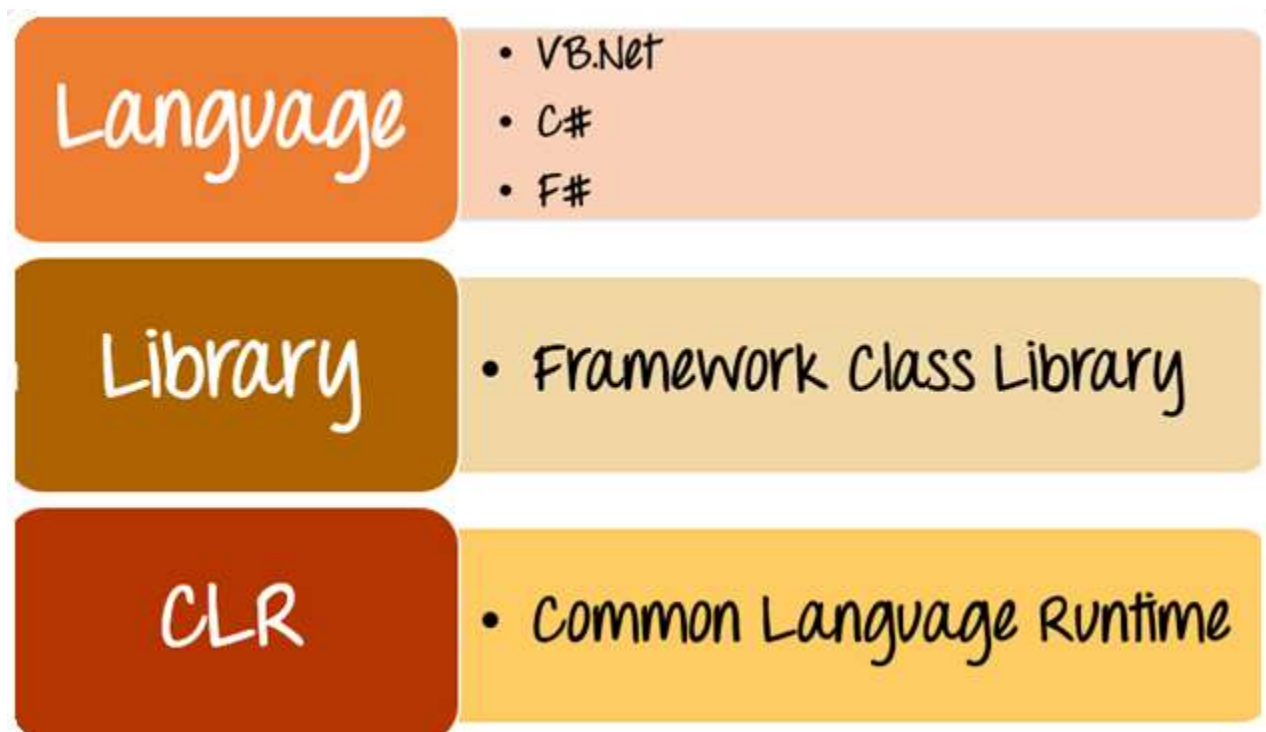


ASP.NET

ASP.NET is an open-source server-side web framework. It is used to build great websites and web applications using HTML, CSS, and JavaScript. We can also create Web APIs and use real-time technologies like Web Sockets. It was developed by Microsoft. It allows programmers to build dynamic web sites, web applications and web services. It works HTTP protocol. It uses the HTTP commands and policies to set a browser-to-server communication and cooperation.

We can use full featured programming language such as C# or VB.NET to build web applications easily. ASP.Net builds interactive, data-driven web applications over the internet. It includes a large number of controls like text boxes, buttons, and labels for assembling, configuring, and manipulating code to design and develop HTML Web pages.

ASP.Net Architecture



ASP.Net framework architecture is based on three components: The Language, The Library, and The CLR.

The Language: Dot net framework support different languages to write codes. We can use C# or VB.net to develop web applications.

