Web Site development using ASP.NET AND C#

UNIT – I

Overview of ASP.NET framework, Understanding ASP.NET Controls, Applications, Web servers, installation of IIS. Web forms, web form controls -server controls, client controls, web forms & HTML, Adding controls to a web form ,Buttons, Text Box , Labels, Checkbox, Radio Buttons, List Box, etc.

Running a web Application, creating a multiform web project.

UNIT-II

Form Validation: Client side validation, server Side validation, Validation Controls : Required Field Comparison Range. Calendar control, Ad rotator Control, Internet Explorer Control.

State management- View state, Session state, Application state,

UNIT-III

Architecture of ADO.NET, Connected and Disconnected Database, Create Connection using ADO.NET Object Model, Connection Class, Command Class, DataAdapter Class, Dataset Class. Display data on data bound Controls and Data Grid.

Database Accessing on web applications: Data Binding concept with web, creating data grid, Binding standard web server controls. Display data on web form using Data bound controls.

UNIT-IV

Writing datasets to XML, Reading datasets with XML. Web services: Introduction, Remote method call using XML, SOAP, web service description language, building & consuming a web service, Web Application deployment.

UNIT-V

Overview of C#, C# and .NET, similarities & differences from JAVA, Structure of C# program. Language features: Type system, boxing and unboxing, flow controls, classes, interfaces, Serialization, Delegates, Reflection.

UNIT – I

Overview of ASP.NET framework
Understanding ASP.NET Controls
Applications
Web servers
Installation of IIS
Web forms
Web form controls-
- server controls
- client controls

Web forms & HTML
Adding controls to a web form:
- Buttons
- Text Box
- Labels
- Checkbox
- Radio Buttons
- List Box etc.

Running a web Application
Creating a multiform web project.

Overview of ASP.NET framework:

.NET is short name of Microsoft Visual Studio 2003, 2005, 2008, 2010, 2013, 2015. It is not a language but service provider for software development. Web developer uses ASP.NET for designing web sites, C#.NET for programming and ADO.NET for database connectivity. All these services are supported by .NET Framework. Without .NET Framework, application cannot be designed, coded and connected to data base. .NET Framework is consisting using following three parts.

Interface for ASP.NET Web Application & C# code
Class library for ADO.NET & Others
CLR (Common Language Runtime) to execute .NET application

Thus it is clear that, for ASP.NET application, system must have .NET Framework because it provides an interface for ASP.NET application, available base class library for ADO.NET and finally execute using CLR on any machine architecture but run on Microsoft operating system.

Understanding ASP.NET Controls:

ASP.NET is developed by Microsoft for web application. It provides a number of controls in following formats.

```
<asp:ControlName Id="ControlName1" runat="server"/>
```

Example:

```
<asp:TextBox Id="TextBox1" runat="server"/>
<asp:Label Id="TextBox1" runat="server"/>
<asp:Image Id="TextBox1" runat="server"/>
<asp:CheckBox Id="CheckBox1" runat="server"/>
<asp:Button Id="Button1" runat="server"/>
<asp: Id="Button1" runat="server"/>
```

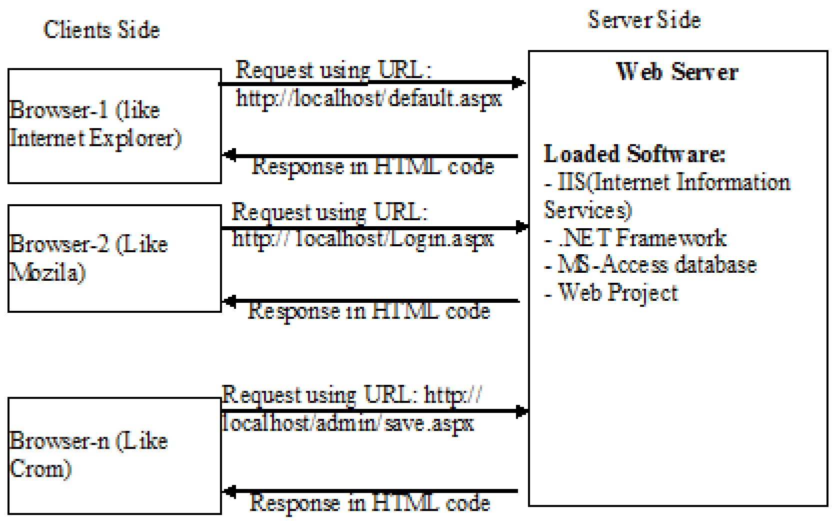
Such coding format called ASP.NET control. They are processed by .NET Framework at server side and convert into HTML / JScript code for client side. At server side, **`asp:ControlName`** is consider as class and **`Id="ControlName1"`** as object.

Applications:

We can use ASP.NET to design websites that includes web pages and web services. It is capable to connect database and access data on ASP.NET controls. We can set security options on restricted pages.

Web Servers and Web Browser:

This client-server architecture to access web items from one place to many places can be shown as.


Clients Side
Server Side
Browser-1 (like Internet Explorer)
Request using URL: http://localhost/default.aspx
Response in HTML code
Browser-2 (Like Mozila)
Request using URL: http:// localhost/Login.aspx
Response in HTML code
Browser-n (Like Crom)
Request using URL: http:// localhost/admin/save.aspx
Response in HTML code
Web Server
Loaded Software:
- IIS(Internet Information Services)
- .NET Framework
- MS-Access database
- Web Project

Web servers:

A computer on which web pages and web services are stored called web server. In Microsoft Windows system, IIS (Internet Information Services) software is used to create web server to host website into "Initpub" folder. Web server accept an URL request for from client side then search into Initpub folder, process it using .NET Framework and finally response to client in HTML+CSS+JScript code.

Web Browser:

At client side, user made a request in URL form then accept its response in HTML+CSS+JScript form with the help of an application software called web browser. for example Internet Explorer, Crome, Mozila etc.

Installation of IIS:

A computer called web server on which IIS software is installed. IIS is supported on Microsoft operating system like WindowsXP, Windows3, Windows7 etc.
To install IIS we perform following action.

1. Open control panel.
2. click on Programs.
3. select "Turn windows features on or off"
4. To turn on Internet Information Services, select checkbox.
5. Press Ok button.

After some time installation process of IIS will be completed. We get a folder **Inetpub** in C drive. When we type **localhost** on address bar of browser and press enter key then we get a default web page. It means IIS have installed successfully.

When system is has Microsoft Visual Studio2005 or higher version then we do not need to installed IIS software because at the time of running ASP.NET application, .NET Framework automatically creates virtual IIS.

Web forms:

A web page that contains <form> tag called web forms.

For example:

```
<html>
<head>
     <title>Home page</title>
</head>
<body>
     <form  method="GET/POST"
     action="http://lrsir.net/scriptpage.aspx.cs">
               Place all web control here
     </form>
</body>
</html>
```

Above html code is processed at client side. When user click on button control then form data are send to scriptpage.aspx.cs file at server side using GET or POST method. GET method send form data through querystring (`http://lrsir.net/scriptpage.aspx.cs?uid=lrsir&pwd=12345`) and POST method send form data in the http header in hidden form. Every web page is defined at least in above format.

Web form controls:

A user interface element through which we submit data from client side to server side or shows server side data to client sides called web form control.

Server controls: Controls which are executed at server side called server controls. For example, all ASP.NET controls are server controls.

```
<asp:TextBox Id="TextBox1" runat="server"></Textbox>
```

This control convert into html text control when web page requested and receive data submitted to textbox at client side.

Client controls: Controls which are executed at client side by browser software called client controls. User submits data using these controls. For example, all html controls.

<type="text" name="uid" id="text1"></text>

It shows textbox on webpage to input data at client side by browser.

Web forms & HTML:

A page is called web page that contain html tag, html control, server controls, css, Jscript and C# / VB code.

HTML (Hyper Text Markup Language) is a formatting language. HTML codes are written in tag format < > and executed by browser at client side. HTML provides a number of tags to show user interface controls, table, background color etc.

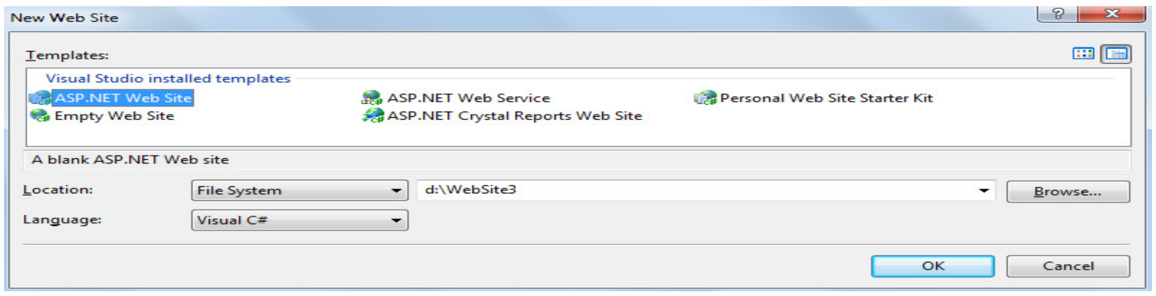
Web form is a web page in which all server side / client side controls are written inside <form></form> tag called web form.

Structure of web form is following:

```
<html>
<head>
     <title>Write here Title of page</title>
     <style type="text/css">
          Write css code here
     </style>
     <script type="text/jscript">
          Write client side code
     </script>
</head>
<body>
     <form attributes>
          Write here client / server side control
     </form>
</body>
</html>
```

Adding controls to a web form:

To add controls on web form, first we have to open ASP.NET from start→All Programs→Microsoft Visual Studio 2005(or next version)→File→New→Website. We get following dialog box.



New Web Site

Templates:

Visual Studio installed templates

ASP.NET Web Site | ASP.NET Web Service | Personal Web Site Starter Kit

Empty Web Site | ASP.NET Crystal Reports Web Site

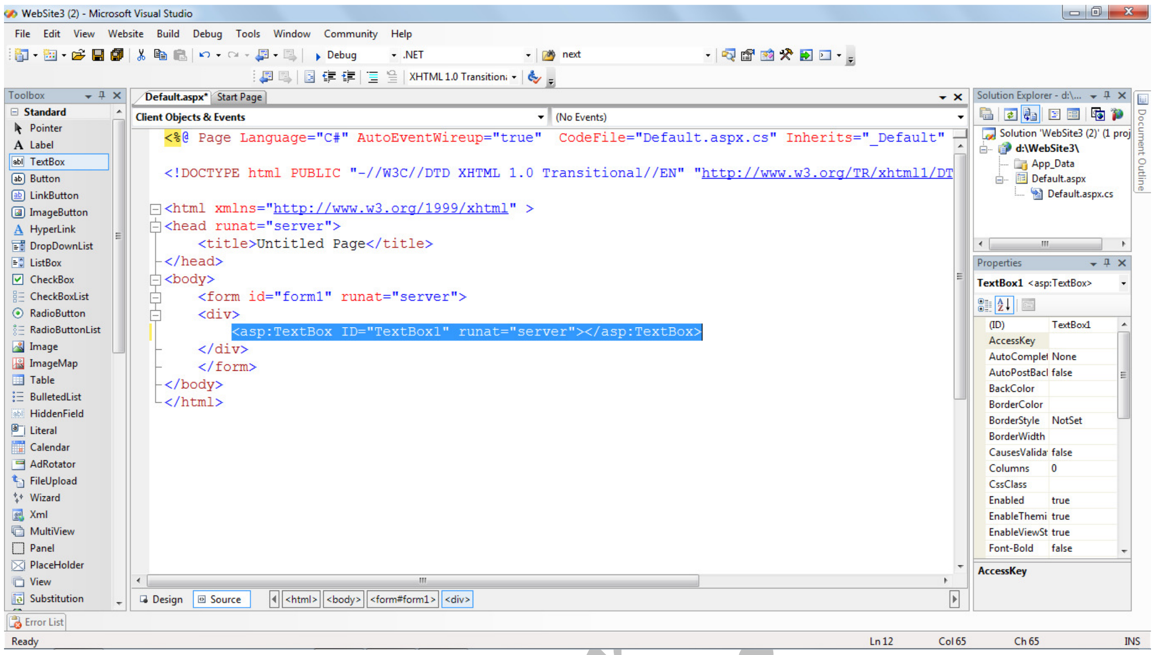
A blank ASP.NET Web site

Location: File System | d:\WebSite3 | Browse...

Language: Visual C#

OK | Cancel

We choose ASP.NET Web Site in template, File System in Location, Website path using browser and C# language then press Ok button. We get following IDE (Integrated Development Environment) with default.aspx web page.



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

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DT

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    </div>
    </form>
</body>
</html>
```

(ID)	TextBox1
AccessKey	
AutoComplet	None
AutoPostBacl	false
BackColor	
BorderColor	
BorderStyle	NotSet
BorderWidth	
CausesValida	false
Columns	0
CssClass	
Enabled	true
EnableThemi	true
EnableViewSt	true
Font-Bold	false

ASP.NET IDE is divided into following sections.

1. **Menubar:** This top most section contains all the options that are applicable for development of ASP.NET website. For Example- File, Edit, View, Website, Built, Debug etc.
2. **Toolbar:** This second top section contains Icons of most applicable options that can apply quickly that is on single click. For Example, Icons for New Project, Add New Item, Open File, Save Selected Items, Save All, Cut, Copy, Paste, Undo, Redo, Start Debugging etc.
3. **Toolbox:** This left section contains all the ASP.NET controls, HTML controls, Ajax Controls. They are used to design web pages.
4. **Solution Explorer:** This right section shows all files of websites. Using this section we can add new item (web form .aspx, stylesheet .css, HTML page .htm, SQL database .dba, class .cs, web.config, Global.asax etc.), Add ASP.NET Folders (App_Code, App_Data, Theme etc.), set as start page, open file code in middle section, Build website etc.
5. **Property Window:** This right section shows all the properties of selected item on website.
6. **Source Section:** This middle section shows source code of selected file.

7. **Design Section:** This middle section shows the layout of controls on web form before running website.

Using toolbox we can add required ASP.NET and HTML controls on web page. Controls can be added using following steps.

1. Open web page from solution explore.
2. Choose source / design tab from bottom of web page.
3. Set position of control at desired place.
4. Select control from toolbox.
5. Hold, Drag and Drop at desired position on web page.
   We can also type any control's code in source section of web page.

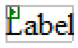
We can assign values to the properties of control when we declare control on the web page or within the code of page in C# language.
There are following common properties for all controls.

1. **ToolTip:** When mouse pointer move over the control then to show message.
2. **TabIndex:** When we press tab key then to set focus order of controls. 0 is starting number.
3. **Enabled:** True means active and false means inactivate and show in half transparent.
4. **Visible**: True means control will visible and false means not visible on browser.
5. **Width**: integer value in pixels to set width of control.
6. **Height**: integer value in pixels to set height of control.
7. **ForeColor**: To set font color like Red, Blue.
8. **Font-Names**: To set view of font like TimesNewRoman etc.
9. **Font-size**: Integer value to set large or small size of font.
10. **Font-Bold**: To set thicker font (true/false). Ex: **www.LRsir.net**
11. **Font-Italic**: To set italic font (true/false). Ex: *www.LRsir.net*
12. **Font-Overline** (true/false): To set line top of font.
13. **Font-Strikeout** (true/false): To set line middle of font. Ex: ~~www.LRsir.net~~
14. **Font-Underline** (true/false): To set line at the bottom of font. Ex: www.LRsir.net
15. **BackColor**: To set background color of the control. Like red.
16. **BorderStyle**: To set border around the control like single line, double line, dotted etc.
17. **BorderWidth**: Integer value to set width of border in pixel.

18. **CssClass**: To apply css class on control.
19. **SkinId**: To apply preset formatting.
20. **EnableTheming**: To apply format of them on control or not. (True/False)
21. **EnableViewState**: If it is true then data on control persist after redirecting to same page on browser.
22. **AutoPostBack**: By default it is false. When it is true then automatically posts the form whenever a change is made to the content of control. Most applicable for DropDownList when we select any item.

Basic ASP.NET controls:

Label control



It is used to show non editable message on web page and it can be modify using code.

Syntax:

