.
- Handling exceptions when they get thrown.

Redirecting the user to an error page

There are two different scopes where we could specify which page the user should be redirected to, when errors go unhandled:

- Page level (applies to errors that happen within a single page).
- Application level (applies to errors that happen anywhere in the application).

Page Level Error Handling

Two facets:

1. Page redirection
2. Using the Page objects error event to capture exceptions

Taking each in turn:

Page redirection

Unforeseen errors can be trapped with the ErrorPage property of the Page object. This allows definition of a redirection URL in case of unhandled exceptions. A second step is required to enable such error trapping however – setting of the customErrors attribute of your web.config file, as follows:

```
<configuration>
  <system.web>
    <customErrors mode="On">
    </customErrors>
  </system.web>
</configuration>
```

Then you'll get your page level redirection rather than the page itself returning an error, so the following would work:

```
<%@ Page ErrorPage="customErrors.aspx" %>
```

A useful option in addition to ‘on’ and ‘off’ for the mode attribute of customErrors is ‘RemoteOnly’ as when specified redirection will only occur if the browser application is running on a remote computer. This allows those with access to the local computer to continue to see the actual errors raised.

Application Level Error Handling

In reality you are more likely to use application level error handling rather than the page level just introduced. The Application_Error event of the global.asax exists for this purpose. Unsurprisingly, this is fired when an exception occurs in the corresponding web application.

In the global.asax you code against the event as follows:

```
Sub Application_Error(sender as Object, e as EventArgs)
  'Do Something
End Sub
```

Unfortunately the EventArgs in this instance are unlikely to be sufficiently informative but there are alternative avenues including that introduced in the code of the last section – the HttpServerUtility.GetLastError method which returns a reference to the last error thrown in the application. This can be used as follows:

```
Sub Application_Error(sender as Object, e as EventArgs)

  dim LastException as string = Server.GetLastError().ToString()

  Context.ClearError()

  Response.Write(LastException)

End Sub
```

Note that the ClearError method of the Context class clears all exceptions from the current request – if you don’t clear it the normal exception processing and consequent presentation will still occur.

Alternatively there is the HttpContext’s class Error property which returns a reference to the first exception thrown for the current HTTP request/ response. An example:

```
Sub Application_Error(sender as Object, e as EventArgs)

  dim LastException as string = Context.Error.ToString()

  Context.ClearError()

  Response.Redirect("CustomErrors.aspx”)

End Sub
```

This illustrates one method for handling application errors – redirection to a custom error page which can then customise the output to the user dependent on the actual error received.

customErrors

we also have the ability to implement an application wide error page redirect, via the customErrors section of web.config already introduced, using the defaultRedirect property:

```
 <configuration>
   <system.web>
     <customErrors mode="on"
defaultRedirect="customerrors.aspx?err=Unspecified">
     <error statusCode="404" redirect="customerrors.aspx?err=File+Not+Found"/>
     </customErrors>
   </system.web>
 </configuration>
```

The mode attribute specifies whether to show user-defined custom error pages or ASP.NET error pages. Three values are supported for this attribute:

- **RemoteOnly** - Custom error pages are shown for all remote users. ASP.NET error pages with rich error information are displayed only for local users.
- **On** - Custom error pages are always shown, unless one is not specified. When a custom error page is not defined, an ASP.NET error page will be displayed which describes how to enable remote viewing of errors.
- **Off** - Custom error pages are not shown. Instead, ASP.NET error pages will be displayed always, which will have rich error information.

It's a bad idea to give users more information than what is required. ASP.NET error pages describe technical details that shouldn't be exposed. Ideally, the mode attribute thus should not be set to Off.

The defaultRedirect attribute specifies the path to a generic error page. This page would typically have a link to let the user go back to the home page or perform the request once again.

Each error element defines a redirect specific to a particular HTTP status code. For example, if the error is a 404 (File Not Found), then you could set the error page as *FileNotFound.htm*. You could add as many error elements in the customErrors section as required, each of which specifies a status code and the corresponding error page path. If ASP.NET can’t find any specific error element corresponding to a status code, it would use the value specified in the defaultRedirect attribute.

FormView

FormView is a new data-bound control that is nothing but a templated version of DetailsView control. The major difference between DetailsView and FormView is, here user need to define the rendering template for each item.
Following are some important properties that are very useful.

Templates of the FormView Control	
EditItemTemplate	The template that is used when a record is being edited.
InsertItemTemplate	The template that is used when a record is being created.
ItemTemplate	The template that is used to render the record to display only.
Methods of the FormView Control	
ChangeMode	ReadOnly/Insert/Edit. Change the working mode of the control from the current to the defined FormViewMode type.
InsertItem	Used to insert the record into database. This method must be called when the DetailsView control is in insert mode.
UpdateItem	Used to update the current record into database. This method must be called when DetailsView control is in edit mode.
DeleteItem	Used to delete the current record from database.

Try Inserting Records into Database

AutoID	
Name	
Address	
Phone	
City	

```
 // FormView control ////////////////////////////////
 <asp:FormView ID="FormView1" runat="server" CellPadding="4"
ForeColor="#333333"
     DataKeyNames="AutoID" DataSourceID="SqlDataSource1"
AllowPaging="true">
        <FooterStyle BackColor="#507CD1" Font-Bold="True"
ForeColor="White" />
        <RowStyle BackColor="#EFF3FB" />
        <PagerStyle BackColor="#2461BF" ForeColor="White"
HorizontalAlign="Center" />
        <HeaderStyle BackColor="#507CD1" Font-Bold="True"
ForeColor="White" />
        <ItemTemplate>
            <table border="1">
                <tr>
                    <td>AutoID</td>
                    <td><%# Eval("AutoID") %></td>
                </tr>
                <tr>
                    <td>Name</td>
                    <td><%# Eval("Name") %></td>
                </tr>
                <tr>
                    <td> </td>
                    <td>
                        <asp:Button ID="btnEdit" runat="Server"
CommandName="Edit" Text="Edit" />
                        <asp:Button ID="btnInsert" runat="Server"
CommandName="New" Text="New" />
```

```
                        <asp:Button ID="btnDelete" runat="Server"
CommandName="Delete" Text="Delete" OnClientClick="return confirm('Are
you sure to Delete?');" />
                    </td>
                </tr>
            </table>
        </ItemTemplate>
  </asp:SqlDataSource>
```

Explain the Global.asax file in ASP.NET

The Global.asax, also known as the ASP.NET application file, is located in the root directory of an ASP.NET application. This file contains code that is executed in response to application-level and session-level events raised by ASP.NET or by HTTP modules. You can also define 'objects' with application-wide or session-wide scope in the Global.asax file. These events and objects declared in the Global.asax are applied to all resources in that web application.

Note 1: The Global.asax is an optional file. Use it only when there is a need for it.

Note 2: If a user requests the Global.asax file, the request is rejected. External users cannot view the file.

The Global.asax file is parsed and dynamically compiled by ASP.NET. You can deploy this file as an assembly in the \bin directory of an ASP.NET application.

How to create Global.asax

Adding a Global.asax to your web project is quiet simple.

Open Visual Studio 2005 or 2008 > Create a new website > Go to the Solution Explorer > Add New Item > Global Application Class > Add.

Examining the methods related to the events in Global.asax

There are 2 'set' of methods that fire corresponding to the events. The first set which gets invoked on each request and the second set which does not get invoked on each request. Let us explore these methods.

Methods corresponding to events that fire on each request

Application_BeginRequest() – fired when a request for the web application comes in.

Application_AuthenticateRequest() –fired just before the user credentials are authenticated. You can specify your own authentication logic over here.

Application_AuthorizeRequest() – fired on successful authentication of user's credentials. You can use this method to give authorization rights to user.

Application_ResolveRequestCache() – fired on successful completion of an authorization request.

Application_AcquireRequestState() – fired just before the session state is retrieved for the current request.

Application_PreRequestHandlerExecute() - fired before the page framework begins before executing an event handler to handle the request.

Application_PostRequestHandlerExecute() – fired after HTTP handler has executed the request.

Application_ReleaseRequestState() – fired before current state data kept in the session collection is serialized.

Application_UpdateRequestCache() – fired before information is added to output cache of the page.

Application_EndRequest() – fired at the end of each request

Methods corresponding to events that do not fire on each request

Application_Start() – fired when the first resource is requested from the web server and the web application starts.

Session_Start() – fired when session starts on each new user requesting a page.

Application_Error() – fired when an error occurs.

Session_End() – fired when the session of a user ends.

Application_End() – fired when the web application ends.

Application_Disposed() - fired when the web application is destroyed.

Show me an example!!

Let us see an example of how to use the Global.asax to catch unhandled errors that occur at the application level.

To catch unhandled errors, do the following. Add a Global.asax file (Right click project > Add New Item > Global.asax). In the Application_Error() method, add the following code:

VB.NET

```
Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)

    ' Code that runs when an unhandled error occurs

    Dim objErr As Exception = Server.GetLastError().GetBaseException()

    Dim err As String = "Error in: " & Request.Url.ToString() & ". Error Message:" &
objErr.Message.ToString()

 End Sub
```

Here we make use of the Application_Error() method to capture the error using the Server.GetLastError().

Asp:GridView control

It provides more flexibility in displaying and working with data from your database in comparison with any other controls. The GridView control enables you to connect to a datasource and display data is tabular format, however you have bunch of options to customize the look and feel. When it is rendered on the page, generally it is implemented through <table> HTML tag.

Following are some important properties that are very useful.

Behavior Properties of the GridView Control	
AllowPaging	true/false. Indicate whether the control should support paging.
AllowSorting	true/false. Indicate whether the control should support sorting.
SortExpression	Gets the current sort expression (field name) that determines the order of the row.
SortDirection	Gets the sorting direction of the column sorted currently (Ascending/Descending).
DataSource	Gets or sets the data source object that contains the data to populate the control.
DataSourceID	Indicate the bound data source control to use (Generally used when we are using SqlDataSource or AccessDataSource to bind the data, See 1st Grid example).
AutoGenerateEditButton	true/false. Indicates whether a separate column should be added to edit the record.
AutoGenerateDeleteButton	true/false. Indicates whether a separate column should be added to delete the record.
AutoGenerateSelectButton	true/false. Indicate whether a separate column should be added to selecat a particular record.
AutoGenerateColumns	true/false. Indicate whether columns are automatically created for each field of the data source. The default is true.
Style Properties of the GridView Control	
AlternatingRowStyle	Defines the style properties for every alternate row in the GridView.
EditRowStyle	Defines the style properties for the row in EditView (When you click Edit button for a row, the row will appear in this style).
RowStyle	Defines the style properties of the rows of the GridView.
PagerStyle	Defines the style properties of Pager of the GridView. (If AllowPaging=true, the page number row appears in this style)
EmptyDataRowStyle	Defines the style properties of the empty row, which appears if

	there is no records in the data source.
HeaderStyle	Defines the style properties of the header of the GridView. (The column header appears in this style.)
FooterStyle	Defines the style properties of the footer of GridView.
Appearance Properties of the GridView Control	
CellPadding	Indicates the space in pixel between the cells and the border of the GridView.
CellSpacing	Indicates the space in pixel between cells.
GridLines	Both/Horizontal/Vertical/None. Indicates whether GrdiLines should appear or not, if yes Horizontal, Vertical or Both.
HorizontalAlign	Indicates the horizontal align of the GridView.
EmptyDataText	Indicates the text to appear when there is no record in the data source.
ShowFooter	Indicates whether the footer should appear or not.
ShowHeader	Indicates whether the header should appear or not. (The column name of the GridView)
BackImageUrl	Indicates the location of the image that should display as a background of the GridView.
Caption	Gets or sets the caption of the GridView.
CaptionAlign	left/center/right. Gets or sets the horizontal position of the GridView caption.
State Properties of GridView Control	
Columns	Gets the collection of objects that represent the columns in the GridView.
EditIndex	Gets or sets the 0-based index that identifies the row currently to be edited.
FooterRow	Returns a GridViewRow object that represents the footer of the GridView.
HeaderRow	Returns a GridViewRow object that represents the header of the GridView.
PageCount	Gets the number of the pages required to display the reocrds of the data source.
PageIndex	Gets or sets the 0-based page index.
PageIndex	Gets or sets the number of records to display in one page of GridView.

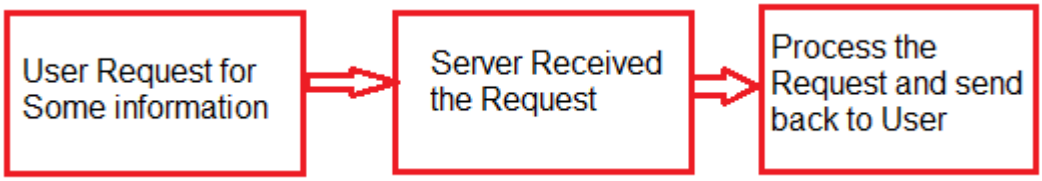
Rows	Gets a collection of GridViewRow objects that represents the currently displayed rows in the GridView.
DataKeyNames	Gets an array that contains the names of the primary key field of the currently displayed rows in the GridView.
DataKeys	Gets a collection of DataKey objects that represent the value of the primary key fields set in DataKeyNames property of the GridView.
Events associated with GridView Control	
PageIndexChanging, PageIndexChanged	Both events occur when the page link is clicked. They fire before and after GridView handles the paging operation respectively.
RowCommand	Fires when a button is clicked on any row of GridView.
RowDeleting,RowDeleted	Both events fires when Delete button of a row is clicked. They fire before and after GridView handles deleting operaton of the row respectively.
RowEditing	Fires when a Edit button of a row is clicked but before the GridView hanldes the Edit operation.
RowUpdating, RowUpdated	Both events fire when a update button of a row is clicked. They fire before and after GridView control update operation respectively.
Sorting, Sorted	Both events fire when column header link is clicked. They fire before and after the GridView handler the Sort operation respectively.

What is IIS - Internet Information Server

Internet Information Server

Internet Information Server (IIS) is one of the most popular web servers from Microsoft that is used to host and provide Internet-based services to ASP.NET and ASP Web applications. A web server is responsible for providing a response to requests that come from users. When a request comes from client to server IIS takes that request from users and process it and send response back to users.

Internet Information Server (IIS) has it's own ASP.NET Process Engine to handle the ASP.NET request. The way you configure an ASP.NET application depends on what version of IIS the application is running on.



Internet Information Server (IIS) includes a set of programs for building and administering Web applications, search engines, and support for writing Web-based applications that access databases such as SQL Server. With IIS, you can make your computer to work as a Web server and provides the functionality to develop and deploy ASP.NET Web applications on the server. You can also set security for a particular Website for specific Users and Computer in order to protect it from unauthorized access.

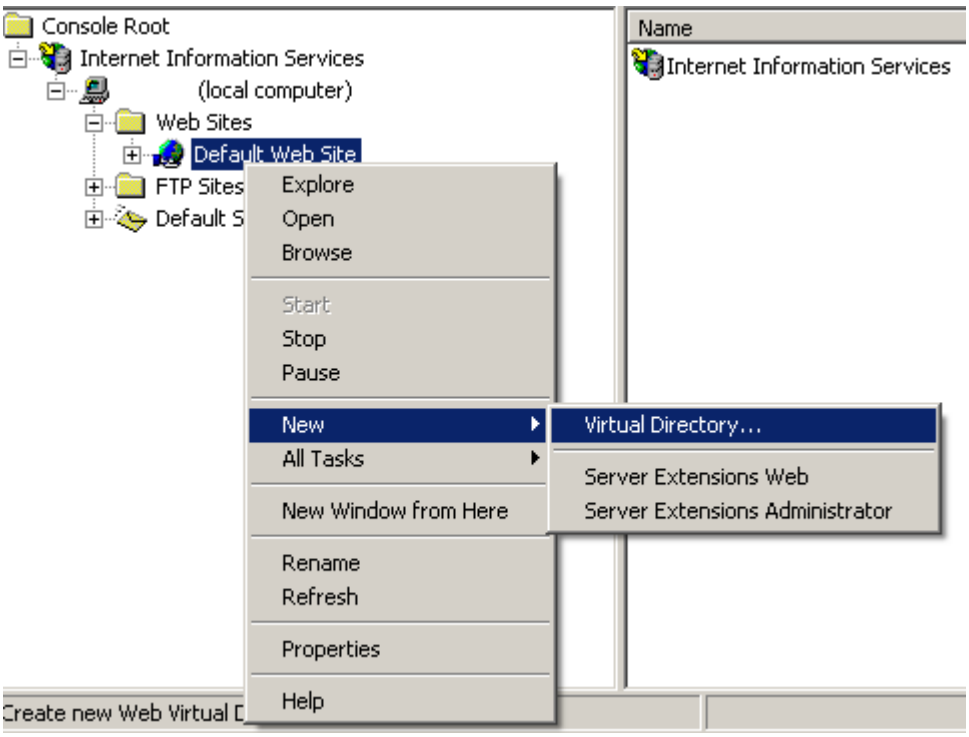
What is Virtual Directory

Virtual Directory

A virtual directory is a directory name that you specify in IIS and map to physical directory on a local server's hard drive or a directory on another server (remote server). You can use Internet Information Services Manager to create a virtual directory for an ASP.NET Web application that is hosted in IIS.