The Library: Dot net framework consists of a set of class libraries. System.Web is the most common class library used in asp.net web application development

The CLR: Common Language Runtime (CLR) is the main part of dot net architecture. It performs all main activities of the framework. It performs activities like exception handling and garbage collection.

ASP.Net Project

An ASP.NET application includes several items: the web content files (.aspx), source files (.cs files), assemblies (.dll and .exe files), data source files (.mdb files), references, icons, user controls and miscellaneous other files and folders. All these files that build up the website are contained in a Solution. The contents of a project are compiled into an assembly as an executable file (.exe) or a dynamic link library (.dll) file.

ASP.Net Characteristics

Code Behind File: This concept separates design and coding. Because of this separation ASP.Net application is easy to maintain. The general file type of an ASP.Net file is .aspx. Let us assume that we have a web page called MyWebPage.aspx. There will be another file called MyWebPage.aspx.cs which denotes the code part of the page. Visual Studio IDE creates separate files for each web page, one for the design and another for respective code.

State Management: It provides the facility to control state management. HTTP is known as stateless protocol. Let us take an example of a shopping cart application. Here, after deciding the purchase items when a user finally wants to buy from the site, he will press the submit button.

The application needs to remember all the items the user selected for the purchase. This is known as remembering the state of an application at specific point in time. HTTP is a stateless protocol. When the user goes to the purchase page, HTTP will not store the information on the cart items. Additional coding needs to be done to ensure that the cart items can be carried forward to the purchase page. Such an implementation can become complex at times. But here, it can do state management on our behalf. So that ASP.Net can remember the cart items and passes it over to the purchase page.

Caching: It can implement the concept of Caching. Caching improves the performance of the application. With the help of caching frequently requested pages by the user can be stored in a temporary location. These pages can be retrieved faster and fast responses can be sent to the user.

ASP.Net Page Life Cycle

An ASP.NET page goes through a life cycle and performs a series of processing steps. These are: initialization, instantiating controls, restoring and maintaining state, running event handler code, and rendering. To understand the page life cycle is essential so that we can write appropriate code at the appropriate life-cycle stage for the effect we intend.

	Page methods	Page events	Control methods and events
Start	Construct ◇		<div> <div>◇ Method name</div> <div>† Event name</div> <div>■ Postback only</div> </div>
	ProcessRequest ◇		
	InitializeCulture ◇		
	DeterminePostBackMode ◇		
	OnPreInit ◇	PreInit †	
	OnInit ◇	Init †	
	TrackViewState ◇		
	OnInitComplete ◇	InitComplete †	
Load	LoadPageStateFromPersistenceMedium ◇		
	LoadViewState ◇		LoadViewState ◇
	ProcessPostData ◇		IPostBackDataHandler. LoadPostData ◇
	OnPreLoad ◇	PreLoad †	
	OnLoad ◇	Load †	
			Load †
Event handling	RaisePostBackEvent ◇		Control-changed events
Validation	Validate ◇		
PreRendering	OnLoadComplete ◇	LoadComplete †	
	OnPreRender ◇	PreRender †	PreRender †
			Data binding events
	OnPreRenderComplete ◇	PreRenderComplete †	
	SaveViewState ◇		SaveViewState ◇
	SavePageStateToPersistenceMedium ◇		
Rendering	OnSaveStateComplete ◇	SaveStateComplete †	
	RenderControl ◇		
	Render ◇		Render ◇
	RenderChildren ◇		
Unload			Unload †
	OnUnload ◇		
	Dispose ◇		

Page request

The page request step comes before the page life cycle begins. When the user request for a page, ASP.NET decides whether the page needs to be parsed and compiled (therefore beginning the life of a page), or just a cached version of the page can be sent in response without running the page.

Start

Page properties such as Request and Response are set in start stage. In this step, the page also decides whether the request is a postback or a new request and sets the IsPostBack and UICulture properties.

Initialization

At initialization stage all page controls are available and UniqueID property of each control is set. Master page and themes are also applied to the page, if applicable. If the current request is a postback, the postback data has not yet been loaded and control property values have not been restored to the values from view state.