```
<asp:Label ID="Label1" runat="server"/>
```

Important Properties:

- Text: Gets or sets the text displayed by the Label control.

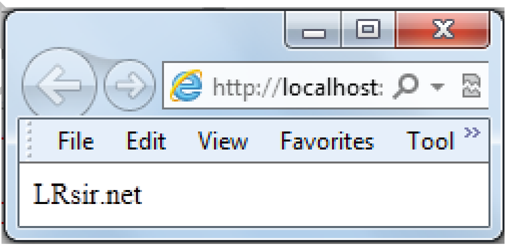
Methods:

- None

Events:

- None

Example: Display LRsir.net on Label control



LRsir.net

1) using ASP.NET code

```
<asp:Label ID="Label1" runat="server"
     Text="LRsir.net" />
```

2) Using C# code

Webpage: *Showtime.aspx*

```
<%@ Page Language="C#" %>
<script runat="server">
     void Page_Load()
     {
       Label1.Text = "LRsir.net";
     }
</script>
<html >
<head><title>Show Label</title></head>
<body>
<form id="form1" runat="server">
<div>
     <asp:Label ID="Label1" runat="server" />
</div>
</form>
</body>
</html>
```

Remark: By default, a Label control renders its contents in an HTML <span> tag.

TextBox control

In this control, user can type data to make input on web form.

Syntax:

```
     <asp:TextBox ID="TextBox1" runat="server"/>
```

Important Properties:

- Text: Gets or sets the text displayed by the TextBox control.
- TextMode: It can accepts following three values:
    - i. SingleLine—Displays a single-line input field.
    - ii. MultiLine—Displays a multi-line input field.
    - iii. Password—Displays a single-line input field in which the text is hidden.
- MaxLength: The maximum number of characters that can be entered.
- ReadOnly: True mean text cannot be changed in textbox.
- Rows: If TextMode property is MultiLine then it sets the number of lines.
- Columns: It sets number of columns to display.
- Wrap: True / False for wrapping text or not when TextMode is MultiLine.

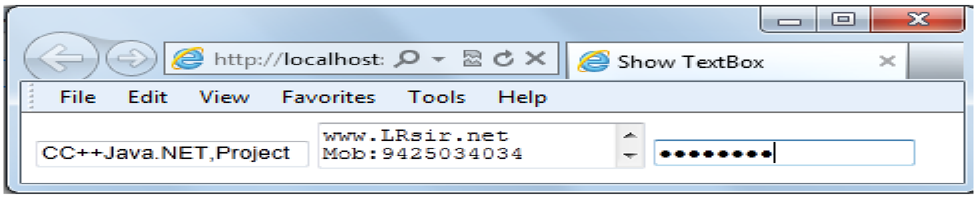
Methods:

- Focus: To set focus on that control.

Events:

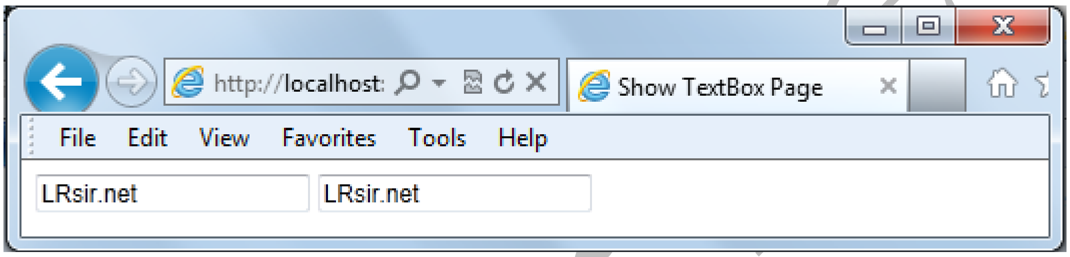
- TextChanged: Raised on server when user made changes in textbox.(AutoPostBack property must be true for this event)

Example1: Display TextBox for single line, multiline and password using ASP.NET code of TextBox.



```
<asp:TextBox ID="TextBox1" runat="server"/>
<asp:TextBox ID="TextBox2" runat="server"
     TextMode="MultiLine"/>
<asp:TextBox ID="TextBox3" runat="server"
     TextMode="Password"/>
```

Example: Copy text of one textbox into another using TextChanged event.



```
<%@ Page Language="C#" %>
<script runat="server">
    void TextBox1_TextChanged(object sender, EventArgs e)
    {
        TextBox2.Text = TextBox1.Text;
    }
</script>
<html>
<head> <title>Show TextBox Page</title></head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:TextBox ID="TextBox1" runat="server"
         AutoPostBack="true"
         OnTextChanged="TextBox1_TextChanged"/>
        <asp:TextBox ID="TextBox2" runat="server"/>
     </div>
    </form>
</body>
</html>
```

Remark: By default, a TextBox control renders its contents in an HTML <Input Type="Text"> tag.

CheckBox control

Using this control, we display message along with check box. One or more CheckBoxs can be selected.

Syntax:

```
        <asp:CheckBox ID="CheckBox1" runat="server"/>
```

Important Properties:

- Text: Gets or sets the text displayed by the CheckBox control.
- Checked: True/False. True for select and False for unselect.

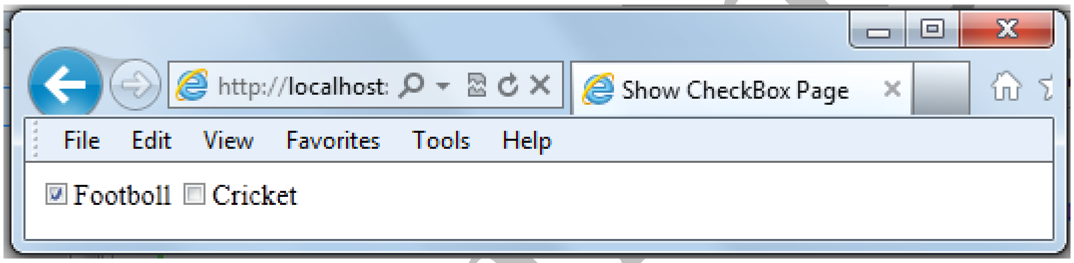
Methods:

- Focus: To set focus on that control.

Events:

- CheckedChanged: Raised on server when user made checked or unchecked in check box. (AutoPostBack property must be true for this event)

Example1: Display two CheckBoxs for Hobbies using ASP.NET code.

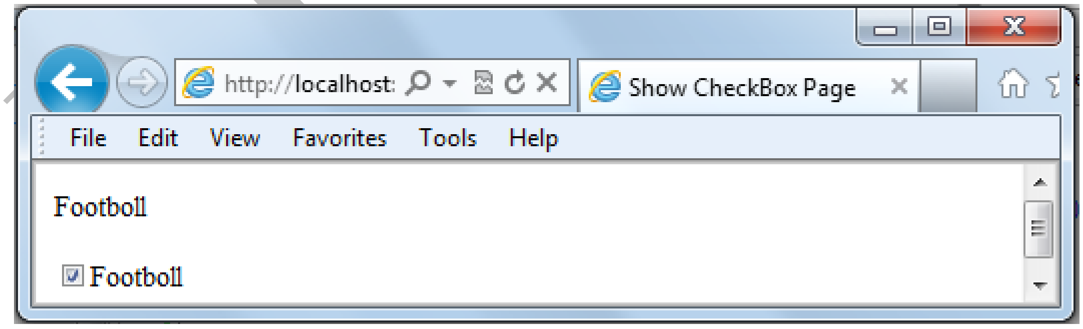


☑ Footboll ☐ Cricket

```
<asp:CheckBox ID="CheckBox1" runat="server" Text="Footboll"
Checked="True" />
```

```
<asp:CheckBox  ID="CheckBox2"  runat="server"  Text="Cricket"
/>
```

Example: Display text of checkbox when checked using CheckedChanged event.



Footboll

☑ Footboll

```
<%@ Page Language="C#" %>
<script runat="server">
  void CheckBox1_CheckedChanged(object sender, EventArgs e)
  {
     if(CheckBox1.Checked==true)
     {
          Response.Write(CheckBox1.Text);
     }
  }
</script>
<html>
<head> <title>Show CheckBox Page</title></head>
<body>
    <form id="form1" runat="server">
    <div>
     <asp:CheckBox ID="CheckBox1" runat="server"
          Text="Footboll" AutoPostBack="true"
          OnCheckedChanged="CheckBox1_CheckedChanged"/>
     </div>
    </form>
</body>
</html>
```

Remark: By default, a CheckBox control renders its contents in an HTML <Input Type="checkbox"> tag.

RadioButton control

This control always use in group. Only one radio button can be checked in one group of radio buttons.

Syntax:

```
     <asp:RadioButton ID=" RadioButton1" runat="server"/>
```

Important Properties:

- Text: Gets or sets the text displayed by the RadioButton control.
- Checked: True/False. True for select and False for unselect.
- GroupName: Group that any radio button belongs to.

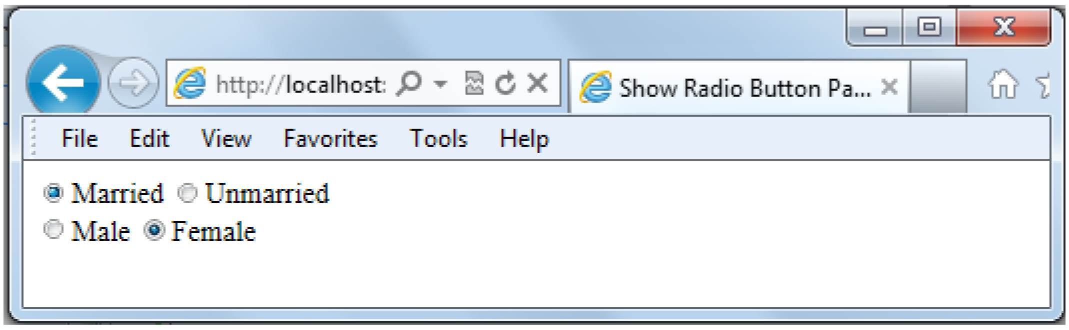
Methods:

- Focus: To set focus on that control.

Events:

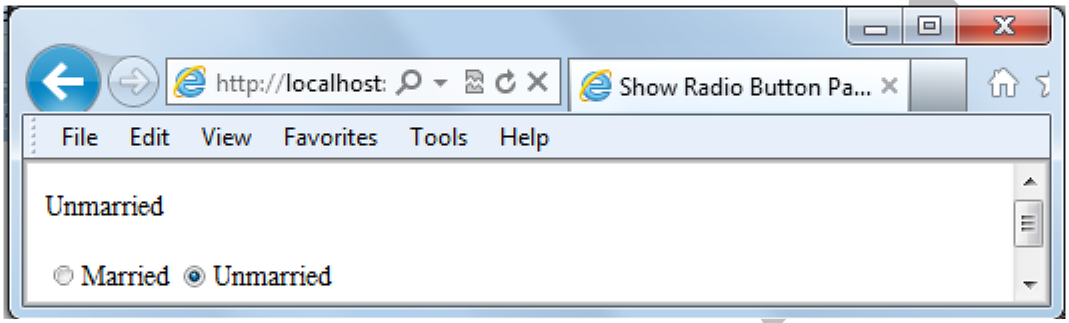
- CheckedChanged: Raised on server when user made checked or unchecked in RadioButton. (AutoPostBack property must be true for this event)

Example1: Display two groups of RadioButtons using ASP.NET code.



```
<asp:RadioButton ID="RadioButton1" runat="server"
GroupName="status" Text="Married" />
<asp:RadioButton ID="RadioButton2" runat="server"
GroupName="status" Text="Unmarried" />
<br />
<asp:RadioButton ID="RadioButton3" runat="server"
GroupName="sex" Text="Male" />
<asp:RadioButton ID="RadioButton4" runat="server"
GroupName="sex" Text="Female" />
```

Example: Display text of checked radioButtons using CheckedChanged event.



Unmarried

○ Married ● Unmarried

```
<%@ Page Language="C#" %>
<script runat="server">
void RadioButton1_CheckedChanged(object sender,EventArgs e)
    {
        if (RadioButton1.Checked == true)
        {
            Response.Write(RadioButton1.Text);
        }
    }
void RadioButton2_CheckedChanged(object sender,EventArgs e)
    {
        if (RadioButton2.Checked == true)
        {
            Response.Write(RadioButton2.Text);
        }
    }
</script>
<html>
<head> <title>Show RadioButton Page</title></head>
<body>
   <form id="form1" runat="server">
   <div>
     <asp:RadioButton ID="RadioButton1" runat="server"
     GroupName="status" Text="Married" AutoPostBack="True"
     OnCheckedChanged="RadioButton1_CheckedChanged" />
     <asp:RadioButton ID="RadioButton2" runat="server"
     GroupName="status" Text="Unmarried"
     AutoPostBack="True"
     OnCheckedChanged="RadioButton2_CheckedChanged" />
    </div>
    </form>
</body>
</html>
```

Remark: By default, a RadioButton control renders its contents in an HTML <Input Type="radio"> tag.

HyperLink control

HyperLink

This control creates a link to a web page to navigate another web page. This control does not submit form to a server.

Syntax:

```
    <asp:HyperLink ID=" HyperLink1" runat="server"/>
```

Important Properties:

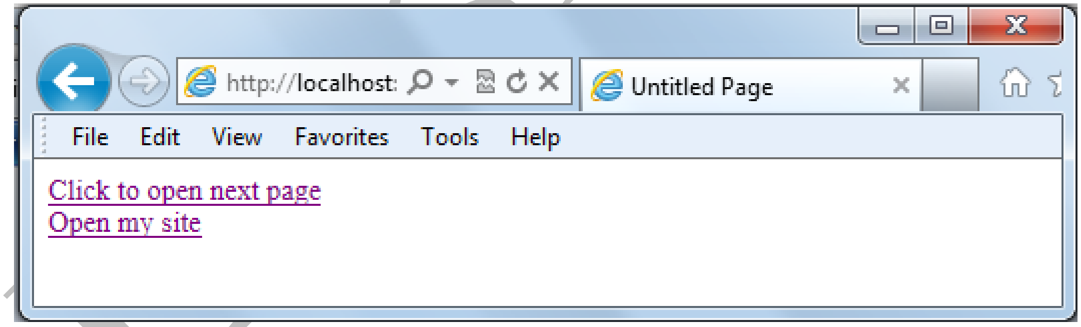
- Text: Gets or sets the text displayed by control.
- NavigateUrl: To specify URL.
- ImageUrl: To specify Image for link.
- Target: The target frame for NavigetUrl.(_blank, _parent)

Methods:

- none

Events:

- **none**



Click to open next page
Open my site

Example1: Navigate to another page of web site using ASP.NET code.

```
<asp:HyperLink ID="HyperLink1" runat="server"
NavigateUrl="webpage1.aspx">Click to open next page
</asp:HyperLink>
```

Example2: Navigate to another website from current web page.