The virtual directory name becomes part of the application's URL. It is a friendly name, or alias because an alias is usually shorter than the real path of the physical directory and it is more convenient for users to type. A virtual directory receives queries and directs them to the appropriate backend identity repositories. It integrates identity data from multiple heterogeneous data stores and presents it as though it were coming from one source.

How to create a virtual directory by using IIS Manager



1. In IIS Manager, expand the local computer and the Web site to which you want to add a virtual directory.

2. Right-click the site or folder in which you want to create the virtual directory, click New, and then click Virtual Directory.

3. In the Add Virtual Directory dialog box, at a minimum enter information in the Alias and Physical path and then click OK.

By default, Internet Information Server uses configuration from Web.config files in the physical directory to which the virtual directory is mapped, as well as in any child directories in that physical directory

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 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:-

Properties of the Login Control	
TitleText	Indicates the text to be displayed in the heading of the control.
InstructionText	Indicates the text that appears below the heading of the control.
UserNameLabelText	Indicates the label text of the username text box.
PasswordLabelText	Indicates the label text of the password text box.
FailureText	Indicates the text that is displayed after failure of login attempt.
UserName	Indicates the initial value in the username text box.
LoginButtonText	Indicates the text of the Login button.
LoginButtonType	Button/Link/Image. Indicates the type of login button.
DestinationPageUrl	Indicates the URL to be sent after login attempt successful.
DisplayRememberMe	true/false. Indicates whether to show Remember Me checkbox or not.
VisibleWhenLoggedIn	true/false. If false, the control is not displayed on the page when the user is logged in.
CreateUserUrl	Indicates the url of the create user page.
CreateUserText	Indicates the text of the create user link.
PasswordRecoveryUrl	Indicates the url of the password recovery page.
PasswordRecoveryText	Indicates the text of the password recovery link.

Events of the Login Control	
LoggingIn	Fires before user is going to authenticate.
LoggedIn	Fires after user is authenticated.
LoginError	Fires after failure of login attempt.
Authenticate	Fires to authenticate the user. This is the function where you need to write your own code to validate the user.

'Code in Authenticate Event

```
If Login1.UserName = "rofel" And Login1.Password = "bca" Then
          Response.Redirect("default.aspx")
      Else
          Login1.FailureText = "Enter Correct User name and Password"
      End If
```

2. The LoginView Control

The LoginView control is a web server control used for displaying the two different views of a web page. It helps to alter the page view for different logged in users. The current user's status information is stored in the control. The control displays appropriate information depending on the user.

The LoginView class provides the LoginView control. The methods, properties and events provided by the login class are as listed below:

Methods of the LoginView class

1. DataBind: It helps user to bind the data source through the LoginView control.
2. OnViewChanged: It raises the ViewChanged event after the view for the control is changed.
3. OnViewChanging: It raises the ViewChanging event before the LoginView control changes the view.

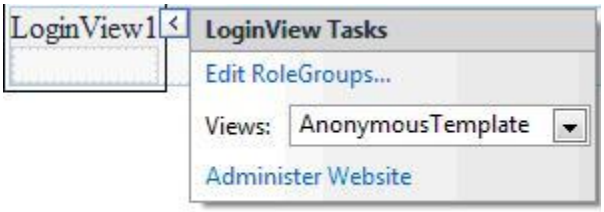
Properties of the LoginView class

1. Controls: It accesses the ControlCollection object containing the child controls for the LoginView control
2. EnableTheming: It access or specifies the value indicating the themes to be applied to the control
3. RoleGroups: It access the collection of role groups associated with the content templates

Events of the LoginView class

1. ViewChanged: It is initiated when the view is changed
2. ViewChanging: It is initiated when the view is in the process to be changed.

The LoginView control at the design time is as shown below:



3. The LoginStatus Control

It specifies that a particular user has logged into the web site. The login status is displayed as a text. The login text is displayed as a hyperlink but provides the navigation to the login page. The authentication section of the web.config file is useful for accessing the login page URL.

The LoggedIn and LoggedOut are the rwo status provided by the LoginStatus control. TheLoginStatus class provides the control. The methods, properties and events for the control are as mentioned below:

Methods of the LoginStatus Control

1. OnLoggedOut: It raises the event when the logout link is clicked by the user.
2. OnLoggingOut: It raises the event when the user clicks the logout link of the control.

Properties of the LoginStatus Control

1. LoginImageUrl: It accesses or specifies the URL of the image used for the login link.
2. LoginText: It access the text added for the login link
3. LogoutAction: It retrieves the value for determining the action when the user logs out of the web site.
4. LogoutText: It retrieves the text used for logout the link

Events of the LoginStatus Control

1. LogginOut: It is initiated when the user sends the logout request to the server.
2. LoggedOut: It is initiated by the LoginStatus class when the user logout process is completed

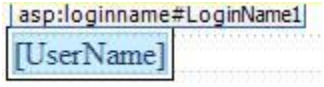
The LoginStatus control at the design time is as shown below:

4. LoginName Control

It is used for displaying the name of the authenticated users. The Page.User.Identity.Name is used for returning the user name. The control is not displayed if it does not contain any logged in user. The LoginName class is used for the control.

The control does not contain any method, property or events associated with it. The FormatString property is used for displaying the string in the control.

The LoginName control at the design time is as shown below:



5. PasswordRecovery Control

It is used to recover or reset the password for the user. The password is sent through an email as a message at the registration time. The Membership service is used for creating and resetting the password.

The control contains the following three views.

1. Question: It refers the view where the user can enter the answer to the security question.
2. UserName: It refers to the view where the user can enter the username for the password to be recovered.
3. Success: It represents the view where the message is displayed to the user.

The control contains various properties, methods and events as mentioned below:

Methods of the PasswordRecovery Control

1. OnSendingMail: It raises the SendingMail event when the user is verified and the password is sent to the user.
2. OnUserLookupErrror: It raises the UserLookupError when the username does not match with the one stored in the database,
3. OnSendMailError: It raises an error when the mail message is not sent to the user.
4. OnVerifyingUser: It raises the event once the username is submitted, and the membership provider verification is pending.

Properties of the control

1. Answer: The answer provided by the user to confirm the password recovery through the valid user
2. FailureTextStyle: It accesses the reference to the collection of properties defining the error text look
3. HelpPageIconUrl: It image to be displayed for the link to the password is retrieved

Events of the control

1. SendingMail: It is initiated when the server is sending an email message containing the password once the answer is correct
2. AnswerLookupError: It is initiated when the user answer to the question is incorrect
3. VerifyingAnswer: It is initiated when the user has submitted the answer to the password recovery confirmation question

The PasswordRecovery control at the design time is as shown below:

6. CreateUserWizard Control

The control uses the Membership service for creation of a new user. The control can be extended to the existing Wizard control. The control can be customized through templates and properties.
Some of the properties, methods and events related to the control are as mentioned below:

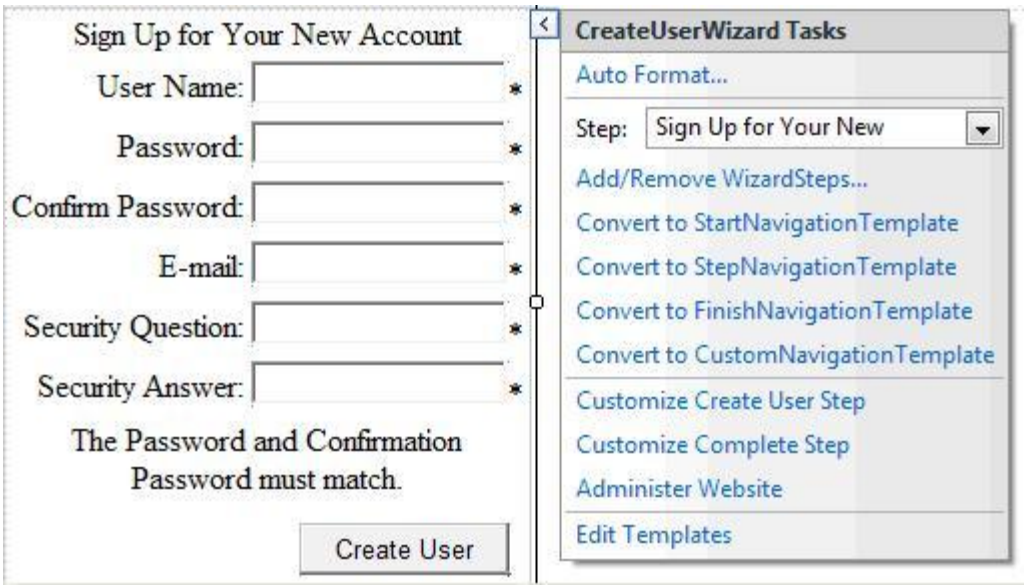
Properties of the Control

1. Answer: It retrieves or specifies the answer to the password recovery confirmation question.
2. CompleteStep: It shows the final step of the process for creating the user account.
3. ContinueButtonText: It accesses or specifies the collection of properties defining the look of the control
4. Email: It retrieves the email address of the user
5. LoginCreatedUser: It accesses or specifies the value indicating the new user login once the account is created.

Events of the control

1. CreatedUser: It is initiated after the membership service provider has created a new user account
2. CreatingUser: It is initiated before the membership service provider is called for creating user account
3. SendingMail: It is initiated before sending the conformation email on the successful creation of the account
4. SendMailError: It is initiated when the SMTP error occurs during the mail sent to the user.

The CreateUserWizard control at the design time is as shown below:



7. ChangePassword Control

The control helps user to change the password. The user adds the current password and adds the new password. If the old password is incorrect, the new one cannot be added.

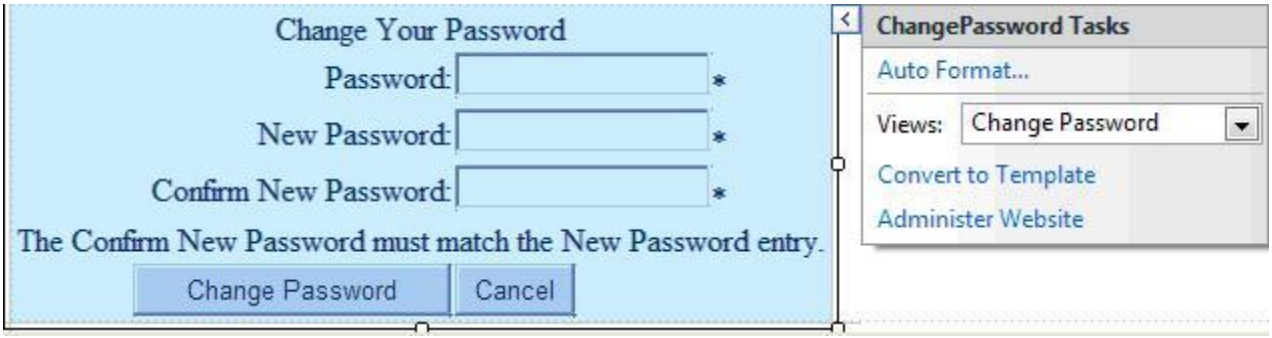
Properties of the control

1. CancelDestinationPageUrl: It accesses or retrieves the URL of the page that the user is shown once it clicks the Cancel button.
2. CurrentPassword: It retrieves the current password of a user.
3. DisplayUserName: It retrieves the value indicating whether the ChangePassword control should be display the control and label
4. NewPassword: It retrieves the new password entered by the user
5. UserName: It shows the username for which the password is to be modified.

Events of the control

1. ChangedPassword: It is initiated when the password is changed for the user account.
2. ChangePasswordError: It is initiated when there is an error in changing the password for the user account
3. SendMailError: It is initiated when the SMTP error occurs during sending an email message

The ChangePassword control at the design time is as shown below:



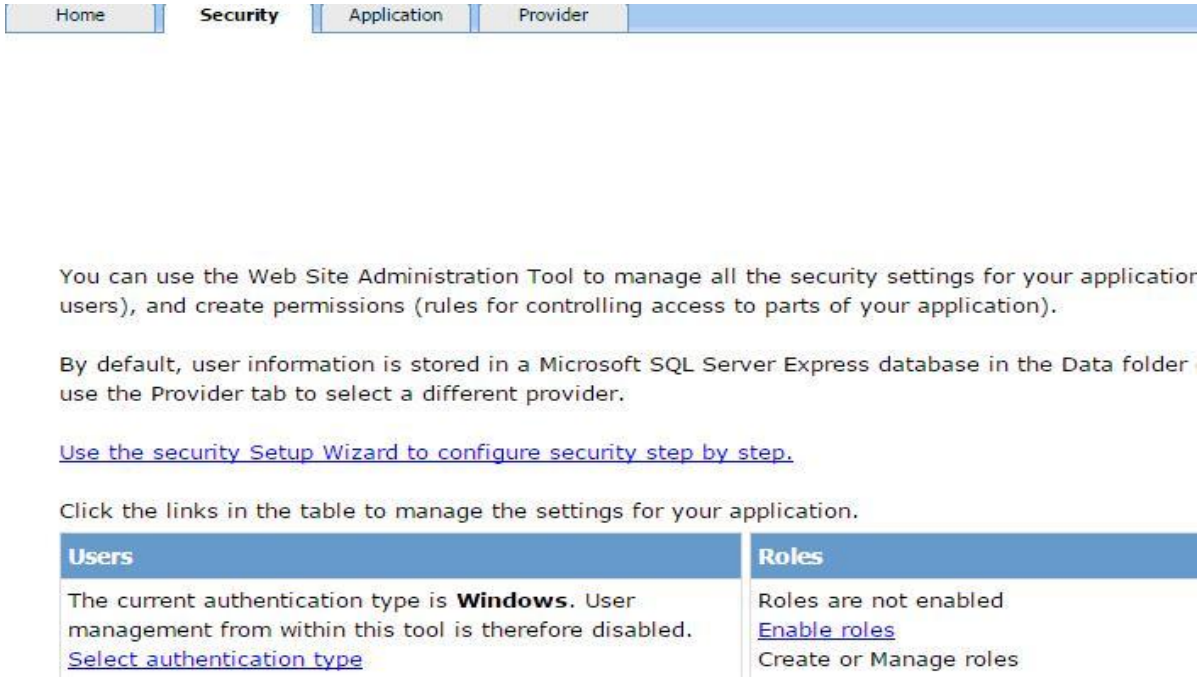
Implementing Authentication in ASP.NET login controls

Consider an example to demonstrate the login controls in an ASP.NET application. Perform the following steps to demonstrate the implementation of the login controls in application.

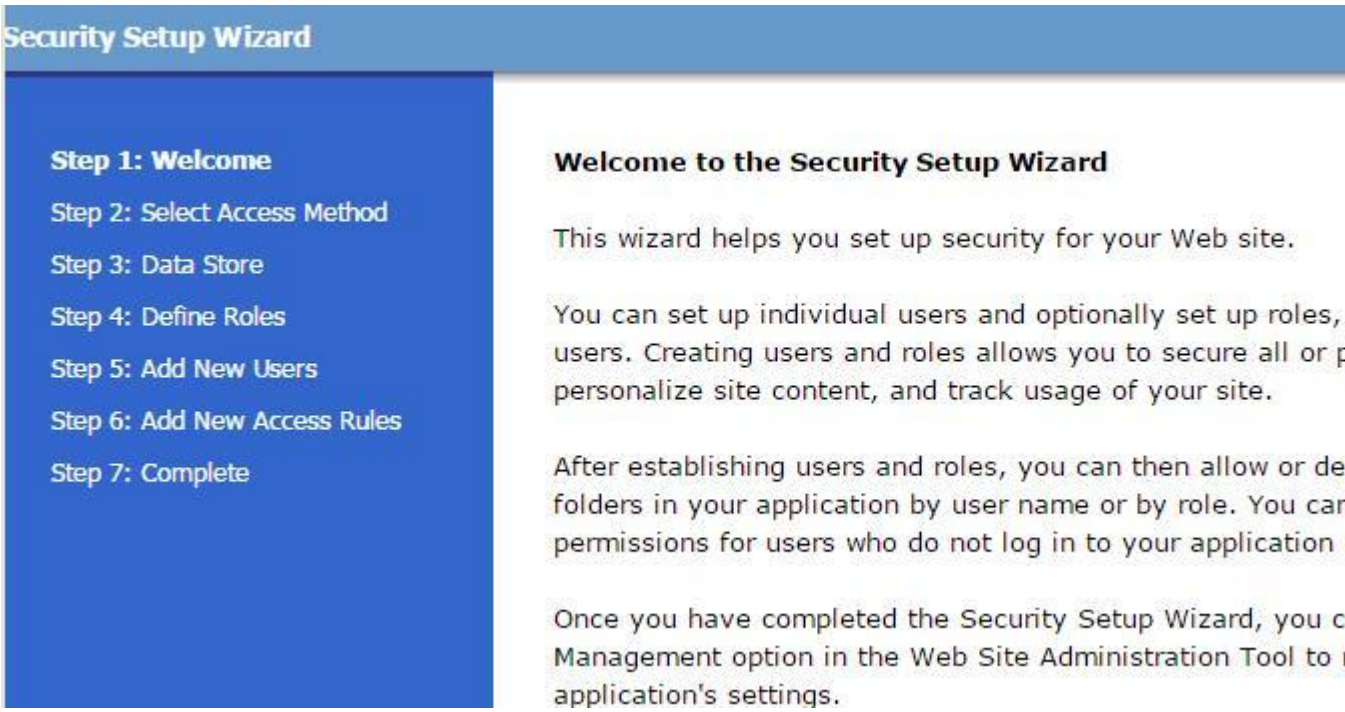
1. Place the login control in the .aspx form and change the AutoFormat style property to Classic.