Load

At the time of page load, if the current request is a postback, control properties are loaded with information recovered from view state and control state.

Postback event handling

Control event handlers are called only if the request is a postback. Then the Validate method of all validator controls is called, which sets the IsValid property of individual validator controls and of the page. (NOTE: The handler for the event that caused validation is called after validation.)

Rendering

View state is saved for the page and all controls before rendering. During the stage of rendering the page calls the Render method for each control, providing a text writer that writes its output to the OutputStream object of the Response property of the page.

Unload

The Unload event occurs after the page has been fully rendered, sent to the client, and is ready to be disposed. At this moment, Response and Request etc. properties of the page are unloaded and cleanup is performed.

ASP.Net Page Life Cycle Events

At each stage of the ASP.Net Page life cycle, the page raises some events that we can handle to run our own code. For control events, we bind the event handler to the event either by declaration using attributes such as **onclick** or by **writing code**.

Pages also support automatic event wire-up where ASP.NET looks for methods with particular names and automatically runs those methods when certain events are raised. For example, if the AutoEventWireup attribute of the @Page directive is set to true, page events are automatically bound to methods that use the naming convention of Page_event, such as Page_Load and Page_Init.

PreInit: It checks the IsPostBack property and determines whether the page is a postback. It sets the themes and master pages, creates dynamic controls, and gets and sets profile property values. This event can be handled by overloading the OnPreInit method or creating a Page_PreInit handler.

Init: It initializes the control property and build the control tree. This event can be handled by overloading the OnInit method or creating a Page_Init handler.

InitComplete: This event allows tracking of view state. All the controls turn on view-state tracking.

LoadViewState: This event helps in loading of view state information into the controls.

LoadPostData: The contents of all the input fields are defined with the <form> tag are processed.

PreLoad: This event raises before the post back data is loaded in the controls. This event can be handled by overloading the OnPreLoad method or creating a Page_PreLoad handler.

Load: This event is raised for the page first and then recursively for all child controls. The controls in the control tree are created. This event can be handled by overloading the OnLoad method or creating a Page_Load handler.

LoadComplete: When the loading process is completed, control event handlers run, and page validation takes place. This event can be handled by overloading the OnLoadComplete method or creating a Page_LoadComplete handler.

PreRender: This event occurs just before the output is rendered. Pages and controls can perform any updates before the output is rendered by using it.

PreRenderComplete: As the PreRender event is recursively fired for all child controls, this event ensures the completion of the pre-rendering phase.

SaveStateComplete: control state on the page is saved. Personalization, control state and view state information is saved. The HTML markup is generated. This stage can be handled by overriding the Render method or creating a Page_Render handler.

UnLoad: UnLoad is the last phase of the page life cycle. It raises the UnLoad event for all controls recursively and lastly for the page itself. Final cleanup is done and all resources and

references such as database connections are freed. This event can be handled by modifying the OnUnLoad method or creating a Page_UnLoad handler.

ASP.Net Web Development

ASP.NET offers three frameworks for creating web applications: **Web Forms**, **ASP.NET MVC**, and **ASP.NET Web Pages**. Each framework targets a different development style. The one we choose depends on a combination of our programming assets (knowledge, skills, and development experience), the type of application we're creating and the development approach we're comfortable with.

Web Forms

ASP.NET Web Forms allows us to build dynamic websites using a familiar drag-and-drop, event-driven model. It provides a design surface and hundreds of controls and components so that we can rapidly build sophisticated, powerful UI-driven sites with data access.

ASP.Net MVC

ASP.NET MVC provides a powerful way to build dynamic websites on the basis of patterns that enables a clean separation of concerns. It gives us full control over markup for agile development with ease. MVC includes so many features that enable fast, TDD-friendly development for creating sophisticated applications based on latest web standards.

ASP.Net Web Pages

ASP.NET Web Pages and the Razor syntax provide a fast, approachable, and lightweight medium to bind server code with HTML to create dynamic web content. Database connection, embedding video, link to social networking sites, and many more features are included that helps a lot to create beautiful sites that conform to the latest web standards.