```
<asp:HyperLink ID="HyperLink1" runat="server"
NavigateUrl="http://www.LRsir.net">open my
site</asp:HyperLink>
```

Remark: By default, a RadioButton control renders its contents in an HTML <Input Type="radio"> tag.

Image control

This control is used to display image.This control does not submit form to a server.

Syntax:

```
<asp:Image ID="Image1" runat="server"/>
```

Important Properties:

- AlternateText: Display text when image is unavailable.
- ImageUrl: To specify Image URL that shown.
- ImageAlign: To specify alignment of Image.

Methods:

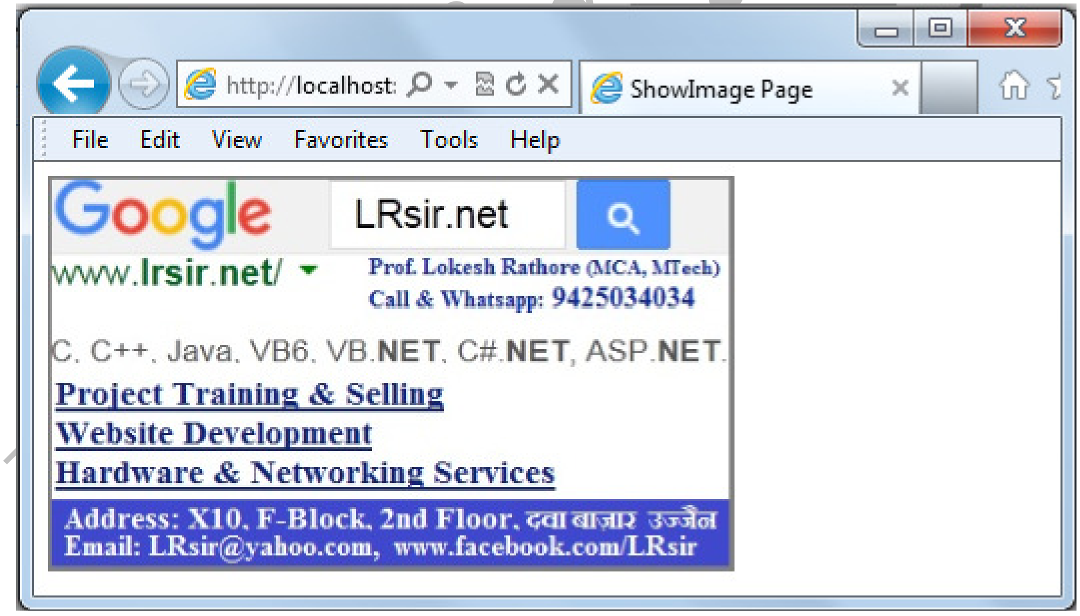
- none

Events:

- **none**

Example1: Show Image to a web page using ASP.NET code.

```
<asp:Image ID="Image1" runat="server"
AlternateText="LRsir.net" ImageUrl="~/visiting2.jpg" />
```


http://localhost:
ShowImage Page
File Edit View Favorites Tools Help
Google
LRsir.net
www.lrsir.net/
Prof. Lokesh Rathore (MCA, MTech)
Call & Whatsapp: 9425034034
C, C++, Java, VB6, VB.NET, C#.NET, ASP.NET
Project Training & Selling
Website Development
Hardware & Networking Services
Address: X10, F-Block, 2nd Floor, दवा बाजार उज्जैन
Email: LRsir@yahoo.com, www.facebook.com/LRsir

Remark: By default, a Image control renders its contents in an HTML <img> tag.

Submitting Form Data

The ASP.NET Framework includes three controls to submit a form to the server: the Button, LinkButton, and ImageButton controls. These controls have the same function, but each control has a distinct appearance.

Simple Button control

When we click Button Control then control is push down and web form is submitted to the server so that we can process C# code.

Syntax:

```
<asp:Button ID="Button1" runat="server"/>
```

Important Properties:

- Text: The text to be shown on button.
- CommandArgument: Argument that passed to Command event of button.
- CommandName: Specify a command name that is passed to command event.
- PostBackUrl: Url for after click button.
- UseSubmitBehavior: (true/false) Indicate whether the button render as a submit button.
- OnClientClick: The client side script that is executed on client-side OnClick.

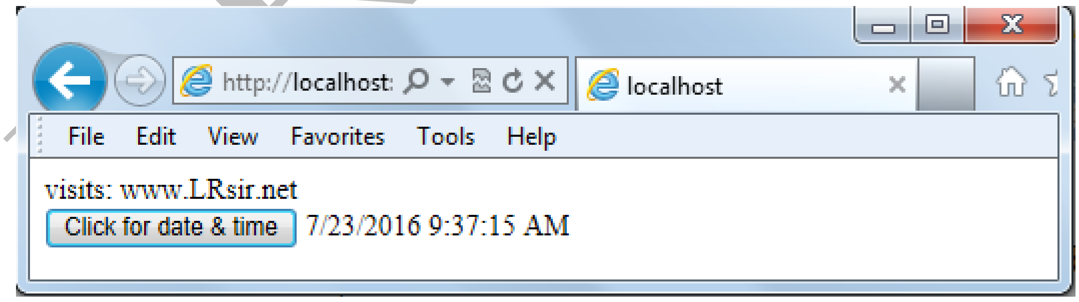
Methods:

- Focus: To set focus on that control.

Events:

- Click: Raised when button control is clicked.
- Command: Raised when button control is clicked. The command name and command argument passed to this event.

Example: Display current time to a Label control when button control is clicked, also pass argument to command name.



visits: www.LRsir.net

7/23/2016 9:37:15 AM

```
<%@ Page Language="C#" %>
<script runat="server">
void Button1_Click(object sender, EventArgs e)
{
   Label1.Text = DateTime.Now.ToString();
}
void Button1_Command(object sender, CommandEventArgs e)
{
    if (e.CommandName == "website")
```

```
       {
        Response.Write("visits: " + e.CommandArgument);
       }
 }
</script>
<html>
<head>
    <title>Show Button control Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Button ID="Button1" runat="server"
          Text="Click for date & time"
          CommandArgument="www.LRsir.net"
          CommandName="website"
          OnClick="Button1_Click"
          OnCommand="Button1_Command"/>
        <asp:Label ID="Label1" runat="server"/>
    </div>
    </form>
</body>
</html>
```

Remark: By default, a Button control renders its contents in an HTML <Input Type="submit"> tag.

LinkButton control

LinkButton

Working of LinkButton Control is similar to push Button. It look like HyperLink control. When we click then web form is submitted to the server to process C# code.

Syntax:

```
     <asp:LinkButton ID="LinkButton1" runat="server"/>
```

Important Properties:

- Text: The text to be shown on button.
- CommandArgument: Argument that passed to Command event of button.
- CommandName: Specify a command name that is passed to command event.
- PostBackUrl: Url for after click button.
- OnClientClick: The client side script that is executed on client-side OnClick.

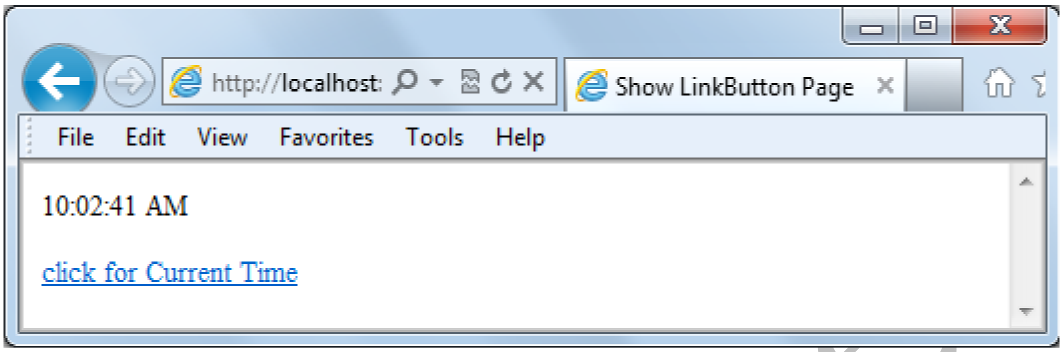
Methods:

- Focus: To set focus on that control.

Events:

- Click: Raised when button control is clicked.
- Command: Raised when button control is clicked. The command name and command argument passed to this event.

Example: Display current time when LinkButton control is clicked.



10:02:41 AM

click for Current Time

```
<%@ Page Language="C#" %>
<script runat="server">
  void LinkButton1_Click(object sender, EventArgs e)
    {
        Response.Write(DateTime.Now.ToString("T"));
    }
</script>
<html>
<head>
    <title>Show LinkButton Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:LinkButton ID="LinkButton1" runat="server"
          Text="click for Current time"
          OnClick="LinkButton1_Click"/>
    </div>
    </form>
</body>
</html>
```

Remark: By default, a Button control renders its contents in an HTML <a> tag.

ImageButton control

Working of ImageButton Control is similar to push Button. It looks like Image control. When we click then web form is submitted to the server to process C# code.

Syntax:

```
     <asp:ImageButton ID="ImageButton1" runat="server"/>
```

Important Properties:

- AlternateText: The text displayed when image cannot be shown.
- ImageUrl: The URL of image to be shown.
- ImageAlign: for the alignment of image.
- CommandArgument: Argument that passed to Command event of button.
- CommandName: Specify a command name that is passed to command event.
- PostBackUrl: URL for after click button.
- OnClientClick: The client side script that is executed on client-side OnClick.

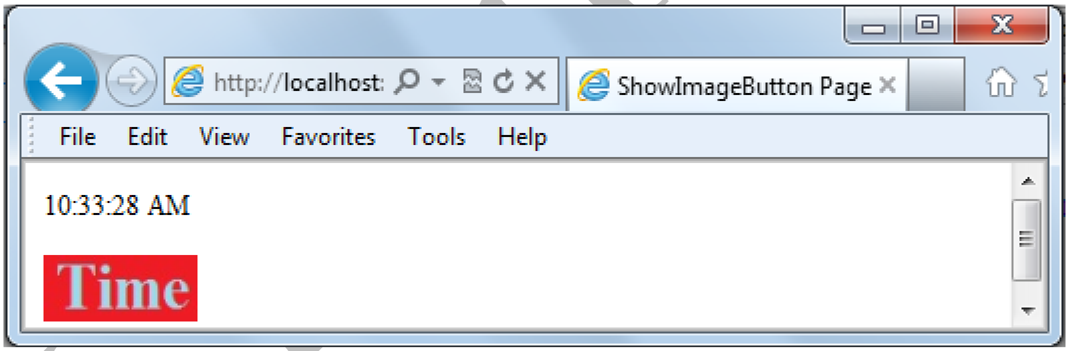
Methods:

- Focus: To set focus on that control.

Events:

- Click: Raised when button control is clicked.
- Command: Raised when button control is clicked. The command name and command argument passed to this event.

Example: Display current time when ImageButton control is clicked.



10:33:28 AM

```
<%@ Page Language="C#" %>
<script runat="server">
void ImageButton1_Click(object sender,ImageClickEventArgs e)
    {
        Response.Write(DateTime.Now.ToString("T"));
    }
</script>
<html>
<head>
    <title>ShowImageButton Page</title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
     <asp:ImageButton ID="ImageButton1" runat="server"
```

```
      ImageUrl="ImgBtn.jpg"
      OnClick="ImageButton1_Click" /></div>
    </form>
</body>
</html>
```

Remark: By default, a ImageButton control renders its contents in an HTML <Img> tag.

List Controls:

(DropDownList, ListBox, RadioButtonList, CheckBoxList, BulletedList)

These controls are used to display a list of items in different views. All these controls share a common set of properties and methods.

Display a list of items:

A list of items in such controls can be display using **ListItem** class in following manners.

```
<asp:ListControlName ID="ListControlName1 runat="server">
      <asp:ListItem Text="Item1" Value="value1" Selected="False"/>
      <asp:ListItem Text="Item2" Value="value2" Selected="True"/>
</asp: ListControlName>
```

- **Text:** To display item on list control.
- **Value:** To specify hidden value associated with list item.
- **Selected:** True for select and False for unselect.
- **Enabled:** True for enable and false for disable.

ListControlName: DropDownList, ListBox, RadioButtonList, CheckBoxList and BulletedList.

Binding to a Data source:(using C#)

We can show a list of items from database programmatically using following two propeties of List controls.

- **DataTextField:** To set Text property of each item.
- **DataValueField:** To set Value property of each item.

Determining the Selected List Item: (using C#)

All List controls support three properties that can use to determine the selected list item:

- **SelectedIndex:** Gets or sets the index of the selected list item.
- **SelectedItem:** Gets the first selected list item and set to ListItem.
- **SelectedValue:** Gets or sets the value of the first selected list item.
- **SelectedItem.Text:** Gets the first selected list item's text item.

```
ListItem item=ListControlName1.SelectedItem;
```

Adding item into ListItem of List Control:(C# code)

```
      ListItem item=new ListItem("NewText","NewValue");
      ListControlName1.Items.Add(item);
```

Removing item from ListItem of List Control:(C# code)

```
    ListControlName1.Items.Remove(ListItem item);
```

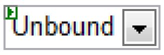
To clear selection of item: (C# code)

```
    ListControlName1.ClearSelection();
```

AutoPostBack property: True means page is automatically post back to the server when we select item from list control. (except BulettedList)

SelectedIndexChanged event: Raised when we made selection and AutoPostBack property is true.

DropDownList control:



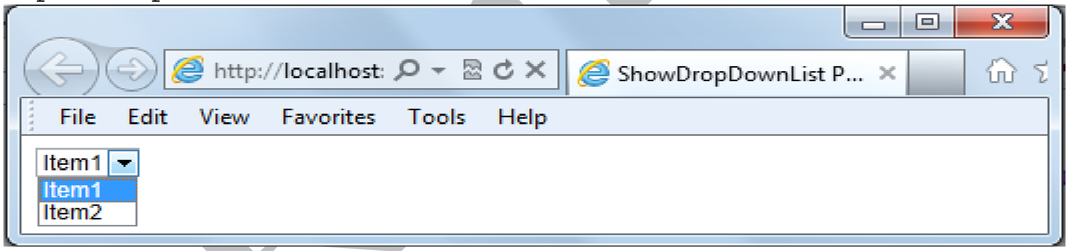
This control can hold a list of items but display any one of them. When we click on arrow button of this control then we get a list of items to select any one item for further processing using C# code.

Syntax:

```
  <asp:DropDownList ID="DropDownList1" runat="server">
  </asp:DropDownList>
```

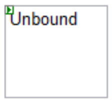
Display a list of items:

```
<asp: DropDownList ID=" DropDownList1 runat="server">
     <asp:ListItem Text="Item1" Value="value1"/>
     <asp:ListItem Text="Item2" Value="value2"/>
</asp: DropDownList>
```



SelectedIndexChanged event: Raised when we made selection and AutoPostBack property is true.

ListBox control:

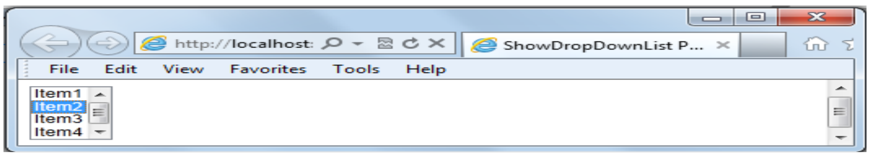


This control can display more items in list form and we can select multiple items.

Syntax:

```
  <asp:ListBox ID="ListBox1" runat="server"></asp:ListBox>
```

Display a list of items:



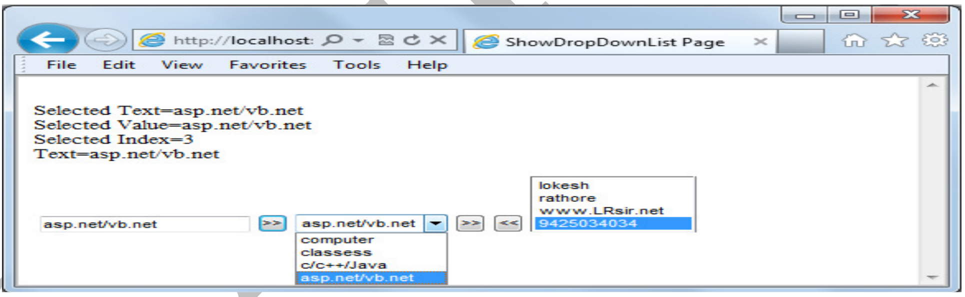
```
<asp:ListBox ID="ListBox1 runat="server">
     <asp:ListItem Text="Item1" Value="value1"/>
     <asp:ListItem Text="Item2" Value="value2"/>
     <asp:ListItem Text="Item3" Value="value3"/>
     <asp:ListItem Text="Item4" Value="value4"/>
</asp:ListBox>
```

Properties:
Rows: Number of visibles rows to display.
SelectionMode: single / muliple selection.

SelectedIndexChanged event: Raised when we made selection and AutoPostBack property is true.

Practical task of DropDownList and ListBox:

1. Read text from TextBox and add onto DropDownList control.
2. Show item details when item selected into DropDownList control.
3. Remove selected item from DropDownList and add into ListBox control.
4. Removes selected item from ListBox control and add into DropDownList control.