2. Click the Smart Tag and open the Login Tasks and select the Administer Website option.

3. Click the Security link in the window



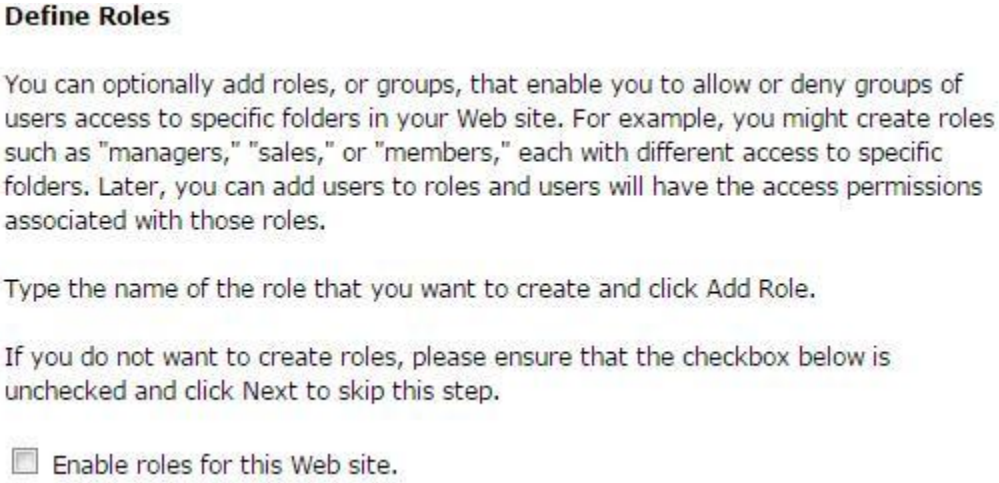
4. Click the Use the security Setup Wizard to configure security step by step link to open the setup wizard
5. Click Next button in the welcome the security setup wizard.



6. Click the From the Internet radio button and click the Next button.

7. Click the Next button in the Advance provider settings page.

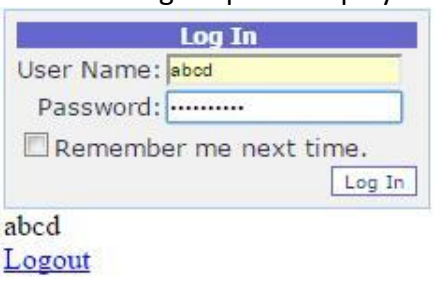
8. Select the Enable roles for this web site check box and click the Next button



9. Add the details in the text boxes and click the Create User button to create the user account.

10. Select the All Users radio button in the Rule applies to section.
11. Click the Add this Rule button. Click Next button

12. Click Finish button, click Close button
13. Add the LoginName and LoginStatus controls on the web page
14. Set the LogoutAction property to Redirect, click the smart tag of the LoginStatus control and select the Logged In option from the Views drop down list.
15. Execute the application and enter the username and password in the text boxes. Click Log In button.
16. The following output is displayed when the application is executed on the server.



What are Master Pages ?

Master pages let you make a consistent layout for your application, you can make one **master page** that holds the layout/look & feel and common functionality of your whole application and upon this **master page**, you can build all the other **page**s, we call these **page**s **Content Pages.** So simply you can build your **master page** and then build content **page**s, and while creating the content **page**s you bind them to the **master page** you have created before, those two **page**s are merged at runtime to give you the rendered **page**.

Master Pages and ContentPlaceHolders

The **master page** has the extension .**master** and it's actually an aspx **page** but with the directive `<%@Master Language="vb"%>`, instead of the standard **page** directive, almost the attributes of the **Master** directive are the same as that of the **page**, you can add any kind of controls the same as you design .aspx **page**s,

Every **MasterPage** should include at least one `ContentPlaceHolder`, this control is a placeholder for the content that will be merged at runtime, noting that the content **page** is just an aspx standard **page** but you should supply the attribute **`MasterPage`**`File` to the **page** directive to bind the content **page** to the **master page**, for example:

```
<%@ Page Language="VB" MasterPageFile="~/MasterPages/SiteLayout.master"%>
```

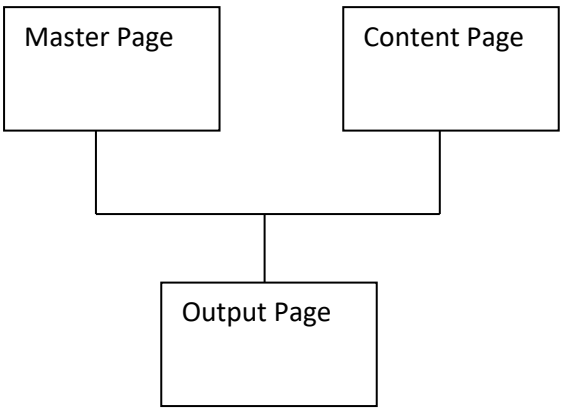
Content Server Control

Inside the content **page**s you will find one Content server control added by default, actually when you add controls you add them to the content server control. For example,

```
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1"
             Runat="Server">
```

The attribute `ContentPlaceHolderID` is very important as it decides what content will be bound to which `ContentPlaceHolder` on the **master page**, this is a very nice way to decide the location of where the contents will be displayed; so this way you can have multiple content on one content **page** and also multiple `ContentPlaceHolder`s on the **master page**.

Note: The content **page**s don't include the common tags as `<Body>`, `<Head>`,etc. Remember that, that was the same with user controls, as after merging, there should be only one `Body` and `Head` tags.



Terminology

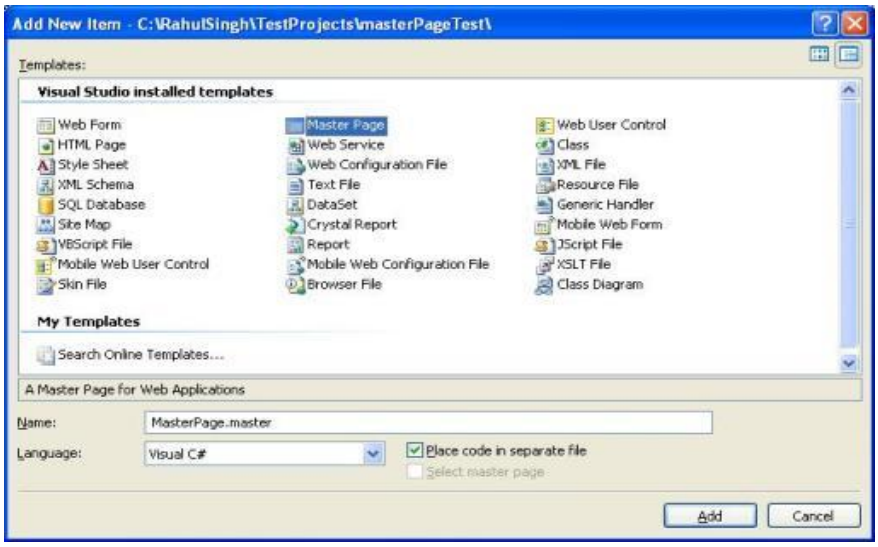
Let us look at the basic terminology that needs to be understood before jumping into master pages:

- `Masterpage`: Gives us a way to create common set of UI elements that are required on multiple pages of our website.
- `ContentPage`: The ASP.NET web page that will use master page to have the common UI elements displayed on rendering itself.
- `ContentPlaceHolder`: A control that should be added on the `MasterPage` which will reserve the area for the content pages to render their contents.
- `ContentControl`: A control which will be added on content pages to tell these pages that the contents inside this control will be rendered where the `MasterPage`'s `ContentPlaceHolder` is located.

Creating a MasterPage

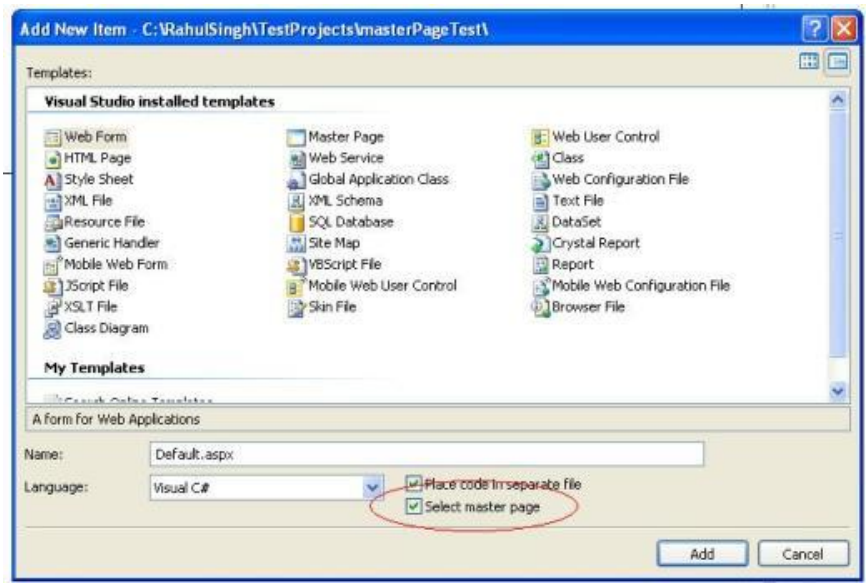
To create a master page, we need to:

1. Go to "Add New Item".
2. Select the `MasterPage`.

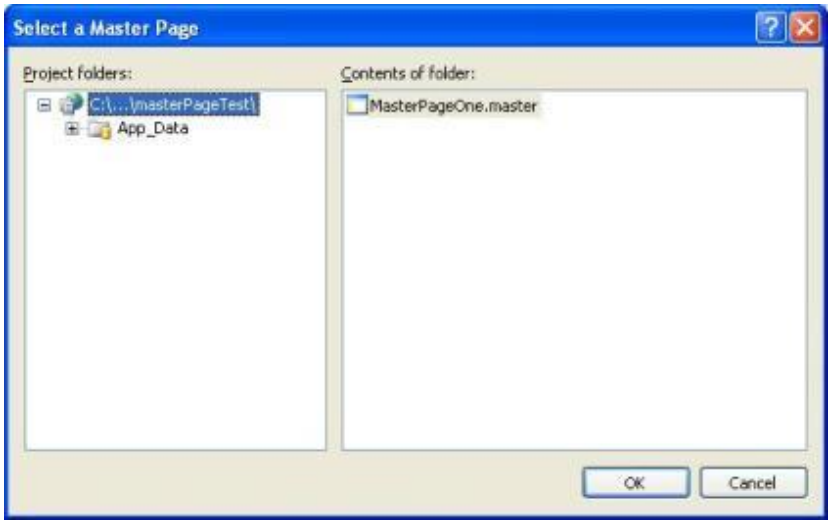


3. Let's say our master page is `MasterPageOne.Master`.
4. We will now add a menu bar on this master page on top of the page. This Menu bar will be common to all the pages (since it is in `Masterpage`).
5. Once we have menubar added, we can have content pages use the master page.
6. Let's add few content pages like *default.aspx*, *about.aspx*, *Contact.aspx*. (We are simply creating some dummy pages with no functionality as we want to see the masterpage working, but these content pages can have any level of complex logic in them).

7. When we add these content pages, we need to remember to select the option of "Use master Page".



and select the master page.



Now let's look at the stuff that is important. When we look at the `MasterPage`, we will see that `masterpage` has a `ContentPlaceHolder` control. All the code that is common for the content pages is outside the `ContentPlaceHolder` control (in our case, a simple menubar).

Adding the ContentPages

If we look at our content pages, we will find a simple `Content` control added to each content page. This is the area where we will be adding our controls to be rendered along with the master page.

```
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1"
Runat="Server">
<h2>This is a the CONTACT page.</h2>
</asp:Content>
```

Advantages of Master Page

1. You can make updates in one place as they allow you to centralize the common functionality of your pages.
2. With the help of Master pages, it is 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.
3. You can provide an object model which allows you to customize the master page from individual content pages.

Nested Master Pages

You can use more then one master page on your website. When more than one master page is used, you can make use of nested master pages.

For example, consider your company has a number of business partners or franchise companies. In such a scenario, you can define the layout and design for the standard elements such as logos, menus, copyright notices on the main master page of your company's website. The franchise companies can then also define their own master pages and then nest their master page with the master page of your company.

Understanding Nested Master Pages:

When a master page contains a reference of another master page, then it is called a **"nested master page"**. A single master page can have a reference of multiple master pages or a number of master pages can be componentized into a single master page. There is no limit to the number of child master pages in a project. The child masters can contain some unique properties of their own, besides using the layout and other properties of their parent master.

Menu

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 node displayed 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.
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).

```
// Menu Control ////////////////////////////
<asp:Menu ID="Menu1" runat="Server" DataSourceID="SiteMapDataSource1"
         Orientation="Horizontal" BackColor="#B5C7DE"
DynamicHorizontalOffset="2" Font-Names="Verdana" Font-Size="0.8em"
ForeColor="#284E98" StaticDisplayLevels="2"
StaticSubMenuIndent="10px"
           >
            <StaticMenuItemStyle HorizontalPadding="5px"
VerticalPadding="2px" />
            <DynamicHoverStyle BackColor="#284E98" ForeColor="White"
/>
            <DynamicMenuStyle BackColor="#B5C7DE" />
            <StaticSelectedStyle BackColor="#507CD1" />
            <DynamicSelectedStyle BackColor="#507CD1" />
            <DynamicMenuItemStyle HorizontalPadding="5px"
VerticalPadding="2px" />
            <StaticHoverStyle BackColor="#284E98" ForeColor="White"
/>
        </asp:Menu>


// SiteMapDataSource Control ////////////////////////////
        <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="Server"
/>
```

ASP.NET Page Directory

The asp.net application folder is contains list of specified folder that you can use of specific type of files or content in an each folder. The root folder structure is as following

- BIN
- App_Code
- App_GlobalResources
- App_LocalResources
- App_WebReferences
- App_Data
- App_Browsers
- App_Themes

Bin Directory

It is contains all the precompiled .Net assemblies like DLLs that the purpose of application uses.

App_Code Directory

It is contains source code files like .cs or .vb that are dynamically compiled for use in your application. These source code files are usually separate components or a data access library

App_GlobalResources Directory

It is contains to stores global resources that are accessible to every page.

App_LocalResources Directory

It is serves the same purpose as app_globalresources, exept these resources are accessible for their dedicated page only

App_WebReferences Directory

It is stores reference to web services that the web application uses.

App_Data Directory

It is reserved for data storage and also mdf files, xml file and so on.

App_Browsers Directory

It is contains browser definitions stored in xml files. These xml files define the capabilities of client side browsers for different rendering actions.

App_Themes Directory

It is contains collection of files like .skin and .css files that used to application look and feel appearance.

ASP.NET Page Life Cycle

When a page is requested, it is loaded into the server memory, processed, and sent to the browser. Then it is unloaded from the memory. At each of these steps, methods and events are available, which could be overridden according to the need of the application. In other words, you can write your own code to override the default code.