Why ASP.Net?

Less Coding: ASP.NET requires less coding for application development.

JIT: Just-in-time compilation, early binding and caching services are available.

Library: a large library of built-in classes is available to use. This significantly reduces the amount of coding required.

CLR: supports common language runtime. It can support several dot net languages such as C#, VB, etc.

Fast: Execution is fast. The ASP.net application is located in the server in compiled form. This reduces response time.

Secure: Security features such as Windows authentication and form authentication are present.

Easy Database Integration: easy to integrate ASP.NET with ADO.NET. This streamlines database functionalities for websites.

Code-behind file: ASP.NET separates presentation and programming logic with the use of code-behind classes.

Web Application Using ASP.NET

ASP.NET is a free web framework for building great websites and web applications using HTML, CSS, and JavaScript. We can also create Web APIs and use real-time technologies like Web Sockets.

Websites and web applications

ASP.NET offers three frameworks for creating web applications: Web Forms, ASP.NET MVC, and ASP.NET Web Pages. All three frameworks are stable and mature, and we can create great

web applications with any of them. No matter what framework we choose, we will get all the benefits and features of ASP.NET everywhere.

Each framework targets a different development style. The one we choose depends on a combination of our programming assets (knowledge, skills, and development experience), the type of application we're creating and the development approach we're comfortable with.

Working with controls

Label control

Label control and the Literal control are used to display text in a page. Literal control only supports text property, but the Label control has a number of formatting properties. By default, a Label control renders its content in an HTML tag. After executing our application, we can see the respective label tag by using view source on the browser.

The Label control displays text at a particular position on a Web page. Generally Label control is used as the caption of a TextBox.

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

Literal control

It is light weight control. The Literal Control is similar to the Label Control but it does not support properties like BackColor, ForeColor, BorderColor, BorderStyle, BorderWidth, etc. Also we cannot apply a style to a literal control. It is used to display static text on a Web page without adding additional properties or HTML tags.

```
<asp:Literal ID="Literal1" runat="server">It is a Literal Control</asp:Literal>
```

TextBox control

The TextBox control is frequently used and one of the most important control. It is used to collect information from a user. It is an input control which is used to input the data.

The TextBox control contains an important property called TextMode. By using this property we can set textbox as

- SingleLine
- MultiLine
- Password

SingleLine mode is default mode of textbox control and allows the user to enter data in a single line of text.

Password mode masks (text is hidden) the values entered by the user.

MultiLine mode enables user to enter text in more than one line. We can use MultiLine mode, in combination with the Columns and Rows properties. Column specifies the number of columns (Width) and rows specify the number of rows (Height) to display.

Another important property of textbox control is MaxLength. It sets the limit on the number of character that a user can enter into textbox.

TextBox control supports the following event:

TextChanged: It fires, when we change the content or text of the textbox control. TextChanged event only fire, when AutoPostBack property has the value True. By default AutoPostBack property has the value false.

If we change the text of the TextBox control and press tab key from the keyboard, the form is automatically posted back to the server.

Button Controls

There are three types of button control available in ASP.NET:

- Button
- Link Button
- Image Button

Button control postbacks the web page to webserver, when user clicks the button. A Button control can be used as a submit button (default) or a command button.

We can also use Button control as a command button. It is useful when we want to group a set of buttons. Button control has property called CommandName. Assign unique CommandName value for each button. Create a single command event handler explicitly, that will handle the event of all command buttons.

Suppose that we have four buttons as given below and want to create all buttons as a command button.



Provide a unique CommandName to each button. As example First, Last, Previous, Next.