```
Selected Text=asp.net/vb.net
Selected Value=asp.net/vb.net
Selected Index=3
Text=asp.net/vb.net
```

asp.net/vb.net >> asp.net/vb.net >> <<

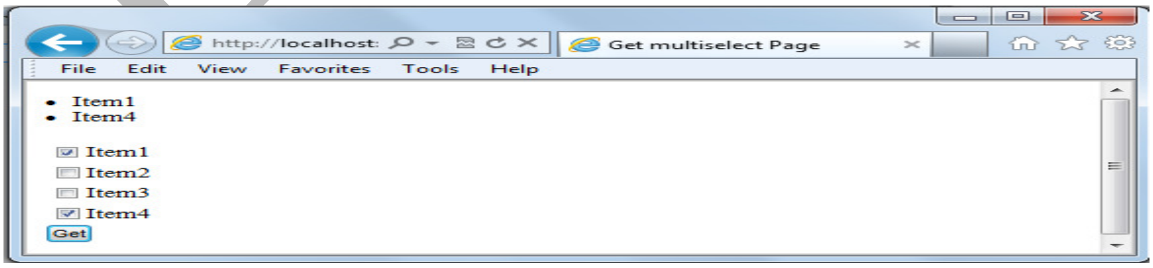
computer
classess
c/c++/Java
asp.net/vb.net

lokesh
rathore
www.LRsir.net
9425034034

```
<%@ Page Language="C#" %>
<script runat="server">
void Button1_Click(object sender, EventArgs e)
{
    ListItem item = new ListItem(TextBox1.Text, TextBox1.Text);
    DropDownList1.Items.Add(item);
    DropDownList1.Text = TextBox1.Text;
    TextBox1.Text = "";
    TextBox1.Focus();
}
void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
{
Response.Write("<br>Selected Text=" + DropDownList1.SelectedItem.Text);
Response.Write("<br>Selected Value=" + DropDownList1.SelectedValue);
Response.Write("<br>Selected Index=" + DropDownList1.SelectedIndex);
Response.Write("<br>Text=" + DropDownList1.Text);
}
```

```
void Button2_Click(object sender, EventArgs e)
{
  ListBox1.Items.Add(DropDownList1.SelectedItem);
  ListBox1.ClearSelection();
  DropDownList1.Items.Remove(DropDownList1.SelectedItem);
}

void Button3_Click(object sender, EventArgs e)
{
  ListItem item=ListBox1.SelectedItem;
  DropDownList1.Items.Add(item);
  DropDownList1.ClearSelection();
  ListBox1.Items.Remove(item);
}
</script>

<html>
<head><title>ShowDropDownList Page</title></head>
<body>
    <form id="form1" runat="server">
    <div>
    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    <asp:Button ID="Button1" runat="server"
     OnClick="Button1_Click" Text=">>" />
    <asp:DropDownList ID="DropDownList1" runat="server"
     AutoPostBack="True"
     OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged>
    </asp:DropDownList>
    <asp:Button ID="Button2" runat="server" Text=">>"
     OnClick="Button2_Click" />
    <asp:Button ID="Button3" runat="server"
     OnClick="Button3_Click" Text="<<" />
    <asp:ListBox ID="ListBox1" runat="server">
    </asp:ListBox></div>
    </form>
</body>
</html>
```

Practical Task: Get multi select items from CheckBoxList / ListBox control.



```
<%@ Page Language="C#" %>
<script runat="server">
void Button1_Click(object sender, EventArgs e)
{
  string str="";
  foreach (ListItem item in CheckBoxList1.Items)
  {
    if (item.Selected)
    {
     str += "<li>"+item.Text;
    }
  }
  Response.Write(str);
}
</script>

<html>
<head><title>Get multiselect Page</title></head>
<body>
    <form id="form1" runat="server">
    <div>
       <asp:CheckBoxList ID="CheckBoxList1" runat="server">
         <asp:ListItem Text="Item1" Value="Value1" />
         <asp:ListItem Text="Item2" Value="Value2" />
         <asp:ListItem Text="Item3" Value="Value3" />
         <asp:ListItem Text="Item4" Value="Value4" />
       </asp:CheckBoxList>
       <asp:Button ID="Button1" runat="server"
          OnClick="Button1_Click" Text="Get" /></div>
    </form>
</body>
</html>
```

Running a web Application:

In ASP.NET framework, we create a web form using HTML tags, CSS, JScript, HTML controls, server side ASP.NET web controls and C# / VB code. Such web form is saved using *.aspx* file extension.

When it runs on web server then all the server side contents are first executed at server side by .NET framework then reassemble web forms content and response to request user's browser using IIS.

Before running a web application first do following things for first starting web page among others.

First Solution Explorer→Select Web form→Right click→Set as start page.

In ASP.NET framework, web application can be run from many points.

1. Menubar→Debug→Start Debugging
2. tool bar→Click ▶ button
3. keyboard→Press F5 key
4. Solution Explorer→select web form→Right click→View in Browser

Creating a multiform web project:

A web site is a collection of interlinking web pages for a specific purpose. A web project may have multiple web forms. In ASP.NET we can add many web forms in current web project using following procedure.

1. Open solution explorer
2. Select root folder of web project and right click the mouse
3. click on **Add New Item** option
4. Select **web form** from a template list
5. Rename if required as **default.aspx**
6. Choose coding language C# / VB
7. Mark / unmark for place code in separate file
8. Click on ADD button.

All above steps are follows when we create multiple pages. All these pages are inlinking using navigate controls like **HyperLinking.**

UNIT-II

Form Validation:
- Client side validation
- Server Side validation

Validation Controls:
- Required Field
- Comparison
- Range
- Regular Expression
- Summary
- Custom

Calendar control
Ad rotator Control
Internet Explorer Control
State management:
- View state
- Session state
- Application state,

Form Validation:

A web form has many input controls like TextBoxs. These controls should have right type of data before submitting form at server side for data processing. Thus checking the input controls called form validation.

Form validation includes-

1. Check control has required value.
2. Check value of control falls between minimum and maximum value.
3. Compare value of control against another value.
4. Check value of control is match with given format.

For example: If data is not input into TextBox then an error message should be display. It is done using RequiredFieldValidator control.

Client side and server side Validation:

The validation of control can be performed at client side and server side.

1. **Client side validation:** When validation code is executed at client side by Browser software called client side validation. Client side validation prevents form submission to the server until all input values of control becomes error less. Validation code is written / generated in JScript for client side validation. Client side validation saves time for checking error that may consumes in between client to server and server to client.

2. **Server side validation:** When validation code is executed at server side by any programming language (C# / VB) called server side validation. Server side validation happens after form submission to the server. At server side, values of controls are checked. If any error founds then response to the client side using new web page. This process consumes more time as compared to client side validation.

Validation Controls:

ASP.NET provides following set of validation controls that may validate value of form control at client side (default) or server side.

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

These validation controls can be apply on any control that has decorated with ValidationProperty attribute.

RequiredFieldValidator control:

RequiredFieldValidator

This validation control checks to required value is input or not into specified input control before submitting the form. This control needs to link any one input control like TextBox.

Syntax:

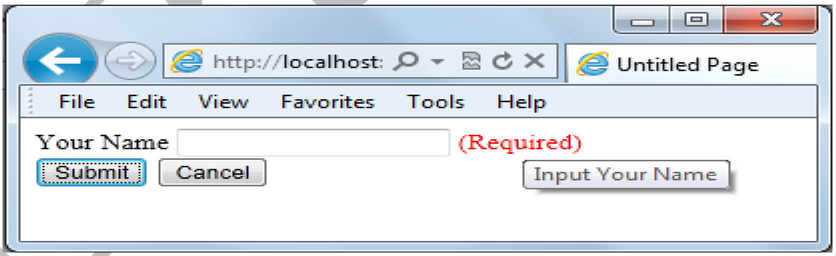
```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
runat="server" />
```

Important Properties:

- **ControlToValidate:** ID of control to be validate.
- **Text:** text to display for the validator when the validated control is invalid.
- **ErrorMessage:** Message to display in a ValidationSummary when the validated control is invalid.
- **ToolTip:** The tooltip displayed when the mouse is over the control.
- **EnableClientScript:** If it is true then validation is performed at client side by browser otherwise performed at server side.

CauseValidation property of Button: If false then button does not causes validation to fire.

Example:



Your Name (Required)

Input Your Name

```
<%@ Page Language="C#" %>
<html>
<head>
    <title>Untitled Page</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
   Your Name
   <asp:TextBox ID="TextBox1" runat="server" />
   <asp:RequiredFieldValidator ID="RequiredFieldValidator1"
     runat="server"
     ControlToValidate="TextBox1"
```

```
      ErrorMessage="Your Name is empty"
      ToolTip="Input Your Name"
      Text="(Required)" />
   <br />
   <asp:Button ID="Button1" runat="server" Text="Submit" />
   <asp:Button ID="Button2" runat="server" Text="Cancel"
      CausesValidation="False" />
   </div>
   </form>
</body>
</html>
```

RangeValidator control:

RangeValidator

This validation control checks input value falls between a certain minimum and maximum value. This control needs to link any one input control like TextBox.

Syntax:

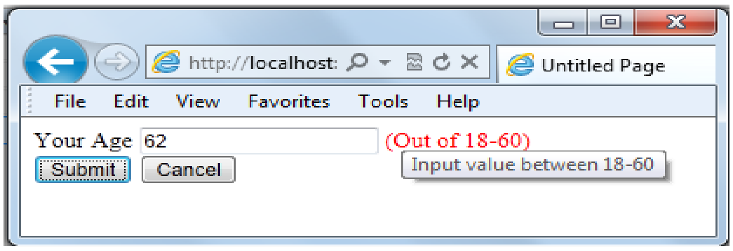
```
<asp:RangeValidator ID="RangeValidator1" runat="server" />
```

Important Properties:

- **ControlToValidate:** ID of control to be validate.
- **MinimumValue:** The minimum value for the control being validated.
- **MaximumValue:** The maximum value for the control being validated.
- **Type:** Data type of values for comparison. (String / Integer / Double / Date / Currency)
- **Text:** text to display for the validator when the validated control is invalid.
- **ErrorMessage:** Message to display in a ValidationSummary when the validated control is invalid.
- **ToolTip:** The tooltip displayed when the mouse is over the control.
- **EnableClientScript:** If it is true then validation is performed at client side by browser otherwise performed at server side.

CauseValidation property of Button: If false then button does not causes validation to fire.

Example:



Your Age 62 (Out of 18-60)

Input value between 18-60

```
<%@ Page Language="C#" %>
<html>
<head>
    <title>Untitled Page</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
   Your Age
   <asp:TextBox ID="TextBox1" runat="server" />
   <asp:RangeValidator ID="RangeValidator1" runat="server"
     ControlToValidate="TextBox1"
     Type="Integer"
     MinimumValue="18"
     MaximumValue="60"
     Text="(Out of 18-60)"
     ToolTip="Input value between 18-60"
     ErrorMessage="Out of range(18-60)" />
     <br />
  <asp:Button ID="Button1" runat="server" Text="Submit" />
  <asp:Button ID="Button2" runat="server" Text="Cancel"
     CausesValidation="False" />
  </div>
  </form>
</body>
</html>
```

CompareValidator control:

CompareValidator

This validation control performs three type of validation.

1. Check data type of value.
2. Compare input value against fixed value.
3. Compare input value against another input value.

Syntax:

```
<asp:CompareValidator ID="CompareValidator1"
runat="server" />
```

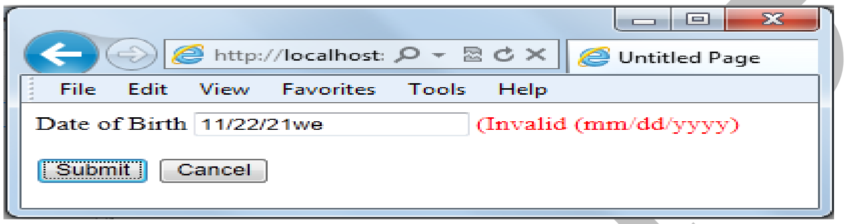
Important Properties:

- **ControlToValidate:** ID of control to be validate.
- **ControlToCompare:** ID of the control to compare with.
- **Type:** Data type of values for comparison. (String / Integer / Double / Date / Currency)
- **Operator:** Comparison operation to apply to value. (Equal / NotEqual / GreaterThan / …)
- **ValueToCompare:** The fixed value to compare against.

- **Text:** text to display for the validator when the validated control is invalid.
- **ErrorMessage:** Message to display in a ValidationSummary when the validated control is invalid.
- **ToolTip:** The tooltip displayed when the mouse is over the control.
- **EnableClientScript:** If it is true then validation is performed at client side by browser otherwise performed at server side.

CauseValidation property of Button: If false then button does not causes validation to fire.

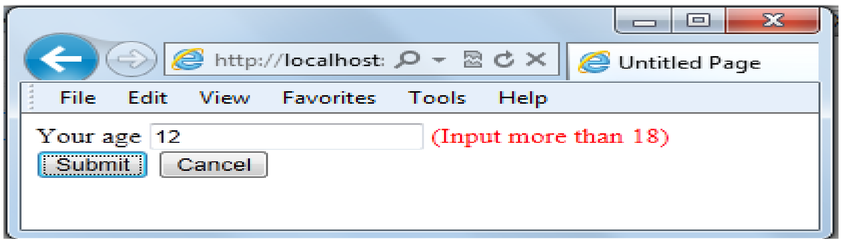
Example1: Check data type.



Date of Birth 11/22/21we (Invalid (mm/dd/yyyy)

```
<%@ Page Language="C#" %>
<html>
<head><title>Untitled Page</title></head>
<body>
  <form id="form1" runat="server">
    <div>
      Date of Birth
          <asp:TextBox ID="TextBox1" runat="server"/>
          <asp:CompareValidator ID="CompareValidator1"
          runat="server" ControlToValidate="TextBox1"
          Operator="DataTypeCheck"
          Type="Date"
          Text="(Invalid (mm/dd/yyyy)" />
   <br />
   <asp:Button ID="Button1" runat="server" Text="Submit" />
   <asp:Button ID="Button runat="server" Text="Cancel"
       CausesValidation="False" />
   </div>
  </form>
</body>
</html>
```

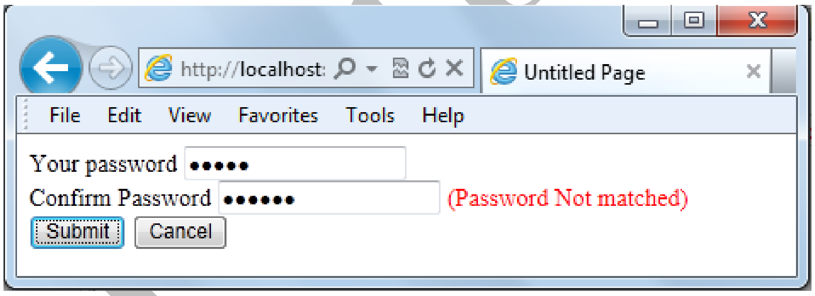
Example2: Check input integer value is more than 18.



Your age 12 (Input more than 18)