Following are the different stages of an ASP.NET page:

- **Page request** - When ASP.NET gets a page request, it decides whether to parse and compile the page, or there would be a cached version of the page; accordingly the response is sent.
- **Starting of page life cycle** - At this stage, the Request and Response objects are set. If the request is an old request or post back, the IsPostBack property of the page is set to true. The UICulture property of the page is also set.
- **Page initialization** - At this stage, the controls on the page are assigned unique ID by setting the UniqueID property and the themes are applied. For a new request, postback data is loaded and the control properties are restored to the view-state values.
- **Page load** - At this stage, control properties are set using the view state and control state values.
- **Validation** - Validate method of the validation control is called and on its successful execution, the IsValid property of the page is set to true.
- **Postback event handling** - If the request is a postback (old request), the related event handler is invoked.
- **Page rendering** - At this stage, view state for the page and all controls are saved. The page calls the Render method for each control and the output of rendering is written to the OutputStream class of the Response property of page.
- **Unload** - The rendered page is sent to the client and page properties, such as Response and Request, are unloaded and all cleanup done.

ASP.NET Page Life Cycle Events

At each stage of the page life cycle, the page raises some events, which could be coded. An event handler is basically a function or subroutine, bound to the event, using declarative attributes such as Onclick or handle.

Following are the page life cycle events:

- **PreInit** - PreInit is the first event in page life cycle. It checks the IsPostBack property and determines whether the page is a postback. It sets the themes and master pages, creates dynamic controls, and gets and sets profile property values. This event can be handled by overloading the OnPreInit method or creating a Page_PreInit handler.
- **Init** - Init event initializes the control property and the control tree is built. This event can be handled by overloading the OnInit method or creating a Page_Init handler.

- **InitComplete** - InitComplete event allows tracking of view state. All the controls turn on view-state tracking.
- **LoadViewState** - LoadViewState event allows loading view state information into the controls.
- **LoadPostData** - During this phase, the contents of all the input fields are defined with the <form> tag are processed.
- **PreLoad** - PreLoad occurs before the post back data is loaded in the controls. This event can be handled by overloading the OnPreLoad method or creating a Page_PreLoad handler.
- **Load** - The Load event is raised for the page first and then recursively for all child controls. The controls in the control tree are created. This event can be handled by overloading the OnLoad method or creating a Page_Load handler.
- **LoadComplete** - The loading process is completed, control event handlers are run, and page validation takes place. This event can be handled by overloading the OnLoadComplete method or creating a Page_LoadComplete handler
- **PreRender** - The PreRender event occurs just before the output is rendered. By handling this event, pages and controls can perform any updates before the output is rendered.
- **PreRenderComplete** - As the PreRender event is recursively fired for all child controls, this event ensures the completion of the pre-rendering phase.
- **SaveStateComplete** - State of control on the page is saved. Personalization, control state and view state information is saved. The HTML markup is generated. This stage can be handled by overriding the Render method or creating a Page_Render handler.
- **UnLoad** - The UnLoad phase is the last phase of the page life cycle. It raises the UnLoad event for all controls recursively and lastly for the page itself. Final cleanup is done and all resources and references, such as database connections, are freed. This event can be handled by modifying the OnUnLoad method or creating a Page_UnLoad handler.

What is Repeater Control?

Repeater Control is a control which is used to display the repeated list of items

Uses of Repeater Control

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 it's templates called item template. Repeater control contains different types of template fields those are

Repeater Control Templates

Repeater controls provides different kinds of templates which helps in determining the layout of control's content. Templates generate markup which determine final layout of content.

Repeater control is an iterative control in the sense it loops each record in the DataSource and renders the specified template (ItemTemplate) for each record in the DataSource collection. In addition, before and after processing the data items, the Repeater emits some markup for the header and the footer of the resulting structure

Repeater control supports five templates which are as follows:

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

5) SeperatorTemplate

ItemTemplate: ItemTemplate defines how the each item is displays 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.

DEMO : Repeater

This is the Header of the Repeater Control

6477	Jjh	Jhjh	jhhjj	jjkjk

6474	MallaReddy	Hyd	12345	Hyd

6480	Mkmk	Ji	eee	eee

6476	Mndsam	Dmsna	mndsa	msna

```
   // Repeater control ////////////////////////////
        <asp:Repeater ID="Repeater1" runat="server"
DataSourceID="SqlDataSource1">
        <HeaderTemplate>
            <h3>This is the Header of the Repeater Control</h3>
        </HeaderTemplate>
        <AlternatingItemTemplate>
            <table border="1" style="background-color:#c0c0c0;" width="100%">
<tr>
<td style="width:10%;"><%# Eval("AutoID") %></td>
<td style="width:25%;"><%# Eval("Name") %></td>
<td style="width:40%;"><%# Eval("Address") %></td></tr></table>
        </AlternatingItemTemplate>
        <ItemTemplate>
            <table border="1" width="100%">
                <tr><td style="width:10%;"><%# Eval("AutoID") %></td>
                    <td style="width:25%;"><%# Eval("Name") %></td>
                    <td style="width:40%;"><%#
Eval("Address") %></td></tr></table></ItemTemplate></asp:Repeater>

    // SqlDataSource control ////////////////////////////
     <asp:SqlDataSource ID="SqlDataSource1" runat="server"
ConnectionString='<%$ ConnectionStrings:ConnStr %>'
     SelectCommand="Select * FROM emp ORDER BY [Name]">
     </asp:SqlDataSource>
```

Request/Response Programming

The server control architecture is built on top of a more fundamental processing architecture, which may be called *request/response*. Understanding request/response is important to solidify our overall grasp of ASP.NET. Also, in certain programming situations request/response is the natural approach.

HttpRequest Class

The System.Web namespace contains a useful class HttpRequest that can be used to read the various HTTP values sent by a client during a Web request. These HTTP values would be used by a classical CGI program in acting upon a Web request, and they are the foundation upon which higher level processing is built. Table 14–1 shows some of the public instance properties of HttpRequest. If you are familiar with HTTP, the meaning of these various properties should be largely self-explanatory. Refer to the .NET Framework documentation of the HttpRequest class for full details about these and other properties.

TABLE 14–1 Public Instance Properties of HttpRequest

Property	Meaning
AcceptTypes	String array of client-supported MIME accept types
Browser	Information about client's browser capabilities
ContentLength	Length in bytes of content sent by the client
Cookies	Collection of cookies sent by the client
Form	Collection of form variables
Headers	Collection of HTTP headers
HttpMethod	HTTP transfer method used by client (e.g., GET or POST)
Params	Combined collection of QueryString, Form, ServerVariables, andCookies items
Path	Virtual request of the current path
QueryString	Collection of HTTP query string variables
ServerVariables	Collection of Web server variables

The Request property of the Page class returns a HttpRequest object. You may then extract whatever information you need, using the properties of HttpRequest. For example, the following code determines the length in bytes of content sent by the client and writes that information to the Response object.

```
Dim length As Integer = Request.ContentLength
```

```
Response.Write("ContentLength = " & length & "<br>")
```

COLLECTIONS

A number of useful collections are exposed as properties of HttpRequest. The collections are of type NamedValueCollection (in System.Collec-tions.Specialized namespace). You can access a value from a string key. For example, the following code extracts values for the QUERY_STRING and HTTP_USER_AGENT server variables using the ServerVariables collection.

```
Dim strQuery As String = _
    Request.ServerVariables("QUERY_STRING")
Dim strAgent as String = _
    Request.ServerVariables("HTTP_USER_AGENT")
```

Server variables such as these are at the heart of classical Common Gateway Interface (CGI) Web server programming. The Web server passes information to a CGI script or program by using environment variables. ASP.NET makes this low-level information available to you, in case you need it.

A common task is to extract information from controls on forms. In HTML, controls are identified by a name attribute, which can be used by the server to determine the corresponding value. The way in which form data is passed to the server depends on whether the form uses the HTTP GET method or the POST method.

With GET, the form data is encoded as part of the query string. The QueryString collection can then be used to retrieve the values. With POST, the form data is passed as content after the HTTP header. The Forms collection can then be used to extract the control values. You could use the value of the REQUEST_METHOD server variable (GET or POST) to determine which collection to use (the QueryString collection in the case of GET and the Forms collection in case of POST).

With ASP.NET you don't have to worry about which HTTP method was used in the request. ASP.NET provides a Params collection, which is a combination (union in the mathematical sense) of the ServerVariables, Que-ryString, Forms, and Cookies collections.

EXAMPLE PROGRAM

We illustrate all these ideas with a simple page Squares.aspx that displays a column of squares.

```
<!-- Squares.aspx -->
<%@ Page Language="VB" Trace="true"%>
<script runat="server">
Sub Page_Init(sender As Object, e As EventArgs)
    Dim strQuery As String = _
        Request.ServerVariables("QUERY_STRING")
    Response.Write("QUERY_STRING = " & strQuery & "<br>")
    Dim strAgent as String = _
        Request.ServerVariables("HTTP_USER_AGENT")
```

```
    Response.Write("HTTP_USER_AGENT = " & strAgent & "<br>")
    Dim length As Integer = Request.ContentLength
    Response.Write("ContentLength = " & length & "<br>")
    Dim strCount As String = Request.Params("txtCount")
    Dim count As Integer = Convert.ToInt32(strCount)
    Dim i As Integer
    For i = 1 To count
        Response.Write(i*i)
        Response.Write("<br>")
    Next
End Sub
</script>
```

How many squares to display is determined by a number submitted on a form. The page GetSquares.aspx submits the request using GET, and PostSquares.aspx submits the request using POST. These two pages have the same user interface, illustrated in Figure 14–11.

Figure 14-11 Form for requesting a column of squares.

Here is the HTML for GetSquares.aspx. Notice that we are using straight HTML. Except for the Page directive, which turns tracing on, no features of ASP.NET are used.

```
<!-- GetSquares.aspx -->
<%@ Page Trace = "true" %>
<html>
<head>
</head>
<body>
<P>This program will print a column of squares</P>
<form

method="get" action = Squares.aspx

>
How many:
<INPUT type=text size=2 value=5

name=txtCount

>
<P></P>
<INPUT type=submit value=Squares

name=cmdSquares

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

The form tag has attributes specifying the method (GET or POST) and the action (target page). The controls have a name attribute, which will be used by server code to retrieve the value.

Run GetSquares.aspx and click Squares. You will see some HTTP information displayed, followed by the column of squares. Tracing is turned on, so details about the request are displayed by ASP.NET. Figure 14–12 illustrates the output from this GET request.

Figure 14-12 Output from a GET request.

You can see that form data is encoded in the query string, and the content length is 0. If you scroll down on the trace output, you will see much information. For example, the QueryString collection is shown.

Now run PostSquares.aspx and click Squares. Again you will then see some HTTP information displayed, followed by the column of squares. Tracing is turned on, so details about the request are displayed by ASP.NET. Figure 14–13 illustrates the output from this POST request.

Figure 14-13 Output from a POST request.

You can see that now the query string is empty, and the content length is 29. The form data is passed as part of the content, following the HTTP header information. If you scroll down on the trace output, you will see that now there is a Form collection, which is used by ASP.NET to provide access to the form data in the case of a POST method.

By comparing the output of these two examples, you can clearly see the difference between GET and POST, and you can also see the data structures used by ASP.NET to make it easy for you to extract data from HTTP requests.

HttpResponse Class

The HttpResponse class encapsulates HTTP response information that is built as part of an ASP.NET operation. The Framework uses this class when it is creating a response that includes writing server controls back to the client. Your own server code may also use the Write method of the Response object to write data to the output stream that will be sent to the client. We have already seen many illustrations of Response.Write.

REDIRECT

The HttpResponse class has a useful method, Redirect, that enables server code to redirect an HTTP request to a different URL. A simple redirection without passing any data is trivial—you need only call the Redirect method and pass the URL. An example of such usage would be a reorganization of a

Web site, where a certain page is no longer valid and the content has been moved to a new location. You can keep the old page live by simply redirecting traffic to the new location.

It should be noted that redirection always involves an HTTP GET request, like following a simple link to a URL. (POST arises as an option when submitting form data, where the action can be specified as GET or POST.) A more interesting case involves passing data to the new page. One way to pass data is to encode it in the query string. You must preserve standard HTTP conventions for the encoding of the query string. The class HttpUtility provides a method UrlEncode, which will properly encode an individual item of a query string. You must yourself provide code to separate the URL from the query string with a "?" and to separate items of the query string with "&".

The folder Hotel provides an example of a simple Web application that illustrates this method of passing data in redirection. The file default.aspx provides a form for collecting information to be used in making a hotel reservation. The reservation itself is made on the page Reservation1.aspx. You may access the starting default.aspx page through the URL

http://localhost/Chap14/Hotel/

As usual, we provide a link to this page in our home page of example programs. Figure 14–14 illustrates the starting page of our simple hotel reservation example.

Figure 14-14 Starting page for making a hotel reservation.

Here is the script code that is executed when the Make Reservation button is clicked.

```
e As EventArgs)
  Dim query As String = "City=" & _
      HttpUtility.UrlEncode(txtCity.Text)
  query += "&Hotel=" & _
      HttpUtility.UrlEncode(txtHotel.Text)
  query += "&Date=" & _
      HttpUtility.UrlEncode(txtDate.Text)
  query += "&NumberDays=" & _
      HttpUtility.UrlEncode(txtNumberDays.Text)
  Response.Redirect("Reservation1.aspx?" + query)
End Sub
```

We build a query string, which gets appended to the Reservation1.aspx URL, separated by a "?". Note the ampersand that is used as a separator of items in the query string. We use the HttpUtility.UrlEncode method to encode the individual items. Special encoding is required for the slashes in the date and for the space in the name San Jose. Clicking the button brings up the reservation page. You can see the query string in the address window of the browser. Figure 14–15 illustrates the output shown by the browser.

Figure 14-15 Browser output from making a hotel reservation

Our program does not actually make the reservation; it simply prints out the parameters passed to it.

```
<%@ Page language="VB" Debug="true" Trace="false" %>
<script runat="server">
    Sub Page_Load(sender As Object, e As EventArgs)
        Response.Write("Making reservation for ...")
        Response.Write("<br>")
        Dim city As String = Request.Params("City")
        Response.Write("City = " & city)
        Response.Write("<br>")
        Dim hotel As String = Request.Params("Hotel")
        Response.Write("Hotel = " & hotel)
        Response.Write("<br>")
        Dim strDate As String = Request.Params("Date")
        Response.Write("Date = " & strDate)
        Response.Write("<br>")
        Dim strDays As String = Request.Params("NumberDays")
        Response.Write("NumberDays = " & strDays)
        Response.Write("<br>")
   End Sub
</script>
<HTML>
<body>
</body>
</HTML>
```

State Management Techniques in ASP.NET

This article discusses various options for state management for web applications developed using ASP.NET. Generally, web applications are based on stateless HTTP protocol which does not retain any information about user requests. In typical client and server communication using HTTP protocol, page is created each time the page is requested.

Developer is forced to implement various state management techniques when developing applications which provide customized content and which "remembers" the user.

Here we are here with various options for ASP.NET developer to implement state management techniques in their applications. Broadly, we can classify state management techniques as client side state management or server side state management. Each technique has its own pros and cons. Let's start with exploring client side state management options.

Client side State management Options:

ASP.NET provides various client side state management options like Cookies, QueryStrings (URL), Hidden fields, View State and Control state (ASP.NET 2.0). Let's discuss each of client side state management options.

Bandwidth should be considered while implementing client side state management options because they involve in each roundtrip to server. Example: Cookies are exchanged between client and server for each page request.

Cookie:

A cookie is a small piece of text stored on user's computer. Usually, information is stored as name-value pairs. Cookies are used by websites to keep track of visitors. Every time a user visits a website, cookies are retrieved from user machine and help identify the user.

Let's see an example which makes use of cookies to customize web page.

```
if (Request.Cookies["UserId"] != null)
    lbMessage.text = "Dear" + Request.Cookies["UserId"].Value + ", Welcome to
our website!";
else
    lbMessage.text = "Guest,welcome to our website!";
```

If you want to store client's information use the below code

```
Response.Cookies["UserId"].Value=username;
```

Advantages:

- Simplicity

Disadvantages:

- Cookies can be disabled on user browsers

- Cookies are transmitted for each HTTP request/response causing overhead on bandwidth
- Inappropriate for sensitive data

Hidden fields:

Hidden fields are used to store data at the page level. As its name says, these fields are not rendered by the browser. It's just like a standard control for which you can set its properties. Whenever a page is submitted to server, hidden fields values are also posted to server along with other controls on the page. Now that all the asp.net web controls have built in state management in the form of view state and new feature in asp.net 2.0 control state, hidden fields functionality seems to be redundant. We can still use it to store insignificant data. We can use hidden fields in ASP.NET pages using following syntax

```
protected System.Web.UI.HtmlControls.HtmlInputHidden Hidden1;

//to assign a value to Hidden field
Hidden1.Value="Create hidden fields";
//to retrieve a value
string str=Hidden1.Value;
```

Advantages:

- Simple to implement for a page specific data
- Can store small amount of data so they take less size.

Disadvantages:

- Inappropriate for sensitive data
- Hidden field values can be intercepted(clearly visible) when passed over a network

View State:

View State can be used to store state information for a single user. View State is a built in feature in web controls to persist data between page post backs. You can set View State on/off for each control using **EnableViewState** property. By default, **EnableViewState** property will be set to true. View state mechanism poses performance overhead. View state information of all the controls on the page will be submitted to server on each post back. To reduce performance penalty, disable View State for all the controls for which you don't need state. (Data grid usually doesn't need to maintain state). You can also disable View State for the entire page by adding **EnableViewState=false** to @page directive. View state data is encoded as binary Base64 - encoded which add approximately 30% overhead. Care must be taken to ensure view state for a page is smaller in size. View State can be used using following syntax in an ASP.NET web page.