Example: Default.aspx

```
<% @ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head runat="server">
    <title></title>
  </head>
  <body>
    <form id="form1" runat="server">
      <div>
        <asp:Button ID="Button1" runat="server" OnCommand="Navigate_Command"
Text="<<" CommandName="First" Width="65px"/>
        <asp:Button ID="Button2" runat="server" OnCommand="Navigate_Command"
Text="<" CommandName="Previous" Width="65px"/>
        <asp:Button ID="Button3" runat="server" OnCommand="Navigate_Command"
Text=">" CommandName="Next" Width="65px"/>
```

```
<asp:Button ID="Button4" runat="server" OnCommand="Navigate_Command"
Text=">>" CommandName="Last" Width="65px"/>
</form>
</body>
</html>
```

Default.aspx.cs

```
using System;
using System.Web.UI.WebControls;
using System.Drawing;
public partial class _Default : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }
    protected void Navigate_Command(object sender, CommandEventArgs e)
    {
        switch (e.CommandName)
        {
            case "First":
                Response.Write("First");
                break;
            case "Previous":
                Response.Write("Previous");
                break;
            case "Next":
                Response.Write("Next");
                break;
            case "Last":
                Response.Write("Last");
```

```
        break;
    }
}
}
```

CausesValidation property: It checks the page validation. By default this property is true for button control. If we set CauseValidation value to false, then it will bypass the page validation.

LinkButton

LinkButton control displays a link instead of a push button. By default, a LinkButton control is a Submit button.

Some of the important properties of LinkButton Control are:

CausesValidation: If this property is set as true then validation will be performed when Linkbutton is be clicked. Otherwise it will bypass the validation.

PostBackUrl: It posts a form to a particular page when the LinkButton control is clicked.

ValidationGroup: The group of controls when the LinkButton control causes posts back to the server.

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

OnClientClick: We can attach a client-side script that executes when the LinkButton is clicked.

ImageButton

The ImageButton control is similar to Button and LinkButton controls but it always displays an image. It works as a clickable image. Most of the properties are same as Button or LinkButton. The main difference is ImageUrl and AlternateText property. ImageUrl Gets or Sets the location of the image. AlternateText property provides alternate text for the image.

Let us take an example that will show the use of TextBox and Different type of Button.

Example: Default.aspx

```
<% @ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
    <style type="text/css">
        .auto-style1
        {
            width: 103px;
        }
        .auto-style3
        {
            width: 165px;
        }
        .auto-style4
        {
            width: 143px;
        }
    </style>
</head>
<body>
    <form id="form1" runat="server">
        <table style="width: 76%;">
            <tr>
                <td class="auto-style1">
                    <asp:Label ID="Label1" runat="server" Text="Label">EmployeeName</asp:Label>
                </td>
```

```
<td class="auto-style4">
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
</td>
<td class="auto-style3"> </td>
</tr>
<tr>
<td class="auto-style1">
<asp:Label ID="Label2" runat="server" Text="Label">Password</asp:Label>
</td>
<td class="auto-style4">
<asp:TextBox ID="TextBox2" runat="server" TextMode="Password"> </asp:TextBox>
</td>
<td class="auto-style3"> </td>
</tr>
<tr>
<td class="auto-style1">
<asp:Label ID="Label3" runat="server" Text="Label">Address</asp:Label>
</td>
<td class="auto-style4">
<asp:TextBox ID="TextBox3" runat="server" Height="54px" TextMode="MultiLine"
Width="211px"></asp:TextBox>
</td>
<td class="auto-style3"> </td>
</tr>
<tr>
<td class="auto-style1">
<asp:Button ID="Button1" runat="server" Text="Button" />
</td>
<td class="auto-style4">
<asp:LinkButton ID="LinkButton1" runat="server">Submit</asp:LinkButton>
</td>
```



```

        <td class="auto-style3">
            <asp:ImageButton ID="ImageButton1" runat="server" AlternateText="This is Image
            Button" ImageUrl="~/logo.gif" BorderColor="Blue" BorderStyle="Solid" BorderWidth="2px"
            PostBackUrl="~/Default2.aspx" />
        </td>
    </tr>
</table>
</form>
</body>
</html>

```

Employee Name

Password

Address

Button

[Submit](#)

CareerRide.com

Default Textbox

Password Textbox

Multiline Textbox

LinkButton

Image button

CheckBox Control

A CheckBox control is used to select a single or multiple options from the available choices. For example a person can select more than one city for travelling.