```
<%@ Page Language="C#" %>
<html>
<head><title>Untitled Page</title></head>
<body>
  <form id="form1" runat="server">
    <div>
      Your age
          <asp:TextBox ID="TextBox1" runat="server"/>
          <asp:CompareValidator ID="CompareValidator1"
          runat="server"
          ControlToValidate="TextBox1"
          Operator="GreaterThan"
          Type="Integer"
          ValueToCompare="18"
          Text="(Input more than18)" />
   <br />
   <asp:Button ID="Button1" runat="server" Text="Submit" />
   <asp:Button ID="Button runat="server" Text="Cancel"
       CausesValidation="False" />
   </div>
  </form>
</body>
</html>
```

Example3: Compare input password to confirm password.



Your password ●●●●●
Confirm Password ●●●●●● (Password Not matched)
Submit Cancel

```
<%@ Page Language="C#" %>
<html>
<head><title>Untitled Page</title></head>
<body>
  <form id="form1" runat="server">
    <div>
      Your password
        <asp:TextBox ID="TextBox1" runat="server"
          TextMode="Password" />
        <br />
        Confirm Password
        <asp:TextBox ID="TextBox2" runat="server"
          TextMode="Password" />
        <asp:CompareValidator ID="CompareValidator1"
          runat="server" ControlToCompare="TextBox1"
          ControlToValidate="TextBox2"
          Type="String"
```

```
              Operator="Equal"
              Text="(Password Not matched)" />
       <br />
       <asp:Button ID="Button1" runat="server" Text="Submit" />
       <asp:Button ID="Button runat="server" Text="Cancel"
           CausesValidation="False" />
       </div>
      </form>
   </body>
   </html>
```

RegularExpressionValidator control:

RegularExpressionValidator

This validation control compares input value against a regular expression. We can use a regular expression to represent string pattern such as email address, dates etc.

Syntax:

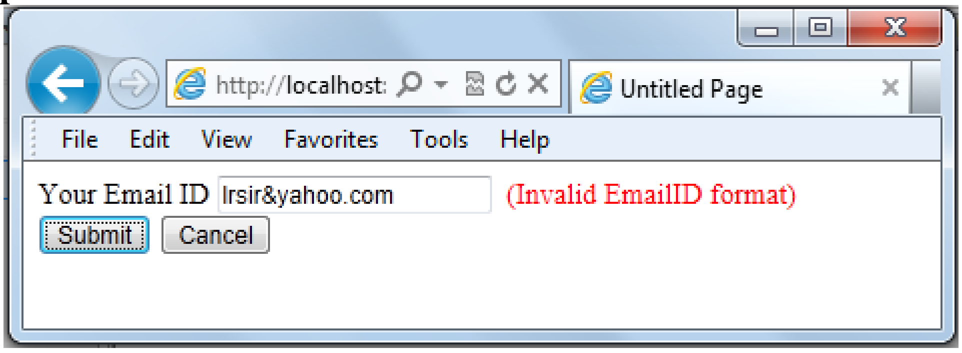
```
<asp:RegularExpressionValidator ID="RegularExpressionValidator1"

runat="server" />
```

Important Properties:

- **ControlToValidate:** ID of control to be validate.
- **ValidationExpression:**The Regular expression is assigned to this property. (http://regexlib.com for all list)
- **Text:** text to display for the validator when the validated control is invalid.
- **ErrorMessage:** Message to display in a ValidationSummary when the validated control is invalid.
- **ToolTip:** The tooltip displayed when the mouse is over the control.
- **EnableClientScript:** If it is true then validation is performed at client side by browser otherwise performed at server side.

CauseValidation property of Button: If false then button does not causes validation to fire.

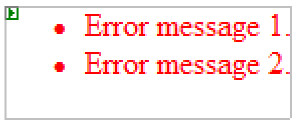
Example: Check valid EmailID format



Your Email ID lrsir&yahoo.com (Invalid EmailID format)
Submit Cancel

```
<%@ Page Language="C#" %>
<html>
<head><title>Untitled Page</title></head>
<body>
  <form id="form1" runat="server">
    <div>
      Your Email ID
          <asp:TextBox ID="TextBox1" runat="server"/>
          <asp:RegularExpressionValidator
          ID="RegularExpressionValidator1" runat="server"
          ControlToValidate="TextBox1"
          ValidationExpression=
          "\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*"
          Text="(Invalid EmailID format" />
     <br />
     <asp:Button ID="Button1" runat="server" Text="Submit" />
     <asp:Button ID="Button runat="server" Text="Cancel"
       CausesValidation="False" />
     </div>
  </form>
</body>
</html>
```

ValidationSummary control:



This control display a list of all validation errors which are given in ErrorMessage of each validation control.

Syntax:

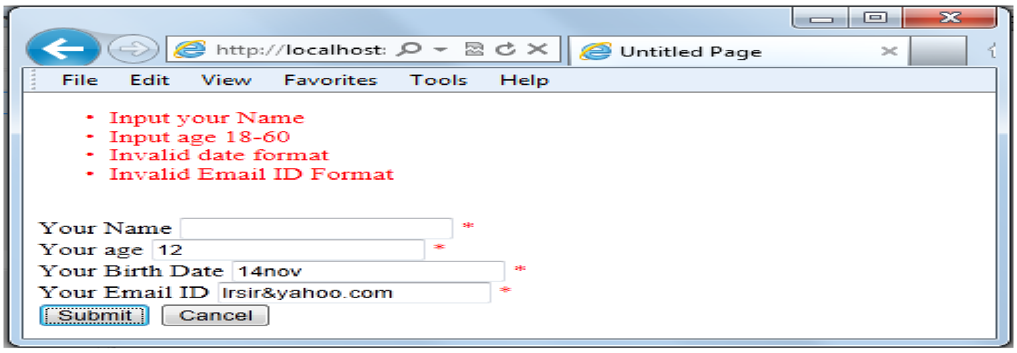
```
<asp:ValidationSummary ID=" ValidationSummary1"

runat="server" />
```

Important Properties:

- **DisplayMode:** Format for error summary display. (BulletList / SingleParaGraph / List)
- **HeaderText:** To display header text on the top of summary control.
- **ShowMessageBox:** True means display a popup alert box.
- **ShowSummary:** False then summary hides.

CauseValidation property of Button: If false then button does not causes validation to fire.

Example: Show summary of validation of all validation controls.



- Input your Name
- Input age 18-60
- Invalid date format
- Invalid Email ID Format

Your Name *
Your age 12 *
Your Birth Date 14nov *
Your Email ID lrsir&yahoo.com *
Submit Cancel

```
<%@ Page Language="C#" %>
<html>
<head>
    <title>Untitled Page</title></head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:ValidationSummary ID="ValidationSummary1"
          runat="server" />
        <br />
        Your Name
        <asp:TextBox ID="TextBox1" runat="server"/>
        <asp:RequiredFieldValidator
          ID="RequiredFieldValidator1" runat="server"
          ControlToValidate="TextBox1"
          ErrorMessage="Input your Name"
          Text="*" />
          <br />
        Your age
        <asp:TextBox ID="TextBox2" runat="server" />
        <asp:RangeValidator ID="RangeValidator1"
          runat="server" ControlToValidate="TextBox2"
          ErrorMessage="Input age 18-60"
          MaximumValue="60" MinimumValue="18"
          Type="Integer" Text="*" />
          <br />
        Your Birth Date
        <asp:TextBox ID="TextBox3" runat="server" />
        <asp:CompareValidator ID="CompareValidator1"
          runat="server"
          ControlToValidate="TextBox3"
          ErrorMessage="Invalid date format"
          Operator="DataTypeCheck"
          Type="Date" Text="*" />
          <br />
        Your Email ID
        <asp:TextBox ID="TextBox4" runat="server"/>
        <asp:RegularExpressionValidator
          ID="RegularExpressionValidator1" runat="server"
          ControlToValidate="TextBox4"
```

```
              ErrorMessage="Invalid Email ID Format"
              ValidationExpression=
              "\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*"
              Text="*" />
              <br />
            <asp:Button ID="Button1" runat="server"
              Text="Submit" />
            <asp:Button ID="Button2" runat="server"
              Text="Cancel" CausesValidation="False" />
        </div>
     </form>
</body>
</html>
```

CustomValidator control:

CustomValidator

If none of the other validation controls perform the type of validation that we need then we can use CustomValidator control. In this control, we can associate function for a custom validation.

Syntax:

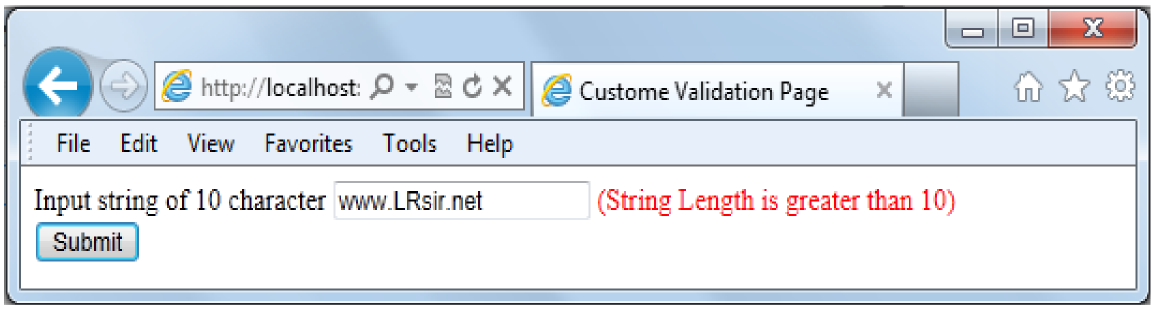
```
<asp:CustomValidator ID="CustomValidator1"

runat="server" />
```

Important Properties:

- **ControlToValidate:** ID of control to be validate.
- **Text:** text to display for the validator when the validated control is invalid.
- **ErrorMessage:** Message to display in a ValidationSummary when the validated control is invalid.
- **ServerValidate(Event):** This event raised when the CustomValidator performs validation.

CauseValidation property of Button: If false then button does not causes validation to fire.

Example: Check input string length is greater than 10.



Input string of 10 character www.LRsir.net (String Length is greater than 10)
Submit

```
<%@ Page Language="C#" %>
<script runat="server">
  void CustomValidator1_ServerValidate(object source,
                             ServerValidateEventArgs args)
    {
        if (args.Value.Length <= 10)
            args.IsValid = true;
        else
            args.IsValid = false;
    }
</script>

<html>
<head><title>Custome Validation Page</title></head>
<body>
    <form id="form1" runat="server">
    <div>
     Input string of 10 character
     <asp:TextBox ID="TextBox1" runat="server" />
     <asp:CustomValidator ID="CustomValidator1"
     runat="server" ControlToValidate="TextBox1"
     OnServerValidate="CustomValidator1_ServerValidate"
     Text="(String Length is greater than 10) />
     <br />
    <asp:Button ID="Button1" runat="server"
     Text="Submit" />
</div>
    </form>
</body>
</html>
```

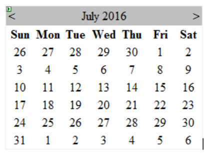
Basics of Regular expression

```
d    (for digits)
w    (for character)
{n}  (n is exact number of allowed digits or character)
[a-z](match any one specified character)
+    (for one or more d/w)
\    (flow of d/w)
```

for example:

```
\d{1} to allow only one digits  ex: 0 to 9
\d{5} to allow only 5 digits    ex:00000 to 99999
\w{6} to allow only 6 character ex: lokesh, ujjain, p-1234
\d+   to allow one or more digits   ex: 1, 21, 345 5444
\w+   to allow one or more digits   ex: l, lo, lok5444

d{4}(-d{5})  to allow 4 digits - 5 digits (3234-32434)

***for one character**
[a]
[ad]
[a-z]
[A-Z]
[0-9]
[-+*/]

***one or more ***
     digits(233)               : \d+
     Small Alphabet(lokesh)    : [a-z]+
     Capital Alphabet(lokesh) : [A-Z]+
     digits with one . dot(233.23) : \d+(.\d+)
***fix number of ***
     5 digits(233)         : \d{5}
     5Small Alphabet(lokesh)   : [a-z]{5}
     (5,2)digits with one . dot(233.23) : \d{5}(.\d{2})
```

Calendar control:



<			July 2016			>
Sun	Mon	Tue	Wed	Thu	Fri	Sat
26	27	28	29	30	1	2
3	4	5	6	7	8	9
10	11	12	13	14	15	16
17	18	19	20	21	22	23
24	25	26	27	28	29	30
31	1	2	3	4	5	6

The Calendar control displays a calendar that can use as a date picker or to display a list of upcoming events.

Syntax:

```
<asp:Calendar ID="Calendar1" runat="server"/ >
```

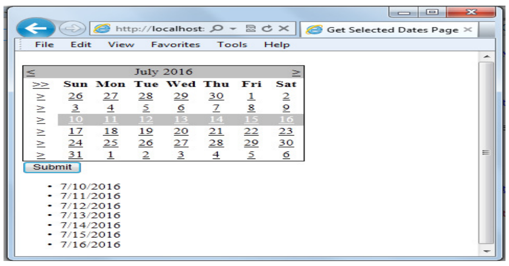
Properties:

- Caption: The caption associated with calender.
- DayNameFormat: To set day name as(Mon/Monday/Mo/M.
- FirstDayOfWeek:Which day of week display first. (Sun/Mon/Tue..).
- NextMonthText: To set text for next month Button.(&gt for >).
- PrevMonthText: To set text for previous month Button. (&lt for <).
- NextPrevFormat: To set format for Next and Previous month navigation buttons.(ShortMonth / FullMonth/ CustomText).
- SelectedDate: To get or set selected date.
- SelectedDates: To get multiple dates(C#).
- ShowDayHeader: If false then dayname becomes hides.
- ShowNextPrevMonth: If false then next and previous month hides.
- ShowTitle: If false then title of calender hides.
- TitleFormat: Set Month or Month Year.
- VisibleDate: Set the month of calender.
- SelectionMode: To set selection day/week/month.

Events:

- DayRender: Raised as each day is rendered.
- SelectionChanged: Raised when a new day, week, or month is selected.
- VisibleMonthChanged: Raised when the next or previous month link is clicked.

Example: Show all dates of selected week in a bullet list control.



<				July 2016			>
>>	Sun	Mon	Tue	Wed	Thu	Fri	Sat
>	26	27	28	29	30	1	2
>	3	4	5	6	7	8	9
>	10	11	12	13	14	15	16
>	17	18	19	20	21	22	23
>	24	25	26	27	28	29	30
>	31	1	2	3	4	5	6

- 7/10/2016
- 7/11/2016
- 7/12/2016
- 7/13/2016
- 7/14/2016
- 7/15/2016
- 7/16/2016

```
<%@ Page Language="C#" %>
<script runat="server">
void Button1_Click(object sender, EventArgs e)
{
    BulletedList1.DataSource = Calendar1.SelectedDates;
    BulletedList1.DataBind();
}
</script>

<html>
<head><title>Get Selected Dates Page</title></head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Calendar ID="Calendar1" runat="server"
          SelectionMode="DayWeekMonth"></asp:Calendar>
        <asp:Button ID="Button1" runat="server"
          OnClick="Button1_Click" Text="Submit" />
        <asp:BulletedList ID="BulletedList1" runat="server"
          DataTextFormatString="{0:d}"></asp:BulletedList>
    </div>
    </form>
</body>
</html>
```

AdRotator control:

The AdRotator control is used to randomly display images of different advertisements in a page. List of advertisements can be stored in an XML file or in a database table.

Syntax:

```
<asp:AdRotator ID="AdRotator1" runat="server" />
```

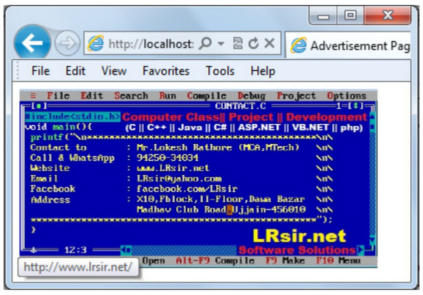
Properties:

- AdvertisementFile: To specify path of XML file containing advertisements.
- AlternateTextField: The element name (AlternateText) that specify which alternate text to retrieve.
- ImageUrlField: The element name (ImageUrl) that specify which image URL to retrieve.
- NavigateUrlField: The element name (NavigateUrl) that specify which advertisement web page URL to retrieve.

Events:

- AdCreated: Raised after the AdRotator control selects an advertisement but before the AdRotator control renders the advertisement.

Procedure to add advertisements on AdRotator control using XML file.


http://localhost:
Advertisement Pag
File Edit View Favorites Tools Help
≡ File Edit Search Run Compile Debug Project Options
CONTACT.C
12:3
Open Alt-F9 Compile F9 Make F10 Menu
http://www.lrsir.net/

```
#include<stdio.h> Computer Class|| Project || Development
void main(){       (C || C++ || Java || C# || ASP.NET || VB.NET || php)
 printf("\n*******************************************\n\
 Contact to        : Mr.Lokesh Rathore (MCA,MTech)   \n\
 Call & WhatsApp   : 94250-34034                     \n\
 Website           : www.LRsir.net                   \n\
 Email             : LRsir@yahoo.com                 \n\
 Facebook          : facebook.com/LRsir              \n\
 Address           : X10,Fblock,II-Floor,Dawa Bazar  \n\
                     Madhav Club Road,Ujjain-456010  \n\
*******************************************");
}
                                        LRsir.net
                                        Software Solutions
```

1) Add AdRotator control on web page. (AdRotator1)
2) Create AdImage folder in root folder of website.
3) Copy Ad Images into AdImage folder.(let ad1.jpg, ad2.jpg)
4) Open an XML: Add New Item→add XML file(adXML.xml)
5) Write following XML code to add list of ad images.

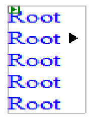
```
<Advertisements>
  <Ad>
      <ImageUrl>~/AdImage/Ad1.jpg</ImageUrl>
      <AlternateText>LRsir</AlternateText>
      <NavigateUrl>http://www.LRsir.net</NavigateUrl>
      <Impressions>50</Impressions>
      <Keyword>banner</Keyword>
      <Width>400</Width>
      <Height>200</Height>
  </Ad>

  <Ad>
      <ImageUrl>~/AdImage/Ad2.jpg</ImageUrl>
      <AlternateText>Advance College</AlternateText>
      <NavigateUrl>www.advcol.com</NavigateUrl>
      <Impressions>20</Impressions>
      <Keyword>banner</Keyword>
      <Width>400</Width>
      <Height>200</Height>
  </Ad>
</Advertisements>
```

6) Attach adXML.xml file to AdvertisementFile property and add Keyword value to KeywordFilter property.

```
<asp:AdRotator ID="AdRotator1" runat="server"
KeywordFilter="banner"
AdvertisementFile="~/AdXMLFile.xml" />
```

When web page is refreshed then Ad images replaced by another Ad images. Number of occurrence of any advertisement depends upon impressions value.

Menu control: Internet Explorer Control



This control is used to create vertical and horizontal list of link in drop down menu format.

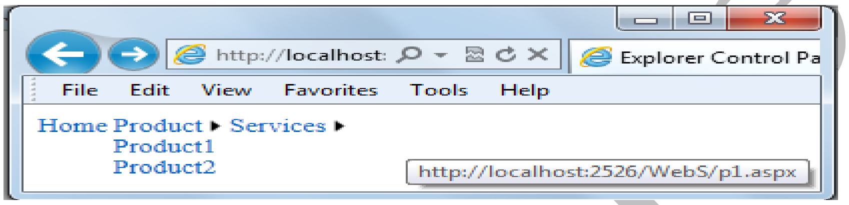
Syntax:

```
<asp:Menu ID="Menu1" runat="server"/>
```

Properties:

- Orientation: To set Vertical / Horizontal menu.
- Items: To add collection of links.

Example: Create Horizontal menu.



```
<%@ Page Language="C#" %>
<html>
<head><title>Explorer Control Page</title></head>
<body>
    <form id="form1" runat="server">
    <div>
      <asp:Menu ID="Menu1" runat="server"
          Orientation="Horizontal">
        <Items>
          <asp:MenuItem Text="Home"
          NavigateUrl="~/default.aspx" />
          <asp:MenuItem Text="Product">
               <asp:MenuItem Text="Product1"
                    NavigateUrl="~/p1.aspx" />
               <asp:MenuItem Text="Product2"
                    NavigateUrl="~/p2.aspx" />
          </asp:MenuItem>
          <asp:MenuItem Text="Services">
               <asp:MenuItem Text="Service1"
                    NavigateUrl="~/s1.aspx" />
               <asp:MenuItem Text="Service2"
                    NavigateUrl="~/s2.aspx" />
          </asp:MenuItem>
        </Items>
      </asp:Menu>
    </div>
    </form>
</body>
</html>
```

ASP.NET State Management:

When user request same page or different page then server cleans up all the created variable and object after serving that page to the user. State management is the process by which we store information between multiple requests for the same page or different pages.

In ASP.NET, We can use following three type of state management system.

1) ViewState
2) Session
3) Application

ViewState:

It is a client side page level state management technique i.e. as long as the user is on the current page, state is available and when user redirect to the next page then state is lost.

It is used when user needs to preserve data temporarily after a post back then the ViewState stores data in the generated HTML using hidden field.

View State can store any type of data because it is object type. View state is enabled by default for all ASP.NET controls.

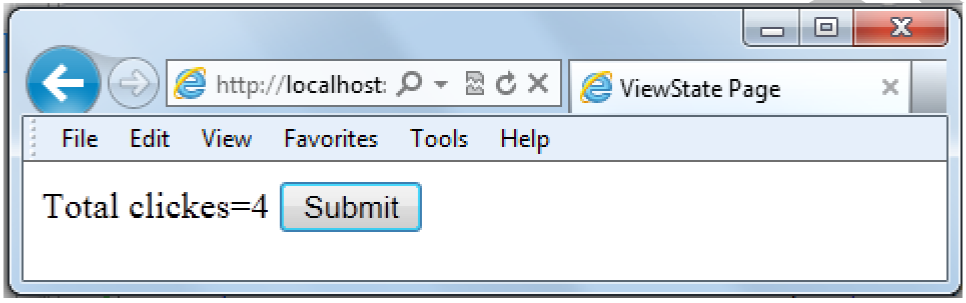
Syntax: **`ViewState["Variable_Name"]`**

We can create a number of ViewState variables.

Example: Count number of button's click

```
<%@ Page Language="C#" %>
<script runat="server">
void Button1_Click(object sender, EventArgs e)
{
     Label1.Text = ViewState["count"].ToString();
}
void Page_Load(object sender, EventArgs e)
{
    if (IsPostBack)
    {
      if (ViewState["count"] == null)
      {
        ViewState["count"] = "1";
      }
      else
      {
         int i = Convert.ToInt32(ViewState["count"]) + 1;
         ViewState["count"] = i.ToString();
      }
    }
 }
```

```
</script>

<html>
<head><title>ViewState Page</title></head>
<body>
    <form id="form1" runat="server">
    <div>
        <asp:Label ID="Label1" runat="server" />
        <asp:Button ID="Button1" runat="server" Text="ok"
          OnClick="Button1_Click"  />
    </div>
    </form>
</body>
</html>
```



Total clickes=4

Session:

It is a server side state management technique i.e. as long as the user is on current page or next page, state is always available and when user idles up to specified session time then session state lost. The server maintains the state of user information by using a session ID.

When user makes a request without a session ID, ASP.NET creates a session ID and sends it with every request and response to the same user.

Syntax: **`Session["Variable_Name"]`**

Session Events: We can executes codes when session starts or end using `Session_Start` and `Session_End` events. Session events are defining in **Global.asax** file that creates into root folder of website.

```
void Session_Start(object sender, EventArgs e)
{
  // Write code that runs when a new session is started
}
void Session_End(object sender, EventArgs e)
{
  // Write code that runs when a active session is expired
}
```

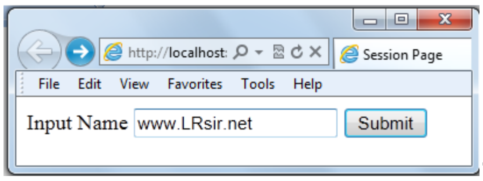
The Session_End event is raised when session ends either because of a time out expiry or explicitly by using Session.Abandon().

Set Session Time: Add <sessionState> into **web.config** file of root folder.

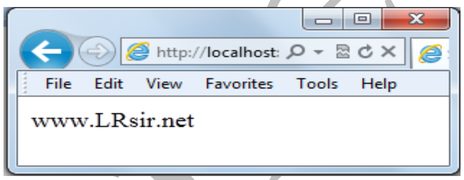
```
<configuration>
     <system.web>
         <sessionState timeout="10" mode="InProc" />
     </system.web>
</configuration>
```

The `Session_End` event is raised only in the case of `InProc` mode not in the state server and SQL Server modes.

Example: Assign name into one page and display into next page. Set 1 minute for session time.



→



www.LRsir.net

1. **Global.aspx**

```
 void Session_Start(object sender, EventArgs e)
 {
     Session["username"] = "none";
  }
```

2. **web.config**

```
<sessionState timeout="10" mode="InProc" />
```

3. **session.aspx**

```
<%@ Page Language="C#" %>
<script runat="server">
Button1_Click(object sender, EventArgs e)
    {
        Session["username"] = TextBox1.Text;
        Response.Redirect("~/nextPage.aspx");
    }
</script>
<html>
<head><title>Session Page</title></head>
<body>
    <form id="form1" runat="server">
    <div>
     Input Name
     <asp:TextBox ID="TextBox1" runat="server" />
     <asp:Button ID="Button1" runat="server"
     Text="Submit" OnClick="Button1_Click" />
    </div>
```

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

4. nextpage.aspx

```
    <%@ Page Language="C#" %>
    <script runat="server">
    void Page_Load(object sender, EventArgs e)
    {
       Label1.Text = Session["username"].ToString();
    }
    </script>

    <html>
    <head><title>Next Page</title>
    </head>
    <body>
        <form id="form1" runat="server">
        <div>
            <asp:Label ID="Label1" runat="server" />
        </div>
        </form>
    </body>
    </html>
```

Application:

Application state is a server side state management technique. The data stored in application state is common for all users of that particular ASP.NET application and can be accessed anywhere in the application. It is also called application level state management. Data stored in the application should be of small size.

Syntax: `Application["Variable_Name"]`

Application Events:

Application events are defining in **Global.asax** file that creates into root folder of website.

`Application_Start`: It is raised when the first request is made using domain(www.LRsir.net) of web site.

```
void Application_Start(object sender, EventArgs e)
{
  // Write code
}
```

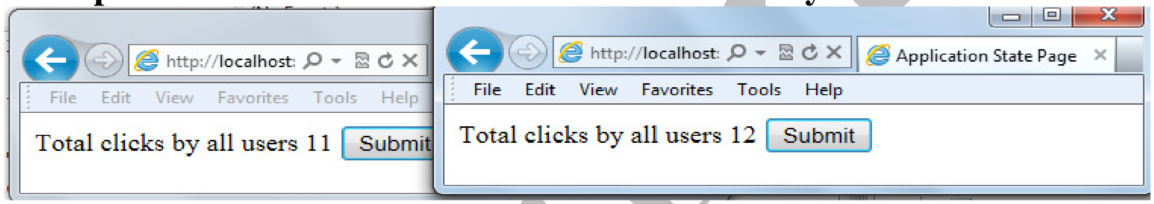
Application_End: It is raised just before the domain ends, server restart, when the first request is made using domain(www.LRsir.net) of web site.

```
void Application_End(object sender, EventArgs e)
{
  // Write code
}
```

Application_Error: It is raised when an exception is occurs then it can be handles using this event.

```
void Application_Error(object sender, EventArgs e)
{
  // Write code
}
```

Example: Count total number of clicks of button by all users.



Total clicks by all users 11

Total clicks by all users 12

1. **Global.aspx**

```
    void Application_Start(object sender, EventArgs e)
    {
        Application["count"] = 0;
     }
```

2. **Application.aspx**

```
<%@ Page Language="C#" %>
<script runat="server">
void Button1_Click(object sender, EventArgs e)
{
        int i=Convert.ToInt32(Application["count"]) + 1;
        Application["count"] = i.ToString();
        Label1.Text = i.ToString();
}
</script>
<html>
<head><title>Application State Page</title></head>
<body>
    <form id="form1" runat="server">
    <div>
        Total clicks by all users
        <asp:Label ID="Label1" runat="server"/>
        <asp:Button ID="Button1" runat="server"
          OnClick="Button1_Click" Text="Submit" />
```

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

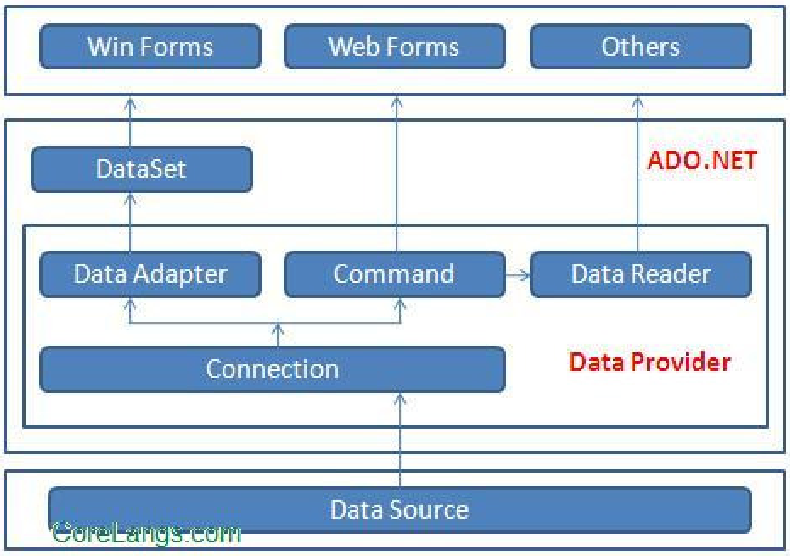
UNIT-III

Architecture of ADO.NET
Connected and Disconnected Database
Create Connection using ADO.NET Object Model
Connection Class
Command Class
DataAdapter Class
Dataset Class
Database Accessing on web applications:
    Data Binding concept with web
    Creating data grid
    Binding standard web server controls
    Display data on web form using Data bound controls.

Architecture of ADO.NET:

The full form of ADO.NET is ActiveX Data Object.Net. Basically it is container of all the standard classes which are responsible for database connectivity of .net application to the any kind of third party database software like Ms Sql Server, Ms Access, MySql, Oracle or any other database software.

All classes belongs to ADO.NET are defined and arrange in *"System.Data"* namespace. The architecture of ADO.NET is following.



From above diagram it is clear that ADO.NET contain following important components.
1. Connection
2. Command
3. DataReader
4. DataAdapter
5. Dataset

1. Connection: This component of Ado.net is responsible to connect our .net application with any data source like Ms sql server database.

2. Command: This component contain classes for executing any INSERT, UPDATE, DELETE AND SELECT command over database software using active connection.

3. DataReader: This component contains classes that capable to hold reference of records that retrieved by Command class after executing SELECT Query.

4. Data Adapter: This component is capable to perform the entire task like executing INSERT, UPDATE, DELETE and SELECT command on give Connection. The most important use of this component is executing SELECT query for a number of records and fill up to Dataset components.

5. Dataset: This component is the main source of records for .net control like GridView. It has capability to generate a number of tables to hold records retrieved from DataAdapter or XML.

Connection String and Connection Class:

The first and essential part for data communication between .NET application and any database software like Ms Sql Server is connection establishments between them.

The connection will established using *driver name, data source file name and security options* that enclosed in string formats called *connection string*. It means right connection string is only responsible for establishing connection with required database file.

In ASP.NET, connection string should be written in ***web.config*** file under root directory of web site so that such connection string can be available to all code behind of web forms.

Let ***database.mdf*** is a Ms sql server database file created using inbuilt tools of Visual studio 2005. Then the Format of connection string is written in web.config file as following.