```
// Add item to ViewState
ViewState["myviewstate"]  = myValue;

//Reading items from ViewState
Response.Write(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

Query strings:

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**.

Let's assume that we have a Data Grid with a list of products, and a hyperlink in the grid that goes to a product detail page, it would be an ideal use of the Query String to include the product ID in the Query String of the link to the product details page (for example, productdetails.aspx?productid=4).

When product details page is being requested, the product information can be obtained by using the following codes:

```
string productid;
productid=Request.Params["productid"];
```

Advantages:

- Simple to Implement

Disadvantages:

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

Server Side State management:

As name implies, state information will be maintained on the server. Application, Session, Cache and Database are different mechanisms for storing state on the server.

Care must be taken to conserve server resources. For a high traffic web site with large number of concurrent users, usage of sessions object for state management can create load on server causing performance degradation

Application object:

Application object is used to store data which is visible across entire application and shared across multiple user sessions. Data which needs to be persisted for entire life of application should be stored in application object.

In classic ASP, application object is used to store connection strings. It's a great place to store data which changes infrequently. We should write to application variable only in application_Onstart event (global.asax) or application.lock event to avoid data conflicts. Below code sample gives idea

```
Application.Lock();
Application("mydata")="mydata";
Application.UnLock();
```

Session object:

Session object is used to store state specific information per client basis. It is specific to particular user. Session data persists for the duration of user session you can store session's data on web server in different ways. Session state can be configured using the <session State> section in the application's web.config file.

Configuration information:

```
<sessionState mode = <"inproc" | "sqlserver" | "stateserver">
 cookieless = <"true" | "false">
 timeout = <positive integer indicating the session timeout in minutes>
 sqlconnectionstring = <SQL connection string that is only used in the SQLServer
mode>
 server = <The server name that is only required when the mode is State
Server>
 port = <The port number that is only required when the mode is State Server>
```

Mode:

This setting supports three options. They are InProc, SQLServer, and State Server

Cookie less:

This setting takes a Boolean value of either true or false to indicate whether the Session is a cookie less one.

Timeout:

This indicates the Session timeout vale in minutes. This is the duration for which a user's session is active. Note that the session timeout is a sliding value; Default session timeout value is 20 minutes

SqlConnectionString:

This identifies the database connection string that names the database used for mode SQLServer.

Server:

In the out-of-process mode State Server, it names the server that is running the required Windows NT service: aspnet_state.

Port:

This identifies the port number that corresponds to the server setting for mode State Server. Note that a port is an unsigned integer that uniquely identifies a process running over a network.

*You can disable session for a page using **EnableSessionState** attribute. You can set off session for entire application by setting mode=off in web.config file to reduce overhead for the entire application.*

Session state in ASP.NET can be configured in different ways based on various parameters including scalability, maintainability and availability

- In process mode (in-memory)- State information is stored in memory of web server
- Out-of-process mode- session state is held in a process called aspnet_state.exe that runs as a windows service.
- Database mode â€" session state is maintained on a SQL Server database.

In process mode:

This mode is useful for small applications which can be hosted on a single server. This model is most common and default method to store session specific information. Session data is stored in memory of local web server

Configuration information:

```
<sessionState mode="Inproc"
 sqlConnectionString="data source=server;user
id=freelance;password=freelance"
 cookieless="false" timeout="20" />
```

Advantages:

- Fastest mode
- Simple configuration

Disadvantages:

- Session data will be lost if the worker process or application domain recycles
- Not ideal for web gardens and web farms

Out-of-process Session mode (state server mode):

This mode is ideal for scalable and highly available applications. Session state is held in a process called aspnet_state.exe that runs as a windows service which listens on TCP port 42424 by default. You can invoke state service using services MMC snap-in or by running following net command from command line.

Net start aspnet_state

Configuration information:

```
<sessionState mode="StateServer"
 StateConnectionString="tcpip=127.0.0.1:42424"
 sqlConnectionString="data source=127.0.0.1;user id=freelance;
password=freelance"
 cookieless="false" timeout="20"/>
```

Advantages:

- Supports web farm and web garden configuration
- Session data is persisted across application domain recycles. This is achieved by using separate worker process for maintaining state

Disadvantages:

- Out-of-process mode provides slower access compared to In process
- Requires serializing data

SQL-Backed Session state:

ASP.NET sessions can also be stored in a SQL Server database. Storing sessions in SQL Server offers resilience that can serve sessions to a large web farm that persists across IIS restarts.

SQL based Session state is configured with aspnet_regsql.exe. This utility is located in .NET Framework's installed directory C:\<windows>\microsoft.net\framework\<version>. Running this utility will create a database which will manage the session state.

Configuration Information:

```
<sessionState mode="SQLServer"
  sqlConnectionString="data source=server;user
id=freelance;password=freelance"
  cookieless="false" timeout="20" />
```

Advantages:

- Supports web farm and web garden configuration
- Session state is persisted across application domain recycles and even IIS restarts when session is maintained on different server.

Disadvantages:

- Requires serialization of objects

Choosing between client side and Server side management techniques is driven by various factors including available server resources, scalability and performance. We have to leverage both client side and server side state management options to build scalable applications.

When leveraging client side state options, ensure that little amount of insignificant information is exchanged between page requests.

Various parameters should be evaluated when leveraging server side state options including size of application, reliability and robustness. Smaller the application, In process is the better choice. We should account in the overheads

involved in serializing and deserializing objects when using State Server and Database based session state. Application state should be used religiously.

The GridView

The GridView is an extremely flexible grid control that displays a multicolumn table. Each record in your data source becomes a separate row. Each field in the record becomes a separate column.
This functionality includes features for automatic paging, sorting, selecting, and editing. The GridView is also the only data control that can show more than one record at a time.

Defining Columns

By default, the GridView.AutoGenerateColumns property is True, and the GridView creates a column for each field. This automatic column generation is good for creating
quick test pages, but it doesn't give you the flexibility you'll usually want. For example, what if you want to hide columns, change their order, or configure some aspect of their display, such as the formatting or heading text? In all these cases, you need to set AutoGenerateColumns to False and define the columns in the <Columns> section of the GridView control tag.

Table 15-1. ***Column Types***

Column Description

BoundField: This column displays text from a field in the data source.
ButtonField : This column displays a button for each item in the list.
CheckBoxField: This column displays a check box for each item in the list. It's used automatically for True/False fields (in SQL Server, these are fields that use the bit data type).
CommandField: This column provides selection or editing buttons.
HyperlinkField: This column displays its contents (a field from the data source or static text) as a hyperlink.
ImageField: This column displays image data from a binary field (providing it can be successfully interpreted as a supported image format).
TemplateField: This column allows you to specify multiple fields, custom controls, and arbitrary HTML using a custom template. It gives you the highest degree of control but requires the most work.
Following are some important properties that are very useful.

Behavior Properties of the GridView Control	
AllowPaging	true/false. Indicate whether the control should support paging.
AllowSorting	true/false. Indicate whether the control should support sorting.
SortExpression	Gets the current sort expression (field name) that determines the order of the row.
SortDirection	Gets the sorting direction of the column sorted currently (Ascending/Descending).
DataSource	Gets or sets the data source object that contains the data to populate the control.
DataSourceID	Indicate the bound data source control to use (Generally used when we are using SqlDataSource or AccessDataSource to bind the data, See 1st Grid example).
AutoGenerateEditButton	true/false. Indicates whether a separate column should be added to edit the record.

AutoGenerateDeleteButton	true/false. Indicates whether a separate column should be added to delete the record.
AutoGenerateSelectButton	true/false. Indicate whether a separate column should be added to selecat a particular record.
AutoGenerateColumns	true/false. Indicate whether columns are automatically created for each field of the data source. The default is true.
Style Properties of the GridView Control	
AlternatingRowStyle	Defines the style properties for every alternate row in the GridView.
EditRowStyle	Defines the style properties for the row in EditView (When you click Edit button for a row, the row will appear in this style).
RowStyle	Defines the style properties of the rows of the GridView.
PagerStyle	Defines the style properties of Pager of the GridView. (If AllowPaging=true, the page number row appears in this style)
EmptyDataRowStyle	Defines the style properties of the empty row, which appears if there is no records in the data source.
HeaderStyle	Defines the style properties of the header of the GridView. (The column header appears in this style.)
FooterStyle	Defines the style properties of the footer of GridView.
Appearance Properties of the GridView Control	
CellPadding	Indicates the space in pixel between the cells and the border of the GridView.
CellSpacing	Indicates the space in pixel between cells.
GridLines	Both/Horizontal/Vertical/None. Indicates whether GridLines should appear or not, if yes Horizontal, Vertical or Both.
HorizontalAlign	Indicates the horizontal align of the GridView.
EmptyDataText	Indicates the text to appear when there is no record in the data source.
ShowFooter	Indicates whether the footer should appear or not.
ShowHeader	Indicates whether the header should appear or not. (The column name of the GridView)
BackImageUrl	Indicates the location of the image that should display as a background of the GridView.
Caption	Gets or sets the caption of the GridView.
CaptionAlign	left/center/right. Gets or sets the horizontal position of the GridView caption.
State Properties of GridView Control	
Columns	Gets the collection of objects that represent the columns in the GridView.
EditIndex	Gets or sets the 0-based index that identifies the row

	currently to be edited.
FooterRow	Returns a GridViewRow object that represents the footer of the GridView.
HeaderRow	Returns a GridViewRow object that represents the header of the GridView.
PageCount	Gets the number of the pages required to display the reocrds of the data source.
PageIndex	Gets or sets the 0-based page index.
PageIndex	Gets or sets the number of records to display in one page of GridView.
Rows	Gets a collection of GridViewRow objects that represents the currently displayed rows in the GridView.
DataKeyNames	Gets an array that contains the names of the primary key field of the currently displayed rows in the GridView.
DataKeys	Gets a collection of DataKey objects that represent the value of the primary key fields set in DataKeyNames property of the GridView.
Events associated with GridView Control	
PageIndexChanging, PageIndexChanged	Both events occur when the page link is clicked. They fire before and after GridView handles the paging operation respectively.
RowCancelingEdit	Fires when Cancel button is clicked in Edit mode of GridView.
RowCommand	Fires when a button is clicked on any row of GridView.
RowCreated	Fires when a new row is created in GridView.
RowDataBound	Fires when row is bound to the data in GridView.
RowDeleting,RowDeleted	Both events fires when Delete button of a row is clicked. They fire before and after GridView handles deleting operaton of the row respectively.
RowEditing	Fires when a Edit button of a row is clicked but before the GridView hanldes the Edit operation.
RowUpdating, RowUpdated	Both events fire when a update button of a row is clicked. They fire before and after GridView control update operation respectively.
Sorting, Sorted	Both events fire when column header link is clicked. They fire before and after the GridView handler the Sort operation respectively.

For Eample:

```
<asp:GridView ID="GridView1" runat="server" DataSourceID="SqlDataSource1"
AllowPaging="True" AllowSorting="True" AutoGenerateEditButton="true"
PageSize="8">
    <Columns>
<asp:BoundField DataField="name" HeaderText="name" SortExpression="name" />
    </Columns>
```

```
</asp:GridView>
```

Themes

One of the neat features of ASP.NET 2.0 is themes, which enable you to define the appearance of a set of controls once and apply the appearance to your entire web application. For example, you can utilize themes to define a common appearance for all of the CheckBox controls in your application, such as the background and foreground color, in one central location. By leveraging themes, you can easily create and maintain a consistent look throughout your web site. Themes are extremely flexible in that they can be applied to an entire web application, to a page, or to an individual control. Theme files are stored with the extension .skin, and all the themes for a web application are stored in the special folder named App_Themes.

The implementation of themes in ASP.NET 2.0 is built around two areas: skins and themes. A skin is a set of properties and templates that can be applied to controls. A theme is a set of skins and any other associated files (such as images or stylesheets). Skins are control-specific, so for a given theme there could be a separate skin for each control within that theme. Any controls without a skin inherit the default look. There are two types of themes:

- **Customization themes**: These types of themes are applied after the properties of the control are applied, meaning that the properties of the themes override the properties of the control itself.
- **Stylesheet themes**: You can apply this type of theme to a page in exactly the same manner as a customization theme. However, stylesheet themes don't override control properties, thus allowing the control to use the theme properties or override them.

Characteristics of ASP.NET 2.0 Themes

Some of the important characteristics of ASP.NET 2.0 themes are:

- Themes make it simple to customize the appearance of a site or page using the same design tools and methods used when developing the page itself, thus obviating the need to learn any special tools or techniques to add and apply themes to a site.
- As mentioned previously, you can apply themes to controls, pages, and even entire sites. You can leverage this feature to customize parts of a web site while retaining the identity of the other parts of the site.
- Themes allow all visual properties to be customized, thus ensuring that when themed, pages and controls can achieve a consistent style.
- Customization themes override control definitions, thus changing the look and feel of controls. Customization themes are applied with the Theme attribute of the Page directive.
- Stylesheet themes don't override control definitions, thus allowing the control to use the theme properties or override them. Stylesheet themes are applied with the StylesheetTheme attribute of the Page directive.

Now that you have an understanding of the concepts behind themes, the next section provides you with a quick example of creating a theme and utilizing it from an ASP.NET page.

Creating a Simple Theme

To create a theme and apply it to a specific page, go through the following steps:

1. Create a folder called ControlThemes under the App_Themes folder.
2. Create a file with the extension .skin and add all the controls (that you want to use in a page) and their style properties. Or you can also create individual skin files for each and every control. When you are defining skin files, remember to remove the ID attribute from all of the controls' declarations. For example, you can use the following code to define the theme for a Button control:
   <asp:Button runat="server" BackColor="Black" ForeColor="White" Font-Name="Arial" Font-Size="10px" />
3. Name the skin file Button.skin and place it under the ControlThemes folder. Once you have created the .skin file, you can then apply that theme to all the pages in your application by using appropriate settings in the Web.config file. To apply the theme to a specific page, all you need to do is to add the Theme attribute to the Page directive as shown below:
   <%@Page Theme="ControlThemes"%>

That's all there is to creating a theme and utilizing it in an ASP.NET page. It is also possible for you to programmatically access the theme associated with a specific page using the Page.Theme property. Similarly, you can also set the SkinID property of any of the controls to specify the skin. If the theme does not contain a SkinID value for the control type, then no error is thrown and the control simply defaults to its own properties. For dynamic controls, it is possible to set the SkinID property after they are created.

Themes:

An ASP.NET Theme enables you to apply a consistent style to the pages in your website. You can use a Theme to control the appearance of both the HTML elements and ASP.NET controls that appear in a page.

You create a Theme by adding a new folder to a special folder in your application named App_Themes. Each folder that you add to the App_Themes folder represents a different Theme. If the App_Themes folder doesn't exist in your application, then you can create it. It must be located in the root of your application.

A Theme folder can contain a variety of different types of files, including images and text files. You also can organize the contents of a Theme folder by adding multiple subfolders to a Theme folder. The most important types of files in a Theme folder are:-

- Skin Files.
- Cascading Style Sheet Files

A Theme can contain one or more Skin files. A Skin enables you to modify any of the proprieties of an ASP.NET control that have an effect on its appearance.

For Example, Imagine that you want to show every label in the application to appear with a yellow background and red color in text. You can create a folder in the App_Themes folder named **Default**. Under this folder create a new skin file named **Label.Skin**

In the skin file enter the code as in Listing 1.1

Listing 1.1
App_Themes\Default\Label.Skin

```
<asp:Label BackColor="Yellow" Font-Bold="true" Font-Names="Verdana" ForeColor="red" runat="server" />
```

To use that skin in the pages in a website just set the Theme Property in the Page directive. For example

```
<%@ Page Language="vb" AutoEventWireup="true" CodeFile="UsingTheme.aspx.vb" Inherits="UsingTheme" Theme="Default" %>
```

Rather than add the Themes attribute to each and every page to which you want to apply Theme, you can register a Theme for all pages in your application in the **web configuration file.** For Example

```
<?xml version="1.0"?>
<configuration>
        <system.web>
          <pages theme="default">
          </pages>
        </system.web>
 </configuration>
```

TreeView

The TreeView control is the most impressing new control in ASP.NET 2.0. It is used to display the hierarchical data in tree view format. It also supports dynamic population of the node on demand without page refresh.
When TreeView is displayed for the first time, it displays all its nodes. However, it can be controlled by setting ExpandDepth property.