Important Properties of the CheckBox Control

Checked: It is used to check if the check box is checked or not. This is a boolean property.

Text: It is used to get or set the text associated with the check box control. This is a string property.

TextAlign: It enables us to set the text right or left of the check box. By default TextAlign property is set to right.

AutoPostBack: If we want that when we change the status of checkbox (check or uncheck), then set AutoPostBack property true. By default this property is set to false.

Methods:

Focus(): It sets the input focus, to a specific checkbox. Call this method for that check box control.

Events:

CheckedChanged: This event is fired when the checked status of the checkbox control is changed. AutoPostBack property should be set as true to fire this event.

RadioButton Control

Radio Button control is used, when we want to select only one option from the available choices. RadioButton control works in a group. There may be more than one group of radio button. We can select only one RadioButton control from particular radio button group.

Gender

☒ Male

☐ Female

Country

☒ India

☐ USA

☐ Australia

Most of the properties and events are same as CheckBox control but GroupName property is different.

When we use RadioButton, they are not in a group. We can select more than one RadioButton, they are not mutually exclusive. We have to form the group of radio button as above given example. We can create group by assigning similar GroupName to RadioButton. Let us take one example and we will use all the controls that we have discussed till now. In this example, when user fill all the details and click on submit button, the information will come in last multiline textbox. When user clicks on cancel button, all fields will be empty.

Example

```
using System;

public partial class EmployeeForm : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
    }

    protected void btnSubmit_Click(object sender, EventArgs e)
    {
        string Name = "Name: " + txtName.Text;
```

```
string Email = "Email: " + txtEmailID.Text;
string Gender = "" ;
if(RDBmale.Checked)
{
    Gender = "Gender: " + RDBmale.Text;
}
if (RDBfemale.Checked)
{
    Gender = "Gender: " + RDBfemale.Text;
}
string Cities = "";
if(chbAgra.Checked)
{
    Cities += chbAgra.Text + "\n";
}
if (chbDelhi.Checked)
{
    Cities += chbDelhi.Text + "\n";
}
if (chbPune.Checked)
{
    Cities += chbPune.Text + "\n";
}
string Address = "Address:" + txtAddress.Text;
txtResult.Text = Name + "\n" + Email + "\n" + Gender + "\n" + "City: " + Cities + "\n" + Address;
}

protected void btnCancel_Click(object sender, EventArgs e)
{
    txtName.Text = String.Empty;
    txtPassword.Text = String.Empty;
    txtAddress.Text = String.Empty;
}
```

```
txtEmailID.Text = String.Empty;  
RDBfemale.Checked = false;  
RDBmale.Checked = false;  
chbPune.Checked = false;  
chbAgra.Checked = false;  
chbDelhi.Checked = false;  
txtResult.Text= String.Empty;  
}  
}
```

Output:

Name	<input type="text" value="Raj Singh"/>
Password	<input type="password" value="....."/>
EmailID	<input type="text" value="raj@mymail.com"/>
Gender	<input checked="" type="radio"/> Male <input type="radio"/> Female
Preferred Cities	<input checked="" type="checkbox"/> Delhi <input type="checkbox"/> Agra <input type="checkbox"/> Pune
Current Address	<input type="text" value="Civil Lies, Pune"/>
<input type="button" value="Submit"/> <input type="button" value="Cancel"/>	

Name: Raj Singh
Email: raj@mymail.com
Gender: Male
City: Delhi

Address: Civil Lies, Pune

ASP.NET provides the following list controls.

- Drop-down list
- List box
- Radio button list
- Check box list
- Bulleted list

These controls display list of options to select. We can select one or more options, the choice depends upon control. They all derive from the `System.Web.UI.WebControls.ListControl` class

Some of the important common properties of list controls are as follows:

- `SelectedValue`: Get the value of the selected item from the dropdown list.
- `SelectedIndex`: Gets the index of the selected item from the dropdown box.
- `SelectedItem`: Gets the text of selected item from the list.
- `Items`: Gets the collection of items from the dropdown list.
- `DataTextField`: Name of the data source field to supply the text of the items. Generally this field came from the datasource.
- `DataValueField`: Name of the data source field to supply the value of the items. This is not visible field to list controls, but we can use it in the code.
- `DataSourceID`: ID of the datasource control to provide data.