```
<connectionStrings>
     <add name="constr"
     connectionString="Data Source= .\SQLEXPRESS;
     AttachDbFilename=|DataDirectory|\Database.mdf;
     Integrated Security=True ;
     User Instance=true"/>
</connectionStrings>
```

Connection String can be copy and paste from server explorer→Select database→property window→copy conection string.

In this ***web.config*** file connection string is enclosed in XML tag format.

Getting connection string on code behind:

Following namespace must be included in code behind of every page to get connection string.

```
          using System.Configuration;
```

After then we declare a string variable to store connection string that getting from web.config file. It is written as.

```
string constr1= ConfigurationManager.
ConnectionStrings["constr"]. ConnectionString;
```

such connection string will enable connection with "*Database.mdf*" file that created using MS-SQL Server database software.

Establishing Connection:

We need to connect our .NET application with required database file before data communication. For this operation the connection must be opened and connection will be open by Connection Class provided by ADO.NET. To estblising connection with different database file, we need Connection Class.

Working with Connection Class:

Step 1: Create a connection string in web.config file.
Step 2: Include namespace: `System.Data.SqlClient.`
Step 3: Get Connection String in required Code Behind.
Step 4: Create Object of Connection Class and pass connection string to this object.
Step 5: Open connection.
Step 6: Perform data access operation.
Step 7: Close active connection.
Above steps can be implemented as.

Namespace: If we want to connect a ***Database.mdf*** file that created under MS-Sql Server Software then Code Behind must included following namespace.

```
using System.Data.SqlClient;
```

Connection Class:

This namespace support following Connection Class to establishing connection.

```
SqlConnection
```

Before establishing connection, we need to create an connection object and passing connection string to the constructor of Connection class like this-

```
SqlConnection con= new SqlConnection(constr1);
```

here:

con = Connection object
constr1= a string variable that contain Connection String

Opening Connection: When Connection Object is created then by calling Open() method of Connection Class, we can create an active connection. Like-

```
con.Open();
```

When this method executes then connection will be establised with required database file.

Closing Connection: When all data communication has been performed over this active connection, then connection must be dis connected so that connection object can be further used with same database file or different database file. Like-

```
con.Close();
```

Command Class:

This is one of component of ADO.NET. With the help of this componenet we can assign SQL command and execute on database software using active connection.

```
SqlCommand
```

It is command class for MS-SQL Server Database.

Working process with Command Class:(for MS-SQL Server database)

Step1: Include ADO.NET namespace in code behind.

Ex: `using System.Data.SqlClient ;`

Step2: Open Database Connection.

Ex:

```
SqlConnection con=new SqlConnection(constr1);
con.Open();
```

Here *constr1* is Connection String for required database connectivity.

Step3: Write Required SQL command like INSERT / UPDATE / DELETE in string formate.

Ex:

To add new record

```
string sql= "INSERT INTO Book (BookName, Price)
VALUES('ASP.NET', 500)";
```

To update new record

```
string sql= "UPDATE Book SET BookName= 'C#.NET',
Price='400' WHERE BookID=1";
```

To delete any record

```
string sql= "DELETE FROM Book WHERE BookID=1";
```

Step4: Create Object of Command Class then Pass Sql Command and Active Connection to the Constructor of command class.

Ex:

```
SqlCommand cmd = new SqlCommand(sql, con);
```

Here-

sql= SQL Command in string form.

con= Active Connection.

Step5: Execute INSERT / UPDATE / DELETE Command using *ExecuteNonQuery()* method of Command class.

Ex:

```
cmd.ExecuteNonQuery();
```

When all above steps are complete then required sql command will be executed on active connection sucessfully.

DataReader Class:

Data Reader class in one of ADO.NET's components. The purpose of this class is to hold the reference of records that retived after executing SELECT command using ***ExecuteReader()*** method of ***Command*** Class.

`SqlDataReader`

It is a DataReader class for MS-SQL Server Database. ***Read()*** method of DataReader class will return **true** value if records found otherwise **false**. *Using index value(0,1,..)* of DataReader object, we can retieved column value of records.

Working process with DataReader Class:(for MS-SQL Server database)

Step1: Include ADO.NET namespace in code behind.

Ex: `using System.Data.SqlClient ;`

Step2: Open Database Connection.

Ex:

```
SqlConnection con = new SqlConnection(constr1);
con.Open();
```

Here *constr1* is Connection String for required database connectivity.

Step3: Write Required SELECT SQL command in string formate.

Ex:

```
string sql= "SELECT * FROM Book WHERE BookID=1";
```

Step4: Create Object of Command Class then Pass Sql Command and Active Connection to the Constructor of command class.

Ex: `SqlCommand cmd = new SqlCommand(sql, con);`

Here-

sql= SQL Command in string form.

con= Active Connection.

Step5: Create Object reference of DataReader class.

Ex: `SqlDataReader dr;`

Step6: Execute SELECT sql Command using ***ExecuteReader()*** method of Command class and assign reference of retrieved records to the DataReader object reference.

Ex: `dr=cmd.ExecuteReader();`

Step7: Check availability of retirieved records in to DataReader using ***Read()*** method. If found then retrives column's value into controls using idex value.

Ex:

```
if(dr.Read()==true)
{
TextBox1.Text=dr[0].ToString(); // First Column Value
TextBox2.Text=dr[1].ToString(); // Second Column Value
TextBox3.Text=dr[2].ToString(); // Third Column Value
}
```

When all above steps are complete then required SELECT sql command will be executed on active connection sucessfully and records will be shown on destination control.

Data Adapter and Datset Class:

DataAdapter:

DataAdapter class of ADO.NET can be be used to execute any SQL comaand on given connection and retrieved records can be filled into given Dataset.

`SqlDataAdapter`

It is a DataAdapter class for MS-SQL Server.

Fill() method of DataAdapter class that can be used to papulate dataset.

Dataset:

Dataset is a class of ADO.NET which is used to store records that retrieved from any source like MS-SQL server, or XML files. When Dataset is papulated by records from any source then records of Dataset can be shown on any Data bind control like GridView.

Working process with DataAdapter and Dataset Class:(for MS-SQL Server database)

Step1: Include ADO.NET namespace in code behind.

Ex: `using System.Data.SqlClient ;`

Step2: Open Database Connection.

Ex:

```
SqlConnection con = new SqlConnection(constr1);
con.Open();
```

Here *constr1* is Connection String for required database connectivity.

Step3: Write Required SELECT SQL command in string formate.

Ex:

```
string sql= "SELECT * FROM Book";
```

Step4: Create Object of Command Class then Pass Sql Command and Active Connection to the Constructor of command class.

Ex: `SqlCommand cmd = new SqlCommand(sql, con);`

Here-

sql= SQL Command in string form.

con= Active Connection.

Step5: Create Object of DataAdapter class then pass object of Command Class to its Constructur.

Ex: `SqlDataAdapter da = new SqlDataAdapter (cmd);`

Step6: Create Object of Dataset class.

Ex: `Dataset ds = new Dataset();`

Step7: Call ***Fill()*** method of DataAdapter class and pass object of Dataset to fill retrived records.

Ex: `da.Fill(ds);`

Here-
da= DataReader
ds= Dataset

When all above steps are complete then required SELECT sql command will be executed on active connection sucessfully and records will befillup into Dataset.

Data Bound Control and Data Grid (GridView Control):

Any control of ASP.NET that can be used to show records of any data source like MS-SQL Server called Data Bound Control. To Bind records on Data Bound control, ***DataSource*** property and ***DataBind()*** method is used of that control.

Data Grid or GridView Control is one most papular and important Data bound control. Using this control we can show records on table form. The Data source of GridView control is Dataset.

Let ***GridView1*** is one Databound control of **GridView** Control then to bind Records of Dataset we have to apply DataSource property and DataBind() method like.

```
    GridView1.DataSource = ds;
    GridView1.DataBind();
```

Here
ds= Dataset with Records.

Working process with DataBound Control (DataGrid / GridView)

First add GridView Control on web page using following asp.net code.

```
<asp: GridView ID= "GridView1" runat= "server" />
```

Now apply following steps on page load event of web page.

Step1 to Step7 *are same as DataAdpter and Dataset.*

Step8: Set Data source of GridView control that is Dataset.

Ex: `GridView1.DataSource=ds;`

Step9: Bind GridView control so that data can be shown on control.

Ex: `GridView1.DataBind();`

When all above steps are completed then data of Dataset can be bindup to the data bound control like GridView.

UNIT-IV

Writing datasets to XML
Reading datasets with XML
Web Application deployment
Web services: Introduction
Remote method call using XML
SOAP
Web service description language
Building & consuming a web service
.

Working with XML Data:

- XML(Extensible Markup Language) has the standard format to represent information on the web.
- XML files (or streams of data) are self-describing nature that is each value has a label.
- XML is case-sensitive.
- XML files can be created, read, and revised using ASP.NET 2.0.

Example: XML format to represent data:

Let we have following data about any book in tabular form as-

Bid	**BookName**	**Price**
1	ASP.NET	600
2	Java	400
3	AI	500

Now the XML format for above table can be represented as-

```
<Books>
     <Book>
          <Bid>1</Bid>
          <BookName>ASP.NET</BookName>
          <Price>600</Price>
     </Book>
     <Book>
          <Bid>2</Bid>
          <BookName>Java</BookName>
          <Price>400</Price>
     </Book>
     <Book>
          <Bid>3</Bid>
          <BookName>AI</BookName>
          <Price>500</Price>
     </Book>
</Books>
```

Here

`<Books>` Represent name of Database

`<Book>` Represent Each Row

`<Bid>`,`<BookName>` and `<Price>` Represent Field Names that repeats for each row with different values.

Writing Dataset to XML:

Dataset is the collection of tables in ASP.Net. After getting data into dataset we can write all data of dataset to the XML file in XML format. In ASP.NET to work with XML format first of all we need to include following name space in code behind-

`using System.Xml;`

This namespace contains all the necessary classes that capable to work with XML file.

If we want to write data of dataset to the XML format file then we call ***`WriteXml()`*** method of Dataset object.

```
Dataset ds=new Dataset();
ds.WriteXml(Server.MapPath("xmlfile.xml"));
```

Process of Writing Dataset content to XML:

1. First of all we read table data from database and fill to the Dataset
2. Write Dataset content to the XML file.

```
     using System.XML;
     using System.Data.OleDb;
     void btnWrite_Click(object sender, EventArgs e)
     {
//Step 1:
     //Establish connection string
     OleDbConnection con = new OleDbConnection
     ("provider=microsoft.jet.oledb.4.0; data source=
     |datadirectory|database.mdb");
     con.Open();
     //Getting data from database
     string sql="select * from student";
     OleDbDataAdapter da = new OleDbDataAdapter(sql, con);
     // Create new Dataset and fill
     DataSet ds = new DataSet();
     da.Fill(ds);
//Step 2:
     //Write Dataset content to XML file
     ds.WriteXml(Server.MapPath("xmlfile.xml"));
     Response.Write("Dataset contents has write to the XML");
     }
```

When above code will be implement on any event like button click and event occurs at run time then contents of Dataset will be save in XML form to the xmlfile.xml.This file store dataset content in following form:

```
<?xml version="1.0" standalone="yes"?>
<NewDataSet>
  <Table>
    <ID>1</ID>
    <sname>lokesh</sname>
    <age>35</age>
  </Table>
  <Table>
    <ID>2</ID>
    <sname>jahnavi</sname>
    <age>3</age>
  </Table>
</NewDataSet>
```

From above explanation it is clear that we can write dataset content to the XML file.

Reading Dataset with XML:

As we know Dataset is the collection of tables in ASP.Net and XML is special markup language to represent tabular data. In ASP.NET we can read XML format content and fill into dataset using ***ReadXml()*** method of Dataset object.

Process to Read XML content into dataset:

Step1: Create XML file as the name "xmlfile.xml" in the root directory of web project that has XML format data like-

```
<?xml version="1.0" standalone="yes"?>
<Books>
  <Book>
    <Bid>1</Bid>
    <BookName>ASP.NET</BookName>
    <Price>600</Price>
  </Book>
  <Book>
    <Bid>2</Bid>
    <BookName>Java</BookName>
    <Price>400</Price>
  </Book>
  <Book>
    <Bid>3</Bid>
    <BookName>AI</BookName>
    <Price>500</Price>
  </Book>
</Books>
```

Step2: Add data grid into web page and at any event like **page_load** we write following code to read XML file content into Dataset object and bind it to data grid control.

```
void Page_Load(object sender, EventArgs e)
{
 // create new dataset
     DataSet ds = new DataSet();
//read XML into dataset
     ds.ReadXml(Server.MapPath("xmlfile.xml"));
//show dataset content to data grid
     GridView1.DataSource = ds;
     GridView1.DataBind();
}
```

When web page executes then XML content via Dataset into data grid will be shown as

Bid	**BookName**	**Price**
1	ASP.NET	600
2	Java	400
3	AI	500

From above explanation it is clear that we can read the dataset with XML file content.

Web application Deployment:

It is required to publish (deploy) a Visual Studio web project to a server where others can access the application over the Internet.

It means Web application deployment is the process of installing web application on the customer's Host machine and making that web application available and accessible to all over the world.

Process to deployment of web application:

Step1: Before deployment of web application we must ensure that application contains everything that is necessary to run application. It may include-

- **HTML and CSS files:** Your design and structure.
- **ASPX files:** Your main pages.
- **ASPX.VB or ASPX.CS files:** The code-behind files.
- **Database files (.MDB or .MDF):** The back end of the site.
- **Image files (.JPG, .GIF, .PNG):**
- **XML files:** .XML and .XSD files. Etc

Step2: After then make sure that web application actually compiles and runs.

Main menu: Debug→Start debugging (F5).

Step3: Convert Web application only in executable mode by following-

Main menu: Select Build→Publish web site.

Step4: We get a window. Choose target location and press ok button.

Step5: Copy web application from target location to an application folder of remote hosting computer (server).

Web Services:

Basic Means: Web Services allow a *consumer* site (local) to obtain information from a *provider* site.

For example: Any local web site can display real-time data using web services provided by the Main site(www.Ford.com), but keep the user on the page of the local site.

Features of Web Services:

- ASP.NET 2.0 offers a complete web-services solution.
- Web services are a method of making information available that could be accessed by any developer's application over the Web.
- Web services can form a library of information that could be anything like a mathematical function calculator.
- A web service is not an local web application and does not rendered as web pages, nor as executable files (.exe); It is just like a user interface.
- The information contained in the web service is wrapped up as an XML document (in other words, plain text).
- Web services communicate using open protocols like SOAP.
- Web services are self-contained and self-describing mechanism.
- HTTP and XML is the basis for Web services.
- Web services can be published, found, and used on the Web.
- Web services use XML to code and to decode data, and SOAP to transport it (using open protocols).

Benefits:

- Web Developer can use easily web services and integrate them into web applications.
- Web services save the time of developer and effort by reducing code duplication.

Way of using web services:

They can be used in one of two ways.

1. You can create a web service that is exposed to the web, to share with other developers and other applications. Or
2. you can search for a web service that can be added to your own application. (They are similar to plug-ins in that respect.)

***Components of Web Services*:**

Everything to do with web services is standardized:

- the method of transmission

- the method used to wrap the web service up
- the way the web service is defined

All have clear W3C standards associated with the technologies involved. And all these standards are based on XML. So they're quick and easy to download, and even easier to use.

Web Services Description Language(WSDL):

Basic Mean: WSDL is a language for describing web services and how to access them.

General Features:

- WSDL is written in XML.
- WSDL became a W3C Recommendation 26. June 2007
- WSDL document is just a simple XML document.
- It contains set of definitions to describe a web service.

The WSDL Document Structure:

A WSDL document describes a web service using a number of elements. The main structure of a WSDL document looks like this:

```
< definitions>


< types>
data type definitions........
< /types>

< message>
definition of the data being communicated....
< /message>

< portType>
set of operations......
< /portType>

< binding>
protocol and data format specification....
< /binding>

< /definitions>
```

A WSDL document can also contain other elements, like extension elements, and a service element.

SOAP: (Simple Object Access Protocol)

Basic Concepts: SOAP provides a way to communicate between applications running on different operating systems, with different technologies and programming languages. It is important for application development to allow Internet communication between programs.

Need of SOAP: Today's applications communicate using Remote Procedure Calls (RPC) between objects like DCOM and CORBA, but HTTP was not designed for this. RPC represents a compatibility and security problem; firewalls and proxy servers will normally block this kind of traffic.

Solution: SOAP was created to accomplish this for better way to communicate between applications over HTTP, because HTTP is supported by all Internet browsers and servers.

General Features of SOAP :

- SOAP is a communication protocol
- SOAP is for communication between applications.
- SOAP is a format for sending messages|
- SOAP communicates via Internet.
- SOAP is platform independent.
- SOAP is language independent.
- SOAP is based on XML.
- SOAP is simple and extensible|
- SOAP is a W3C recommendation in 24. June 2003.

SOAP Building Blocks:

A SOAP message is an ordinary XML document containing the following elements:

Skeleton SOAP Message

```
< ?xml version="1.0"?>
< soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-
envelope" soap:encodingStyle= "http://www.w3.org/2001/12/
soap-encoding">

< soap:Header>
...
< /soap:Header>

< soap:Body>
...
```

```
<soap:Fault>
...
</soap:Fault>
< /soap:Body>

< /soap:Envelope>
```

Here-

- An Envelope element: that identifies the XML document as a SOAP message.
- A Header element: that contains header information.
- A Body element: that contains call and response information
- A Fault element: containing errors and status information

All the elements above are declared in the default namespace for the SOAP envelope.

UNIT-V

Overview of C#
C# and .NET
Similarities & differences from JAVA
Structure of C# program
Language features:
- Type system
- Flow control
- Boxing and unboxing
- Classes
- Interfaces

Serialization
Delegates
Reflection.

Overview of C#:

- C# is Microsoft's programming language for .NET development.
- C# was created at Microsoft late in the 1990s and It was first released in its alpha version in the middle of 2000. C#'s chief architect was Anders Hejlsberg. Different version of C# are 1.0, 1.1, 2.0,3.0.
- The Source code of C# converts into 16 bits MSIL code(Microsoft Intermediate Language) and executed by .NET Framework.
- C# is directly related to C, C++, and Java. From C, C# derives its syntax, many of its keywords, and its operators. **From C++,** C# builds upon and improves the object model.
- C# and Java both descended from C and C++ that shares the C/C++ syntax and object model.
- C# support properties, methods, and events.

C# v/s C, C++ and Java:

- C was invented by Dennis Ritchie in the 1972 based on the Procedure Oriented *programming. Using C,* large programs were difficult to write.
- C++ was invented by Bjarne Stroustrup beginning in 1979 based on Object oriented Model. Using C++, large programs were easy to handle.
- C and C++, always compiled machine dependent executable code and not support internet based programs.

C#	Java
C# was created at Microsoft late in the 1990s by Anders Hejlsberg.	Java was invented by James Gosling team in 1991 at Sun Microsystems. Initially called Oak.
C# is a structured, object-oriented language with a syntax and philosophy derived from C and C++.	Java is also a structured, object-oriented language with a syntax and philosophy derived from C and C++.
C# achieved portability by translating a program's source code into an intermediate language called *MSIL code(Microsoft Intermediated language).* This MSIL code was then executed by the .Net Framework.	Java achieved portability by translating a program's source code into an intermediate language called *byte code.* This byte code was then executed by the Java Virtual Machine (JVM).
A C# program could run only in an	A Java program could run in any

environment where MS.NET framework is available.	environment for which a JVM was available.
C# code is neither upwardly nor downwardly is compatible with C or C++, its syntax sufficiently similar.	Java code is also neither upwardly nor downwardly is compatible with C or C++, its syntax sufficiently similar.
C# has successfully portable in the Internet environment along with ASP.NET.	Java has also successfully portable in the Internet environment.
C# includes features that directly support the constituents of components, such as properties, methods, and events.	Java has also supported.
C#'s has ability to work in a secure, mixed-language Environment.	Java work only in one language environment.

C# and the .NET Framework

C# is a computer language that has a special relationship to its runtime environment, known as the .NET Framework. It has two reasons.

- First, C# was initially designed by Microsoft to create code for the .NET Framework.
- Second, the libraries used by C# are the ones defined by the .NET Framework.

Because of this, it is important to have the .NET Framework for C# programs.

Structure of C# program:

C# program has following structure-

```
// Namespace Declaration
using System;
```

```
// Program start class
class WelcomeCSS
{
 // Main begins program execution.
 static void Main()
 {
  // Write to console
  Console.WriteLine("www.LRsir.net");
 }
}
```

```
// Other user define class
```

Remark: C# is case-sensitive. The C# program has 4 primary elements:

1. A namespace declaration
2. A class
3. A *Main* method and
4. A program statement.

Compile C# Code: It can be compiled with the following command line:

```
csc.exe Welcome.cs
```

This produces a file named ***Welcome.exe***, which can then be executed.

Description:

1. *The namespace declaration:* Namespaces contain groups of code that can be called upon by C# programs.
2. *The class declaration:* It contains the data and method definitions. A class is one of a few different types of elements to describe objects, such as structs, interfaces , delegates, and enums.
3. *The method name:* Main, is reserved for the starting point of a program.

Type System: (Variables, Types and Operators)

Variables: are simply storage locations for data. You can place data into them and retrieve. Data in a variable is controlled through "Types". C# is a "Strongly Typed" language.

The C# data types: Variables are declared using following C# data types.

1. Boolean type (true / false): *bool*
2. Signed Integer type (+ /-): byte (1B), short / int16 (2B), int / int32 (4B), long / int64 (8B).
3. Unsigned Integer (+): ushort, uint, ulong.
4. Floating data type: float, single, double.
5. Character type: char
6. String type: string

Example: `int a;`

```
string name="LRsir.net"
```

The Array Type: Array can be thought of as a container that has a list of storage locations for a specified type. When declaring an Array, specify the type, name, dimensions, and size.

Example:

```
using System;
class Array
{
public static void Main()
{
string[] myStrings = new string[3];
myStrings[0] = "Joe";
myStrings[1] = "Matt";
myStrings[2] = "Robert";
...............
}
}
```

C# Operators:

Results are computed by combining variables and operators together into statements. The following table describes the allowable operators, their precedence and associativity.

Category (by precedence)	**Operator(s)**	**Associativity**
Primary	x++ x-- new typeof	left
Unary	+ - ! ~ ++x --x	right
Multiplicative	* / %	left
Additive	+ -	left
Shift	<< >>	left
Relational	< > <= >= is as	left
Equality	== !=	right
Logical AND	&	left
Logical XOR	^	left
Logical OR	|	left
Conditional AND	&&	left
Conditional OR	||	left
Null Coalescing	??	left
Ternary	?:	right
Assignment	= *= /= %= += -= <<= >>= &= ^= |= =>	right

Left associativity means that operations are evaluated from left to right. Right associativity mean all operations occur from right to left, such as assignment operators where everything to the right is evaluated before the result is placed into the variable on the left.

Flow Controls:

- The *if* statements.
- The *switch* statement with *break*.
- Loop: while, do, for, foreach

The *if* Statement: An *if* statement depends on a given condition. When the condition evaluates *true*, a block of code for that true condition will execute. You have the option to use optional *else* statement with *if* statement
Ex:

```
using System;
class UseIfElse
{
public static void Main()
{
   // Single Decision and Action with braces
   if (condition)
   {
     Logic Code1
   }
   else
   {
     Logic Code2
   }
}
}
```

When condition evaluates to *true*, the statement in the *if* block are executed; when *false*, the statements in the *else* block are executed.

The *switch* Statement: *switch* statement, executes a set of logic depending on the value of a given parameter.

Ex:

```
using System;
class SwitchSelect
{
public static void Main()
{
switch (Expression)
{
case 1:
             Code1;
             break;
case 2:
                Code2;
                break;
………
default:
                Optional code;
                break;
}
}
}
```

The *switch* block follows one or more choices. When the result of the *switch* expression matches one of these choices, statements of the matching choice are executed. After then jumps out from switch block using *break*. If none of the other choices match, then the *default* choice is taken and its statements are executed, although the use of *default* label is optional.

The *while* Loop: A *while* loop will check a condition and then continues to execute a block of code as long as the condition evaluates to a boolean value of *true*.

Syntax:

```
    while (Condition)
    {
        statements
    }
```

Once the statements have executed, control returns to the beginning of the *while* loop to check the boolean expression again.When the boolean expression evaluates to *false*, the *while* loop statements are skipped.

The *do* Loop: A *do* loop is similar to the *while* loop, except that it checks its condition at the end of the loop. This means that the *do* loop is guaranteed to execute at least one time. On the other hand, a *while* loop evaluates its boolean expression at the beginning.

Syntax:

```
    do
    {
        Statements

    }while (Condition);
```

The *for* Loop: A *for* loops are appropriate when you know exactly how many times you want to perform the statements within the loop.

Syntax: The contents within the *for* loop parentheses hold three sections separated by semicolons.

```
for (initializer ; condition ; update initialize)
{
        Statements
}
```
Like
```
for (int i=0; i < 20; i++)
{
Console.Write("{0} ", i);
}
```
Initializer: Evaluated only once during the lifetime of the *for* loop.

Condition: Once the initializer has been evaluated, the *for* loop gives control to condition. When the condition evaluates to *true*, the statements within the curly braces of the *for* loop are executed.

Update Initializer: After executing *for* loop statements, control moves to the top of loop and executes updater, after then control transfer to condition part.

The *foreach* Loop: A *foreach* loop is used to iterate through the items in a list. It operates on arrays or collections such as ArrayList, which can be found in the System.Collections namespace.

Syntax:

```
foreach (<type> <iteration variable> in <list>)
{
 <statements>
}
```

Introduction to Classes:

Classes are declared by using the keyword *class* followed by the *class* name and a set of *class* members surrounded by curly braces.

Every *class* has a constructor, which is called automatically any time an instance of a *class* is created. Constructors do not have return values and always have the same name as the *class*.

Example C# Classes: Classes.cs

```
// Namespace Declaration
using System;
class OutputClass
{
 string myString;
 // Constructor
 public OutputClass(string inputString)
 {
  myString = inputString;
 }

 // Instance Method
 public void printString()
{
 Console.WriteLine("{0}", myString);
 }

// Destructor
~OutputClass()
{
// Some resource cleanup routines
}
}

// Program start class
class ExampleClass
{
// Main begins program execution.
public static void Main()
{
// Instance of OutputClass
OutputClass outCl = new OutputClass("This is printed by the
output class.");
// Call Output class' method
outCl.printString();
}
}
```

HereOutputClass, has a constructor, instance method, and a destructor. It also had a field named *myString*.

Interfaces:

An *interface* looks like a class, but has no implementation. Interface contains only declarations of *events*, *methods* and *properties*. *Interfaces* only inherited by *classes*, which must provide an implementation for each interface member declared.

Defining an Interface:

```
interface MyInterface
{
   void MethodToImplement();
}
```

This method does not have an implementation because the *interface* only specifies the methods that must implement in class.

Using an Interface:

```
class UseInterface : MyInterface
{
 public void MethodToImplement()
 {
  Console.WriteLine("MethodToImplement() called.");
 }
 static void Main()
 {
  UseInterface oi = new UseInterface();
  oi.MethodToImplement();
 }
}
```

The UseInterface *class* implements the *MyInterface interface*. Indicating that a *class* inherits an *interface* is the same as inheriting a *class*.

Interfaces may also inherit other interfaces.

Boxing and Unboxing:

Boxing means the conversion of a value type (int / long / float) on the stack to a **object** type on the heap.

Unboxing means the conversion from an **object** type back to a value type.

Boxing occurs automatically whereas u*nboxing* using an explicit cast from the **object** reference to its corresponding value type.

```
// A simple boxing/unboxing example.

using System;
class BoxingUnboxing
 {
     static void Main()
     {
          int x=10;
          object obj=x;   // Boxing
```

```
            Console.WriteLine(obj);  // obj=10
            x = (int)obj; // Unboxing
            Console.WriteLine(x);    //x=10
        }
}
```

Value in **x** is boxed simply by assigning it to **obj**, whereas integer value in **obj** is retrieved by casting **obj** to **int**.

Delegates:

A delegate is an object that can reference a method just like a function pointer used in C or C++. Therefore, when you create a delegate, you are creating an object that can hold a reference to a method. Furthermore, the method can be called through this reference.
In other words, a delegate can invoke the method to which it refers.

Creating Delegate:

```
delegate ret-type delegateName(parameter-list);
```

```
//delegate is a keyword
```

Example: `delegate int Dx(int,int);`

Here Dx is such delegate that can hold reference of any method whose return type is int and has two arguments of int and int type.

Holding method reference to Delegates:

```
    delegateName objectName=new
    delegateName(Clsobject.MethodName);
```

Example: `Dx  odx=new Dx(oa.getdata);`

Here odx is an object of Dx delegate type that hold reference of a method of any object oa of class A.

Calling Method using delegate:

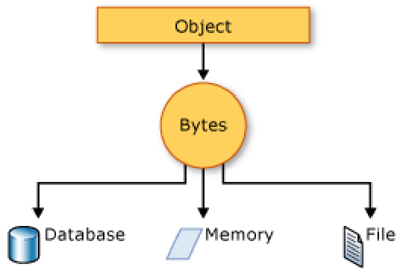
```
        delegateObject(list of  arguments);
```

Example: int x=odx(10,20);

Here odx(10,20) is similar to getdata(10,20);

Serialization

Definition: Serialization is the process of converting an object into a stream of bytes in order to store the object or transmit it to memory, a database, or a file. The reverse process is called deserialization.



The object is serialized to a stream, which carries not just the data, but information about the object's type, such as its version, culture, and assembly name. From that stream, it can be stored in a database, a file, or memory.

Uses: Its main purpose is to save the state of an object in order to be able to recreate it when needed. Through serialization, a developer can perform actions like sending the object to a remote application by means of a Web Service, passing an object from one domain to another, etc.

Making an Object Serializable : To serialize an object, we need an object to be serialized, a stream to contain the serialized object, and *System.Runtime.Serialization* namespace contains the classes necessary for serializing and deserializing objects.

Reflection:

Reflection is the feature that enables you to obtain information about a type. Using this information, you can construct and use objects at runtime. This feature is very powerful because it lets a program add functionality dynamically, during execution.

System.Reflection namespace must be included before using classes of reflection.

Ex:
```
// Analyze methods using reflection.
using System;
using System.Reflection;
class MyClass
{
     Class-members
}
```

```
class ReflectDemo
{
     static void Main()
     {
     MyClass obj=New MyClass();
     Type t = typeof(obj); // get a Type of  object obj
     Console.WriteLine("Class Type of obj is:" + t.Name);
     }
}
```

Output: Class Type of obj is: MyClass

Here **typeof** returns a **Type** object that represents the specified type, which in this case is **MyClass**.