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 TreeView Control	
DataSourceID	Indicates the data source to be used (You can use .sitemap file as datasource).
Text	Indicates the text to display in the node.
Tooltip	Indicates the tooltip of the node 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 to the user when node is clicked. If not set you can handle TreeView.SelectedNodeChanged 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).
ImageUrl	Indicates the image that appears next to the node.
ImageToolTip	Indicates the tooltip text to display for image next to the node.
Styles of TreeView Control	
NodeSpacing	Space (in pixel) between current node and the node above or below it.
VerticalPadding	Space (in pixel) between the top and bottom of the node text.
HorizontalPadding	Space (in pixel) between the left and right of the node text.
ChildNodePadding	Space (in pixel) between the parent node and its child node.

DEMO : TreeView

- Home
- Tutorials
  - ASP.NET Tutorials
  - ASP.NET AJAX Tutorials
  - SiteMapDataSource control tutorials


```
// TreeView Control ////////////////////////////
<asp:TreeView ID="TreeView1" runat="Server"
DataSourceID="SiteMapDataSource1" ImageSet="Simple"
             ExpandDepth="1">
            <ParentNodeStyle Font-Bold="False" />
            <HoverNodeStyle Font-Underline="True" ForeColor="#5555DD"
/>
            <SelectedNodeStyle Font-Underline="True"
ForeColor="#5555DD" HorizontalPadding="0px"
                VerticalPadding="0px" />
            <NodeStyle Font-Names="Tahoma" Font-Size="10pt"
ForeColor="Black" HorizontalPadding="0px"
                NodeSpacing="0px" VerticalPadding="0px" />
        </asp:TreeView>


// SiteMapDataSource Control ////////////////////////////
   <asp:SiteMapDataSource ID="SiteMapDataSource1" runat="Server" />
```

User Control in ASP.NET

User controls are one of ASP.NET methods to increase reusability of code, implement encapsulation and reduce maintenance. User control is similar to web page. Both web pages and user controls contain HTML elements and markup for web controls. Some tags, like <html>, <head> or <form> cannot be used in web user controls.

User controls are saved with .ascx extension, instead of .aspx for web pages. User controls can use code behind feature like web pages. Code behind file of web user controls ends with .ascx.vb. Markup code of user controls starts with @Control directive. Simplest web user control could contains only static HTML, like in this example:

```
<%@ Control Language="VB" CodeFile="WebUserControl.ascx.vb"
Inherits="WebUserControl" %>
<p>Hello, I am very simple <b>web user control</b> with static HTML only.</p>
```

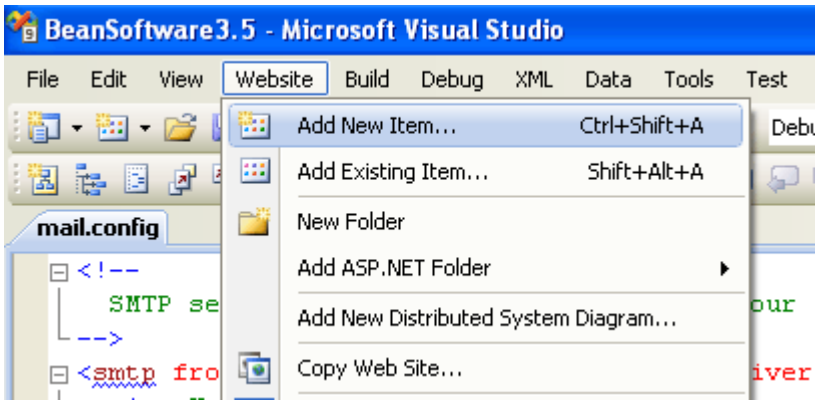
This web user control could be used when you need to repeat same code on many pages. Good examples are page footer, header or some kind of site navigation. Except static HTML code, web user controls could contain web controls. You can simply drag and drop items from toolbox to user control. Like any other control, user controls can have properties, methods and events which make them very useful.

Advantages

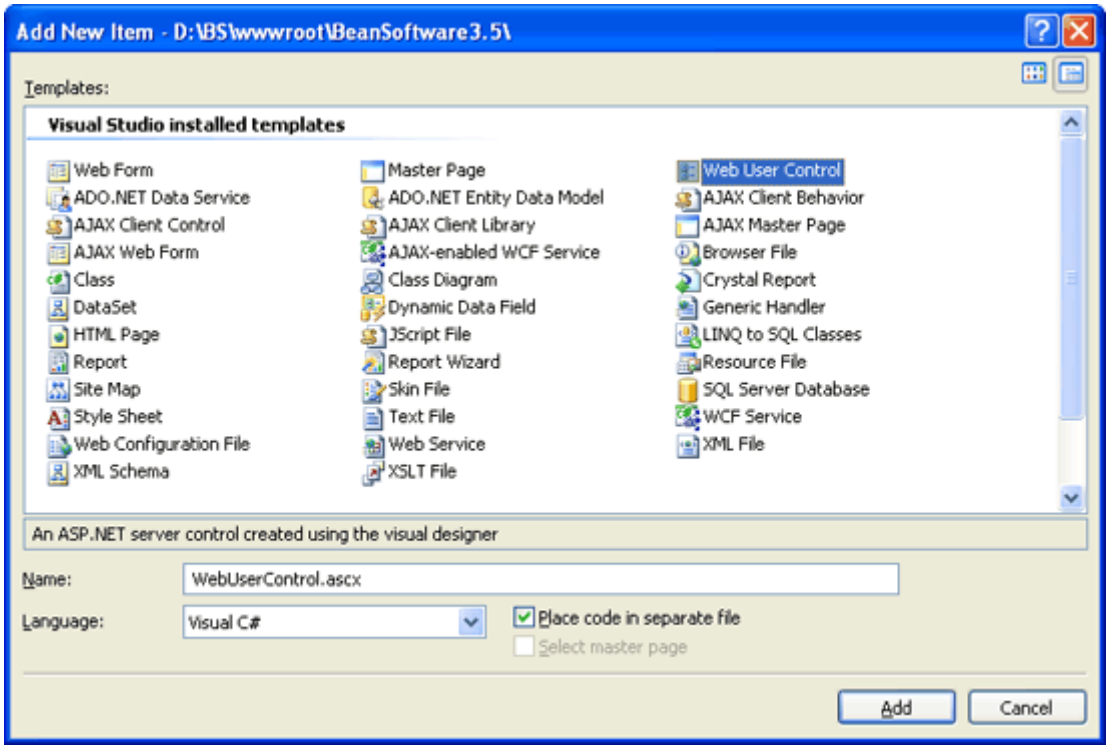
If user control solves more general problem, like progress bar or data pager control then can be useful on many different web sites. On this way, user controls can provide more stable and tested code.

How to create web user control

All user controls are derived from the **System.Web.UI.UserControl** class. UserControl and Page classes both inherits from TemplateControl class. You can add web user control to existing web site project, in Visual Studio menu go to WebSite -> Add New Item..., like in image bellow



New dialog will appear. To create new web user control, select web user control template, like in next image. Choose a name for your control and click Add.



How to add web user control to ASP.NET web page

To add web user control to web page, we need two lines. First, we need to register user control with **Register directive** on top of the page:

```
<%@ Page Language="VB" AutoEventWireup="true"Â  CodeFile="Default.aspx.vb" Inherits="_Default" %>
<%@ Register tagprefix="bean" Tagname="footer" src="~/footer.ascx" %>
```

With Register directive we define user control on page. The important parameters are:

TagPrefix: prefix for user control tag. This prefix will be used in markup code later.

TagName: tag name of user control tag.

Src: path to .ascx file.

After registration with **Register directive**, you can place user control inside web form with tag formatted as <tagprefix:tagname, like this:

```
<bean:footer runat="server" id="MyUserControl" />
```

Registering user controls in web.config

If you'll use user control on many pages on web site then you can register control in web.config file. On this way you can use user control in complete web site without need to register it on every page. You can do this in <controls > section with code like this:

```
<configuration>
  <system.web>
    <pages>
      <controls>
        <add src="Footer.ascx" tagPrefix="bs" tagName="footer" />
      </controls >
    </pages >
  </system.web>
</configuration>
```

Inside the ASP.NET user profile mechanism

Most applications looked the same to any connected user, and HTTP cookies were available to those applications needing to save some user-related data and preferences. An HTTP cookie is a small amount of information sent to the client through response headers. Browsers optionally save this information in an isolated region of the client's hard disk, and send it back with any requests directed at the same originating URL. Internally, a cookie is made of key/value pairs where every piece of information is expressed as raw text. Cookies have two main drawbacks: their limited size, which is seldom greater than 8k, and the control that end users exercise over them. Cookies can in fact be disabled by users, thus making totally ineffective any application features built on top of them.

In either case, an effective personalization layer can't be based on cookies, and can't be left to each team of developers for a custom implementation.

The profile API in ASP.NET 2.0 is Microsoft's answer to the demand for an effective personalization layer built into the ASP.NET framework. With the profile API, developers have no need to resort to the session state or cookies to persist user-specific information across sessions and regardless of possible server failures. Developers just write their pages to use a new HTTP context object – the Profile object – and the object does the magic of reading and writing user information to a persistent store of choice. In this article, we'll look at the internal implementation of the ASP.NET Profile object, its relationship with anonymous identification, and the role of profile providers.

The user profile explained

In the abstract, the ASP.NET user profile is a collection of typed properties grouped into a dynamically generated class. While cookies are a name/value collection of text-based data, a user profile is a class with any number of typed properties that the developer is free to define on a per-application basis. In other words, to store a colour and a number in a cookie you create two untyped entries in a collection. You write the information as raw text with the serialization and deserialization process:

```
HttpCookie cookie = new
        HttpCookie("UserSettings");
cookie.Values["BackgroundColor"]
        = color.Name;
cookie.Values["RowsToDisplay"]
        = rowCount.ToString();
Response.Cookies.Add(cookie);
```

To store the same pieces of information in the ASP.NET Profile object you set properties on the Profile objects, as below:

```
Profile.BackgroundColor = color;
Profile.RowsToDisplay = rowCount;
```

In this example, the BackgroundColor and RowsToDisplay properties are strongly typed and defined by the developer, and shared by all sessions in the application. Each application may have its own profile object with a different set of members, but who's in charge of defining the layout of the profile class? The layout of the user profile is defined in the configuration file (web.config), and consists of a list of typed properties. Each property can be of any .NET common language runtime (CLR) type, including collections, arrays, and user-defined types. In the web.config file, you just store the description of the profile object as you want it to be. You list property names and types, and also may optionally leave there suggestions on how the ASP.NET runtime has to serialize and deserialize property values. Here's a quick example of the profile section in the web.config file:

```
<profile>
        <properties>
        <add name="Theme" type="String" />
        <add name="Rows" type="Integer" />
        <add name="Favorites"
                type="System.Collections.
        Specialized.StringCollection" />
        </properties>
</profile>
```

The <profile> section is located under <system.web>. The <properties> section lists all properties that form the user profile's data model. The contents of this portion of the web.config file are parsed when the application starts up, and originate a dynamic class. The dynamically created profile class is named ProfileCommon and derives from ProfileBase. Here's an example:

```
namespace ASP {
        public class ProfileCommon :
                ProfileBase {
                public virtual string Theme {
                        get {(string)
                        GetPropertyValue("Theme");}
                        set {SetPropertyValue(
                                "Theme", value);}
                }
                // Other properties in the
                // web.config are implemented
                // here
                public virtual ProfileCommon
                        GetProfile(string username) {
                        object o =
                        ProfileBase.Create(username);
                        return (ProfileCommon) o;
                }
        }
}
```

The ProfileBase class is a .NET Framework class that incorporates all the logic required to manage the I/O of user profiles. An instance of the ProfileCommon class is exposed out of the HTTP context through the Profile property. All ASP.NET applications have a Profile object, but each application will be bound to a distinct profile object. The definition of the ProfileCommon class is based on the contents of the <profile> section, and may differ on a per-application basis.

All profile classes, though, share a certain behaviour – the behaviour provided by the parent ProfileBase class. This class provides methods to retrieve and update the profile and manage the persistence of the user profile. An instance of the profile object is associated with each authenticated user that happens to use the application. As the user interacts with the application, the profile may be updated. At the end of each request, the state of the profile is persisted to the data store. When a new request comes in from the same user, the profile is retrieved from the data store and attached to the context of the request. This behaviour is built into the ProfileBase class.

Managing the persistence

Page authors don't have to worry about profile persistence. The ASP.NET runtime environment knows how to retrieve the user's profile and knows where to save the modified profile when the request terminates. The overall functionality is not surprising per se. On the other hand, isn't it through a similar mechanism that session state has always been retrieved in web applications?

In ASP.NET, a special component named the "profile provider" is installed to receive I/O instructions from the ASP.NET runtime, and to save and read the profile of a given user. More interestingly, the default profile I/O provider can be unplugged and replaced with a custom one that uses a customized data store such as an Oracle database or an XML file. By default, however, the profile information goes to a fixed SQL Server Express database.

Any personalization data is persisted on a per-user basis and is permanently stored until someone with administrative privileges deletes it. The data storage is hidden from the user and, to some extent, from the programmer. The user doesn't need to know how and where the data is stored; the programmer simply needs to indicate which profile provider he wants to use. Then, the personalization provider determines the data store to use. Let's take a closer look at the request lifecycle as far as the profile management is concerned. **Figure 1** provides a graphical view of the role played by the profile provider.

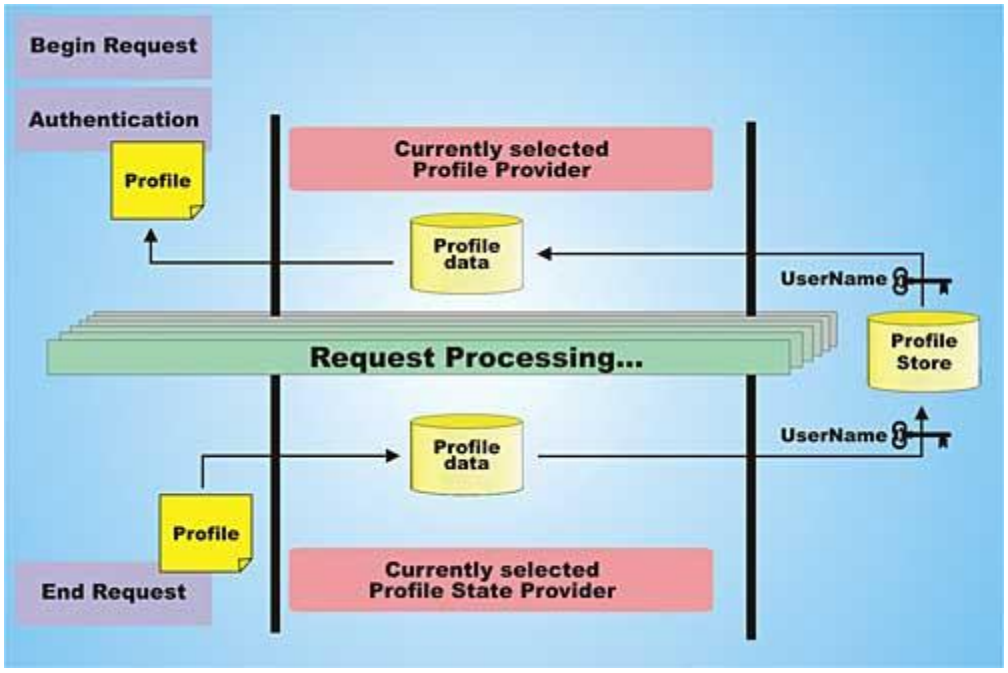


Figure 1: The role of the profile provider in the request lifecycle

The ASP.NET pipeline puts a special HTTP module in charge of the user profile functionality. The profile module kicks in once the request has been authenticated. Using the identity of the user as a key, the module interrogates the registered profile provider to get the profile object for the current user. The HTTP context object associated with the pending request initially exposes an empty instance of the ProfileCommon class with the application-specific layout – the HttpContext.Profile property. The profile provider deserializes any data found in the storage medium, be it a SQL Server database, an XML file, or even an HTTP cookie. Next, the provider stores any information in the profile property on the HttpContext object.

From now on, the code attached to the page can consume this data using the Profile member on the HttpContext class. Any property registered through the web.config data model can be addressed in a strongly-typed manner. The ProfileCommon class, as defined by the development team, is definitely part of the project and can be accessed directly and without intermediate filters. When the request is going to terminate, the profile HTTP module kicks in again and orders the profile provider to serialize the status of the profile object for the current user.

The default profile provider is named AspNetSqlProfileProvider, and uses SQL Server 2005 Express as the data storage. Providers are registered in the <providers> section of the configuration file under the main node <profile>, as in the following configuration script excerpted from the machine.config file:

```
<profile defaultProvider=
        ”AspNetSqlProfileProvider”>
        <providers>
        <add name=”AspNetSqlProfileProvider”
```

```
connectionStringName="LocalSqlServer"
                        applicationName="/"
                type=
"System.Web.Profile.SqlProfileProvider"
                />
        </providers>
</profile>
```

The list of <add> nodes within the <providers> section details the currently registered providers. The name and type attributes are common to all types of providers and indicate the name and class information for the specified provider. Other properties are part of the provider's specific configuration mechanism. The description attribute gives the expected behaviour of the provider, and the connectionStringName contains the information needed to set up a connection with the data store. For the default provider, it doesn't have to contain a plain connection string, but only the name of an entry in the <connectionStrings> section of the configuration file where actual connection information is stored.

A custom profile provider is a class that inherits from the base class ProfileProvider and overrides a handful of methods. Once you have created a similar class, you register it with the application by adding a new <add> section to the above script. To select the default provider, you add a defaultProvider attribute to the <profile> section. The default provider stores profile information to the aspnetdb.mdf database that you can create on the development machine using the ASP.NET Web Site Administration Tool (WSAT), fully integrated in Visual Studio .NET 2005. The database is located in the App_Data special folder of the ASP.NET application and has a fixed set of tables and schema. Note that the same database contains tables to hold membership and roles information. You then deploy the database to the production machine as part of the application setup.

Inside the profile data model

When defining the profile data model in the web.config file, you can use a number of predefined attributes to tune the behaviour of the ProfileCommon class. The profile data model is the set of properties you want to define on the dynamically created ProfileCommon class. The <add> section under the <properties> section can be decorated with any combination of the attributes in **Table 1**. Note that Name and Type are the only mandatory attributes.

Table 1: Attributes for the profile data model

Attribute	Description
Name	Specifies the name of the property.
Type	Specifies the type of the property. Use the "class, assembly" notation in case of user-defined types.
AllowAnonymous	Indicates whether the property is usable by anonymous users.
DefaultValue	Specifies the default value for the property.
Provider	Specifies the name of the custom profile provider object to use to

	manage the property.
ReadOnly	Indicates whether the property is read-only. (Default is false.)
SerializeAs	Indicates how the property should be serialized to the data store: as a string, as a stream of bytes, as XML, or using the specified provider.
CustomProviderData	Specifies any context string to pass to the provider (in case it is not the default one) for initialization purposes.

If the property is assigned a default value, and the property value is not modified during the request processing, then the property is not persisted to the data store. Likewise, a property marked as read-only is not included in any writing operation back to the data store. The value of a property may need serialization to be stored to a persistent data store. By using the SerializeAs attribute, you can decide how this should happen. The feasible options are listed in **Table 2**.

Table 2: Options for profile serialization

Attribute	**Description**
Binary	The value is serialized using binary serialization.
ProviderSpecific	The provider has implicit knowledge of the type of the value and decides how to serialize into the data store. This is the default option.
String	The value is serialized to a string.
Xml	The value is serialized using XML serialization.

You can have multiple profile providers at work for the various profile properties. By default, all properties are saved and read using the currently selected profile provider. However, you can ask the ASP.NET runtime to process a particular property using a specific provider. If this is the case, you just set the Provider attribute to the name of the profile provider to use. The profile provider must be registered with the application.

The <properties> section also accepts a <group> element. The <group> element is meant to group a few related properties as if they are properties of an intermediate object.

```
<properties>
        :
        <group name="Font">
                <add name="Name" type="string"
                        defaultValue="verdana" />
                <add name="SizeInPoints"
                type="int" defaultValue="8" />
        </group>
</properties>
```

To access any of the preceding properties, use the following syntax.

```
Response.Write(Profile.Font.Name);
```

Groups of properties help to better organize the attributes of the data model and can be nested to multiple levels.

Working with the profile object

As mentioned, a system component is responsible for automatically retrieving and saving the profile information. The code-beside attached to ASP.NET pages can access profile information through the Profile property on the HttpContext object. The same object is also mirrored through a property on the page class. The following two instructions are totally equivalent, but the second is a little bit faster:

```
Response.Write(
        Profile.BackgroundColor);
Response.Write(
        Context.Profile.BackgroundColor);
```

To be precise, the Page class has no Profile property defined and you won't find it documented anywhere; yet the first line above works just fine. The point is that a protected Profile property is added to the dynamically created class that is used to serve an ASP.NET request. The ultimate page class used to serve a request has the following piece of code added:

```
protected ProfileCommon Profile {
        get {
                return ((ProfileCommon)(Context.Profile));
}}
```

As you can see, calling the Profile property on the page class returns the same information as the Context object, but requires an extra step.

Part of the request's HTTP context, the profile object is common to all pages visited by the same user. However, each request is attached to an updated version of the object filled with fresh data as modified by previous requests made by the same user in any sessions. It is essential to note that each user has its own copy of the profile object. Any data stored in the profile has no predefined duration and is permanently stored until site administrators delete the information when convenient. A user profile can't be deleted programmatically.

Note that the value associated with the Profile object is never null. When no value is present for a user, an empty instance of the ProfileCommon class is returned. There's just one situation in which the Profile object is null – when the profile functionality is disabled. You enable and disable the profile functionality using the enabled attribute on the <profile> node of the web.config file:

```
<profile enabled="false"> : </profile>
```

While the Profile object is never null by design, the same isn't necessarily true for its child properties. If the profile is expected to contain (for example) a collection property, you have to check the property against null before use:

```
if (Profile.Links != null) DisplayLinks(Profile.Links);
```

To update the value of a property, you simply assign the new value to the property:

```
Profile.Links.Add("http://www.solidqualitylearning.com");
Profile.Links.Add("http://www.vsj.com");
Profile.BackgroundColor = Color.LightYellow;
```

Can you be sure that any modified value is automatically saved to the data store so that it can be retrieved on the next request made by the same user in the same, or another, session? By default, the profile is saved at the end of each request, as long as the profile object is dirty, that is there are pending changes and current values are different from default values of involved properties. You can disable the auto-saving feature by adding an attribute to the <profile> node, as below:

```
<profile automaticSaveEnabled="false"> : </profile>
```

If automatic saving is disabled, you can request a save programmatically by invoking the Profile.Save method.

Anonymous profiles

The profile feature is strictly user-specific; however, ASP.NET also supports profiles for anonymous users. In this context, an anonymous user is simply a user who didn't log in to the ASP.NET application; the feature has nothing to do with the IIS anonymous user and is independent of any type of ASP.NET authentication (Forms, Windows, Passport). To enable anonymous profiles, you set the following script in the web.config file.

```
<anonymousIdentification
        enabled="true" />
```

In this way, a unique identity (typically, a GUID) is associated with each unauthenticated user so that the ASP.NET application can recognize and treat such users as regularly registered users. This approach formalizes a common practice that most developers use in their applications to come up with a simplified logic that doesn't have to distinguish between authenticated and anonymous users. In this way, a role-based UI can be developed, and users will easily pass from one logical role to the next as they log in.

Anonymous users can manage their profile settings, but have access only to those properties explicitly flagged with the allowAnonymous attribute. Needless to say, no anonymous data is ever serialized to the profile data store. How can the application recognize a particular anonymous user when he's back and apply all of his settings? For anonymous users, a cookie is generated and appended to the request. Name, domain, protection, and expiration of the cookie can be tuned in the configuration file.

Weren't profiles a way to replace cookies? Correct, but in this case an alternative and faster mechanism is required to persist data that is less critical than the profile of registered users. However, if you want to avoid cookies, you can opt for a cookieless approach. Here's the full schema of the <anonymousIdentification> section:

```
<anonymousIdentification
        enabled="[true | false]"
        cookieless="[UseUri | UseCookies |
                AutoDetect | UseDeviceProfile]"
        cookieName=".ASPXANONYMOUS"
        cookiePath=""
        cookieProtection="[None |
                Validation | Encryption | All]"
        cookieRequireSSL="[true | false]"
        cookieSlidingExpiration=
                "[true | false]"
```

```
        cookieTimeout="[DD.HH:MM:SS]"
        domain="cookie domain"
/>
```

The cookieless attribute indicates whether cookies are used or not. The effects of all possible choices are shown in **Table 3**.

Table 3: Options for the cookieless attribute

Mode	Description
AutoDetect	Use cookies for anonymous profiles only if the requesting browser supports cookies
UseCookies	Use cookies for anonymous profiles regardless of whether or not the browser supports cookies. This is the default option.
UseDeviceProfile	Base the decision on the browser's capabilities as listed in the device profile section of the configuration file.
UseUri	Store the anonymous profile in the URL regardless of whether the browser supports cookies or not. Use this option if you want to go cookieless no matter what.

The anonymous identification feature is governed by an HTTP module that exposes a Creating event. By handling the event in the global.asax file, you can assign the anonymous user your own ID. If you don't specify a custom ID, a GUID is used. Finally, note that anonymous identification in no way affects the identity of the account that is processing the request.

Migrating anonymous settings

What happens when an anonymous user decides to log in? The profile HTTP module fires an application-wide event named MigrateAnonymous. By handling this event in the global.asax file, an application can easily move all or some of the anonymous settings to the profile of the now registered user. In this case, the settings are then persisted to the data store. An anonymous user that attempts to log in is recognized by the .ASPXANONYMOUS cookie attached to the request. The following listing shows how to migrate anonymous settings to the user profile:

```
void Profile_MigrateAnonymous(
        object sender,
        ProfileMigrateEventArgs e) {
        // Load the profile of the
        // anonymous user
        ProfileCommon anonProfile;
        anonymProfile =
        Profile.GetProfile(e.AnonymousId);

        // Migrate the properties to the
        // new profile
        Profile.Theme = anonymProfile.Theme;
        :
}
```

Role-based profiles

The profile HTTP module may also raise other application-wide events such as Personalize. In particular, this event fires just before the profile data is retrieved. An analogous event – ProfileAutoSaving – is fired just before the profile data is persisted. In both cases, you can take control of the I/O process and alter the data being passed to the application or written to the store. The Personalize event is essential if you want to give users a different profile object based on their role. In this case, you write a global.asax handler for the Personalize event and check the role of the current user:

```
void Profile_Personalize(object sender,
        ProfileEventArgs e) {
        ProfileCommon profile = null;
        if (User == null)
                return;

        // Determine the profile based on
        // the role
        if (User.IsInRole(
                "Administrators"))
                profile = (ProfileCommon)
                        ProfileBase.Create(
                        "Administrator");
        else if (User.IsInRole("Users"))
                profile = (ProfileCommon)
                        ProfileBase.Create("User");

        // Make the HTTP profile module use
        // THIS profile object
        if (profile != null)
                e.Profile = profile;
}
```

Based on the role, you explicitly create a new profile using the name of the role as the key to retrieve profile information from the data store. In other words, you have as many records in the profile store as there are roles supported in your application. Finally, you need to inform the HTTP profile module that you're done, and there's no need for it to read information from the data store. You do this by storing a non-null object in the Profile property of the ProfileEventArgs data structure.

❖ Validation Control

ASP.NET validation controls validate the user input data to ensure that useless, unauthenticated, or contradictory data don't get stored.

ASP.NET provides the following validation controls:

- RequiredFieldValidator
- RangeValidator
- CompareValidator
- CustomValidator
- RegularExpressionValidator
- ValidationSummary

BaseValidator Class

The validation control classes are inherited from the BaseValidator class hence they inherit its properties and methods. Therefore, it would help to take a look at the properties and the methods of this base class, which are common for all the validation controls:

Members	Description
ControlToValidate	Indicates the input control to validate.
Display	Indicates how the error message is shown.
EnableClientScript	Indicates whether client side validation will take.
Enabled	Enables or disables the validator.
ErrorMessage	Indicates error string.
Text	Error text to be shown if validation fails.
IsValid	Indicates whether the value of the control is valid.
SetFocusOnError	It indicates whether in case of an invalid control, the focus should switch to the related input control.
ValidationGroup	The logical group of multiple validators, where this control belongs.
Validate()	This method revalidates the control and updates the IsValid property.

RequiredFieldValidator

RequiredFieldValidator validator control is used to make a field as mandatory in the form. Without filling the field user can't submit the form.

Following are some basic properties of all Validator controls

ControlToValidate	Gets or sets the input control to validate (eg. The ID value of asp:TextBox control).

Display	Dynamic/Static. Used to indicate how the area of error message will be allocated. Dynamic: Error message area will only be allocated when error will be displayed. Static: Error messagea area will be allocated in either case.
Enabled	true/false. Gets or sets whether to enable the validation control or not.
ErrorMessage	Gets or sets the text of the error message that will be displayed when validation fails (This is displayed when ValidationSummary validatoin control is used.).
Text	Gets or sets the description of the error message text.
ValidationGroup	Gets or sets the validation group it belongs to. This is used to group a set of controls.
SetFocusOnError	true/false. Used to move focus on the control that fails the validation.

DEMO : RequiredFieldValidator

Write into TextBox

```
    <asp:RequiredFieldValidator ID="req1" runat="Server"
ControlToValidate="TextBox1" ErrorMessage="TextBox is Mandatory field"
Text="Please write something in the Box."></asp:RequiredFieldValidator>
```

RangeValidator

RangeValidator is used to validate if the given data is in between the specified range or not.

Following are main properties of the validation control.

MinimumValue	Gets or sets the minimum value of the range.
MaximumValue	Gets or sets the maximum value of the range.
Type	Integer/String/Date/Currency/Double. Used to specify the data type to validate.
ControlToValidate	Gets or sets the input control to validate (eg. The ID value of asp:TextBox control).
Display	Dynamic/Static. Used to indicate how the area of error message will be allocated. Dynamic: Error message area will only be allocated when error will be displayed. Static: Error messagea area will be allocated in either case.
Enabled	true/false. Gets or sets whether to enable the validation control or not.

ErrorMessage	Gets or sets the text of the error message that will be displayed when validation fails (This is displayed when ValidationSummary validatoin control is used.).
Text	Gets or sets the description of the error message text.
ValidationGroup	Gets or sets the validation group it belongs to. This is used to group a set of controls.
SetFocusOnError	true/false. Used to move focus on the control that fails the validation.

DEMO : RangeValidator

Write into TextBox

```
                    <asp:RangeValidator ID="range1" runat="Server"
ControlToValidate="TextBox1" MinimumValue="5" MaximumValue="10"
Display="dynamic" Type="Integer" Text="Integer only" ErrorMessage="Value
must be between 5 to 10"></asp:RangeValidator>
```

RegularExpressionValidator

RegularExpressionValidator validation control is used to make sure that a textbox will accept a predefined format of characters. This format can be of any type like you@domain.com (a valid email address).

Following are main properties of the validation control.

ValidationExpression	Gets or sets the regular expression that will be used to validate input control data.
ControlToValidate	Gets or sets the input control to validate (eg. The ID value of asp:TextBox control).
Display	Dynamic/Static. Used to indicate how the area of error message will be allocated. Dynamic: Error message area will only be allocated when error will be displayed. Static: Error messagea area will be allocated in either case.
Enabled	true/false. Gets or sets whether to enable the validation control or not.
ErrorMessage	Gets or sets the text of the error message that will be displayed when validation fails (This is displayed when ValidationSummary validatoin control is used.).
Text	Gets or sets the description of the error message text.
ValidationGroup	Gets or sets the validation group it belongs to. This is used to group a set of controls.

SetFocusOnError	true/false. Used to move focus on the control that fails the validation.

The following table summarizes the commonly used syntax constructs for regular expressions:

Character Escapes	Description
\b	Matches a backspace.
\t	Matches a tab.
\r	Matches a carriage return.
\v	Matches a vertical tab.
\f	Matches a form feed.
\n	Matches a new line.
\	Escape character.

Apart from single character match, a class of characters could be specified that can be matched, called the metacharacters.

Metacharacters	Description
.	Matches any character except \n.
[abcd]	Matches any character in the set.
[^abcd]	Excludes any character in the set.
[2-7a-mA-M]	Matches any character specified in the range.
\w	Matches any alphanumeric character and underscore.
\W	Matches any non-word character.
\s	Matches whitespace characters like, space, tab, new line etc.
\S	Matches any non-whitespace character.
\d	Matches any decimal character.
\D	Matches any non-decimal character.

Quantifiers could be added to specify number of times a character could appear.

Quantifier	Description
*	Zero or more matches.
+	One or more matches.
?	Zero or one matches.
{N}	N matches.
{N,}	N or more matches.
{N,M}	Between N and M matches.

The syntax of the control is as given:

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

CompareValidator

CompareValidator control is used to comapre two values. The value to compare can be either a value of another control or a constant specified. There are predefined data types that can be compared like string, integer etc.

Following are main properties of the validation control.

ControlToCompare	Gets or sets the ID of the control whose value will be compared with the currently entered value.
Operator	DataTypeCheck/Equal/GreaterThan/GreaterThanEqual/LessThan/LessThanEqual/Not Equal. Used to specify the comparison operation to peform. In case of DataTypeCheck, ControlToCompare properties are ingnored.
Display	Dynamic/Static. Used to indicate how the area of error message will be allocated. Dynamic: Error message area will only be allocated when error will be displayed. Static: Error messagea area will be allocated in either case.
Enabled	true/false. Gets or sets whether to enable the validation control or not.
ErrorMessage	Gets or sets the text of the error message that will be displayed when validation fails (This is displayed when ValidationSummary validatoin control is used.).
Text	Gets or sets the description of the error message text.
ValidationGroup	Gets or sets the validation group it belongs to. This is used to group a set of controls.
SetFocusOnError	true/false. Used to move focus on to the control that fails the validation.

DEMO : CompareValidator

Write into TextBox

```
   <asp:CompareValidator ID="CompareValidator1" runat="Server"
ControlToValidate="TextBox2" ControlToCompare="TextBox1" Operator="Equal"
Type="string" Text="Both textbox value should be same." ErrorMessage="Both textbox
values are not equal." Display="Dynamic"></asp:CompareValidator>
```

CustomValidator

CustomValidator control is used to validate an input control with user-defined function either from server side or client side. Generally, this control is used when you feel that no other validation controls fit in your requirement.

Following are main properties of the validation control.

ClientValidationFunction	Gets or sets the validation function that will be used in client side (JavaScript function).
OnServerValidate	Method that fires after post back.
ControlToCompare	Gets or sets the ID of the control whose value will be compared with the currently entered value.
Operator	DataTypeCheck/Equal/GreaterThan/GreaterThanEqual/LessThan/LessThanEqual/NotEqual. Used to specify the comparison operation to peform. In case of DataTypeCheck, ControlToCompare properties are ingnored.
Display	Dynamic/Static. Used to indicate how the area of error message will be allocated. Dynamic: Error message area will only be allocated when error will be displayed. Static: Error message area will be allocated in either case.
Enabled	true/false. Gets or sets whether to enable the validation control or not.
ErrorMessage	Gets or sets the text of the error message that will be displayed when validation fails (This is displayed when ValidationSummary validatoin control is used.).
Text	Gets or sets the description of the error message text.
ValidationGroup	Gets or sets the validation group it belongs to. This is used to group a set of controls.
SetFocusOnError	true/false. Used to move focus on the control that fails the validation.

DEMO : CustomValidator Show Source Code

Write into TextBox

```
    <asp:CustomValidator ID="CustomValidator1" runat="Server"
ControlToValidate="TextBox1" ClientValidationFunction="CheckForHardCodedValue"
ErrorMessage="Value doens't match." Text="TextBox value must be [GOLD]"
OnServerValidate="ValidateServerSide"></asp:CustomValidator>
  // JavaScript validation function ////////////////////////
function CheckForHardCodedValue(source, arguments)
   {
      var tID = '<%= TextBox1.ClientID %>';
      if (document.getElementById(tID).value == 'GOLD')
```

```
        arguments.IsValid = true;
      else
        arguments.IsValid = false;
   }
// Server side function to validate ////////////////////////
 protected void ValidateServerSide(object source, ServerValidateEventArgs args)
   {
      if (args.Value.Equals("GOLD"))
      {  args.IsValid = true;
        lblMessage.Text += "Page is valid.<br />"; }
      else
      {  args.IsValid = false;
        lblMessage.Text += "Page is NOT valid. <br />";      }    }
```

ValidationSummary

ValidationSummary control is used to summarize all validation errors on the page and display.

Following are main properties of the validation control.

ShowMessageBox	true/false. Popup alert box with all validation error, if true.
ShowSummary	true/false. Display summary of all errors on the page, if true.
DisplayMode	BulletList/List/SingleParagraph. Used to display all validation errors in specified format.
HeaderText	Used to write the header of the error summary.
ValidationGroup	Used to specify the group name of input controls for which summary will be displayed.

DEMO : ValidationSummary

Write into TextBox

```
// Form & Validation Control ////////////////////////
   <asp:ValidationSummary ID="ValidationSummary" runat="Server"
ShowMessageBox="true" ShowSummary="true" DisplayMode="List" HeaderText="Following
error occured." />
```

The web.config File

The web.config file uses a predefined XML format. The entire content of the file is nested in a root <configuration> element. This element contains a <system.web> element, which is used for ASP.NET settings. Inside the <system.web> element are separate elements for each aspect of configuration.

Here's the basic skeletal structure of the web.config file:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
<system.web>
<!-- Configuration sections go here. -->
</system.web>
</configuration>
```

This example adds a comment in the place where you'd normally find additional settings. XML comments are bracketed with the <!-- and --> character sequences, as shown here:

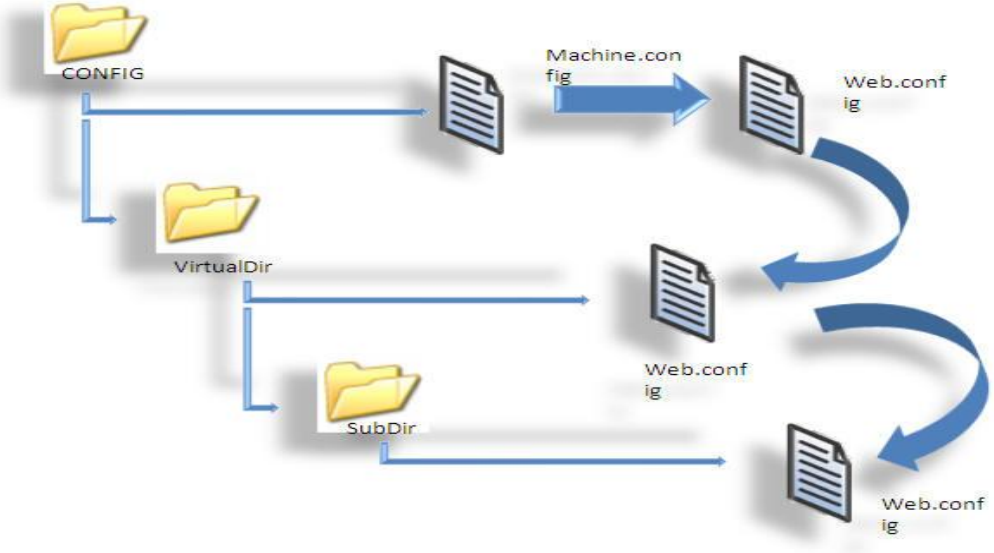
```
<!-- This is the format for an XML comment. -->
```

You can include as few or as many configuration sections as you want. For example, if you need to specify special error settings, you could add just the <customErrors> group.

Note that the web.config file is case-sensitive, like all XML documents, and starts every setting with a lowercase letter. This means you cannot write <CustomErrors> instead of <customErrors>.

If you want an at-a-glance look at all the available settings, head to *C:\WINDOWS\Microsoft.NET\Framework\[Version]\CONFIG* directory, and look at the web.config.comments file. This file consists of XML comments that show the available options for every possible setting.

The entire contents of a configuration file, whether it is *machine.config* or *web.config*,
is nested in a <configuration> element.



In the *web.config*, under the <configuration> element, there is another element <system.web>, which is used for ASP.NET settings and contains separate elements for each aspect of the configuration.

Important Configuration Tags

<authentication>

This element is used to verify the client's identity when the client requests a page from the server. This is set at the application level. We have four types of authentication modes: “None”, “Windows”, “Forms”, and “Passport”.

If we don't need any authentication, this is the setting we use:

<authentication mode="None"/>

<compilation>

In this section, we can configure the settings of the compiler. Here, we can have lots of attributes, but the most common ones are debug and defaultLanguage. Setting debug to true means we want the debugging information in the browser, but it has a performance tradeoff, so normally, it is set as false. And, defaultLanguage tells ASP.NET which language compiler to use: VB or C#.

<customErrors>

This tags includes the error settings for the application, and is used to give custom error pages (user-friendly error pages) to end users. In the case that an error occurs, the website is redirected to the default URL. For enabling and disabling custom errors, we need to specify the mode attribute.

```
<customErrors defaultRedirect="url" mode="Off">
  <error statusCode="403" redirect="/accesdenied.html" />
  <error statusCode="404" redirect="/pagenotfound.html" />
</customErrors>
```

- **"On"** means this settings is on, and if there is any error, the website is redirected to the default URL.
- **"Off"** means the custom errors are disabled.
- **"RemoteOnly"** shows that custom errors will be shown to remote clients only.

<trace>

As the name suggest, it is used for tracing the execution of an application. We have here two levels of tracing: page level and application level. Application level enables the trace log of the execution of every page available in the application. If pageOutput="true", trace information will be displayed at the bottom of each page. Else, we can view the trace log in the application root folder, under the name *trace.axd*.

```
<trace enabled="false" requestLimit="10" pageOutput="false"
      traceMode="SortByTime" localOnly="true" />
```

<appSettings>

This section is used to store custom application configuration like database connection strings, file paths etc. This also can be used for custom application-wide constants to store information over multiple pages. It is based on the requirements of the application.

```
<appSettings>
   <add key="Emailto" value="me@microsoft.com" />
   <add key="cssFile" value="CSS/text.css" />
</appSettings>
```

It can be accessed from code like:

```
ConfigurationSettings.AppSettings("Emailto")
```

Web Service

What is Web Service?

- Web Service is an application that is designed to interact directly with other applications over the internet. In simple sense, Web Services are means for interacting with objects over the Internet.
- Web Service is
    - Language Independent
    - Protocol Independent
    - Platform Independent
    - It assumes stateless service architecture.

Example of Web Service

- **Weather Reporting:** You can use Weather Reporting web service to display weather information in your personal website.
- **Stock Quote:** You can display latest update of Share market with Stock Quote on your web site.
- **News Headline:** You can display latest news update by using News Headline Web Service in your website.
- In summary you can any use any web service which is available to use. You can make your own web service and let others use it. Example you can make Free SMS Sending Service with footer with your companies advertisement, so whosoever use this service indirectly advertise your company... You can apply your ideas in N no. of ways to take advantage of it.

Web Service Communication
Web Services communicate by using standard web protocols and data formats, such as

- HTTP
- XML
- SOAP

Advantages of Web Service Communication
Web Service messages are formatted as XML, a standard way for communication between two incompatible system. And this message is sent via HTTP, so that they can reach to any machine on the internet without being blocked by firewall.

Terms which are frequently used with web services

- **What is SOAP?**
  - SOAP are remote function calls that invokes method and execute them on Remote machine and translate the object communication into XML format. In short, SOAP are way by which method calls are translate into XML format and sent via HTTP.

- **What is WSDL?**
  - WSDL stands for Web Service Description Language, a standard by which a web service can tell clients what messages it accepts and which results it will return.
  - WSDL contains every details regarding using web service
    - Method and Properties provided by web service
    - URLs from which those method can be accessed.
    - Data Types used.
    - Communication Protocol used.

- **What is UDDI?**
  - UDDI allows you to find web services by connecting to a directory.

- **What is Discovery or .Disco Files?**
  - .Disco File (static)
    - .Disco File contains
      - URL for the WSDL
      - URL for the documentation
      - URL to which SOAP messages should be sent.
    - A static discovery file is an XML document that contains links to other resources that describe web services.
  - .VsDisco File (dynamic)
    - A dynamic discovery files are dynamic discovery document that are automatically generated by VS.Net during the development phase of a web service.

- **What is difference between Disco and UDDI?**
  - Disco is Microsoft's Standard format for discovery documents which contains information about Web Services, while UDDI is a multi-vendor standard for discovery documents which contains information about Web Services.

Steps for Creation of webserive

- Craete a web site with suitable name
- Right click on solution explorer and selection add new item
- From pop window select web service and give suitable name and click on add button
- Web service created with default method called “hello world” you can also define your method here

Consumption or Usage:

- For adding web service in you application
- Rig click on solution explorer select add web reference
- Pop window open
- Give URL of your web service location and click on go
- The web service will available with available method and then click on add reference button web service will create

For using in your web page

- Imports webserive.
- Create a object of web serive where you want to use.
- By using object pass parameter in web service methods

4.1 Principles of Mathematics by Aryabhata

Introduction: Aryabhata (476 CE) was one of ancient India's greatest mathematicians and astronomers. His seminal work, *Aryabhatiya*, contains sutras (verses) that describe foundational concepts in mathematics, including arithmetic, geometry, and trigonometry.

4.1.1 Principles of Mathematics: Sutra (Verse 1.1)

Sanskrit Verse (Ganitapāda 1.1)

"Caturadhikaṃ śatamaṣṭaguṇaṃ dvāṣaṣṭistathā sahasrāṇām"

Translation & Interpretation:
This verse introduces a numerical system based on positional decimal values. Aryabhata used Sanskrit syllables to denote numbers—a cryptic yet efficient system that encoded values and operations in verse form.

Mathematical Principle:
Introduction to the place value system and the decimal system (based on powers of 10), which predated similar Western systems by centuries.

4.1.2 Value of Pi: Sutra (Verse 3.1)

Sanskrit Verse (Āryabhatiya 2.10)

"Add four to 100, multiply by 8, and then add 62,000. This is the approximate circumference of a circle with diameter 20,000."

Calculation:

$$\frac{(4+100)\times 8+62000}{20000}=\frac{62832}{20000}=3.1416$$

Mathematical Principle:
This gives an approximation of π accurate to four decimal places — remarkably close to modern values.

4.1.3 Sine Function: Sutra (Verse 3.2)

Concept:
Aryabhata introduced the concept of *ardha-jya* (half-chord), which corresponds to the modern sine function.

Explanation:
He created a table of sines for every 3.75° increment using geometry and interpolation, essential for astronomical calculations.

Significance:
It marks the first recorded use of the sine function, centuries before its formal appearance in Arabic and European mathematics.

4.1.4 Trigonometric Functions: Sutra (Verse 3.11)

Concept:
Aryabhata's work includes the use of *sine (jya)* and *cosine (kojya)* to solve problems involving spherical astronomy.

Mathematical Principle:
Understanding angular measurements, trigonometric ratios, and relationships among sides and angles of triangles — foundational for astronomy and navigation.

4.2 Ancient Knowledge from the Shulba Sutras (Vedic Texts)

Overview:
The *Shulba Sutras* (c. 800 BCE) are part of the Vedas, focusing on geometry for altar construction. "Shulba" means rope, indicating geometry done via rope-measurement.

4.2.1 Construction of a Square

Method:
Using a rope and pegs, Vedic scholars constructed a square with right angles using basic geometric techniques, including diagonals and perpendicular bisectors.

Mathematical Principle:
The square's symmetry and use of right angles are essential for altar accuracy and were among the earliest uses of geometric constructions.

4.2.2 Pythagorean Theorem (Sulbha Sutra 1.2)

Sanskrit Verse:

"The diagonal of a rectangle produces both areas which its length and breadth produce separately."

Modern Interpretation:

$$a^2 + b^2 = c^2$$

Significance:
This is one of the earliest recorded instances of the Pythagorean theorem, predating Pythagoras by several centuries.

4.2.3 Area of a Circle

Method (Sulba Sutras):

They used the formula:

$$Area \approx \left(\frac{13}{15}\right)^2 \times d^2 \quad (\text{where } d \text{ is the diameter})$$

Approximation of π:

$$\pi \approx \left(\frac{13}{15}\right)^2 \times 4 \approx 3.04$$

Though less accurate than Aryabhata's, it was revolutionary for its time.

4.2.4 Area of a Triangle

Method:

$$Area = \frac{1}{2} \times base \times height$$

This was understood through constructions and rope measurements to ensure perpendicularity.

Cultural Relevance:
Used in altar construction for Vedic rituals with precise measurements and symbolic shapes.

4.3 Ancient Knowledge by Brahmagupta

Overview:
Brahmagupta (598–668 CE), a brilliant mathematician, wrote *Brahmasphutasiddhanta*, which includes early algebra, number theory, and geometry.

4.3.1 Area of a Cyclic Quadrilateral (Verse 10)

Sanskrit Verse:

"The square root of the product of the semi-perimeter minus each side, multiplied together, gives the area."

Formula (Brahmagupta's Formula):
For a cyclic quadrilateral with sides a,b,c,da, b, c, da,b,c,d,

$$s = \frac{a+b+c+d}{2}$$

$$Area = \sqrt{(s-a)(s-b)(s-c)(s-d)}$$

Significance:
This generalized Heron's formula for all quadrilaterals inscribed in a circle.