There are several ways through which we can populate these controls such as:

- By using data from database.
- Directly write code to add items.
- Add items through the items collection from property window.
- Write HTML to populate these controls.
- Use inbuilt datasource controls.

DropDownList control

DropDownList control is used select single option from multiple listed items.

Example

```
using System;
using System.Collections.Generic;
public partial class ListControls : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (! IsPostBack)
        {
            List<string> cityList = new List<string>();
            cityList.Add("Pune");
            cityList.Add("Kanpur");
            cityList.Add("Jaipur");
            cityList.Add("Delhi");
            cityList.Add("Agra");
            DropDownList1.DataSource = cityList;
            DropDownList1.DataBind();
        }
    }
    protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
    {
        Label1.Text = DropDownList1.SelectedItem.Text;
    }
}
```

Select the item from the DropDownList, the label control will display the selected item. Please keep in mind that, we should write code with association with IsPostBack property otherwise we will get the first item of the DropDownList.

We can also add items directly to write HTML as follows:

```
<asp:DropDownList ID="DropDownList2" runat="server">
    <asp:ListItem Value="1">India</asp:ListItem>
    <asp:ListItem Value="2">USA</asp:ListItem>
    <asp:ListItem Value="2">Australia</asp:ListItem>
    <asp:ListItem Value="4">Canada</asp:ListItem>
    <asp:ListItem Value="5">Newzealand</asp:ListItem>
</asp:DropDownList>
```

ListBox control

The ListBox control is similar to the DropDownList but main difference is that we can select multiple items from ListBox at a time. ListBox control has SelectionMode property that enables we to select multiple items from ListBox control. By default SelectionMode property is set as single. If we want to select multiple items from the ListBox, then set SelectionMode property value as Multiple and press Ctrl or Shift key when clicking more than one list item.

The ListBox control also supports data binding. We can bind the ListBox through coding with database or attach it with one of the predefined DataSourceControl objects, which contains the items to display in the control. DataTextField and DataValueField properties are used to bind to the Text and Value field in the data source.

Example

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Web.UI.WebControls;

public partial class ListControls : System.Web.UI.Page
{
    List<string> empList;

    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            empList = new List<string>();
            empList.Add("Raj");
            empList.Add("Rajesh");
            empList.Add("John");
            empList.Add("Elina");
            empList.Add("Samy");
            empList.Add("Reena");
            ListBox1.DataSource = empList;
            ListBox1.DataBind();
        }
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        StringBuilder sb = new StringBuilder();
        foreach(ListItem item in ListBox1.Items)
        {
            if(item.Selected)
            {
```

```

        sb.Append(item + "<br>");
    }
}
Label1.Text = sb.ToString();
}
}

```

Execute the above program and select the items from ListBox. Click on show button, we will get the selected items.

The ListBox control has a `SelectedIndexChanged` event handler. If `AutoPostBack` property is set to true, then this event is raised whenever we select new item from the List control.

3w



Common Properties for All Controls

Most of the server controls we find in the VWD Toolbox share some common behavior. For example, each control has an ID to uniquely identify it in the page, a `Runat` attribute that is always set to `Server` to indicate the control should be processed on the server, and a `ClientID` that contains the client-side ID attribute that will be assigned to the element in the final HTML. The

Runat attribute does not really belong to the server control, but is necessary to indicate that the markup for the control should be processed as a server control and not end up as plain text or HTML in the browser.

Besides these properties, many of the server controls share more properties. The following table lists the most common ones and describes what they are used for.

Property	Description
AccessKey	Allows we to set a key with which a control can be accessed in the client by pressing the associated letter.
BackColorForeColor	Allows we to change the color of the background (BackColor) and text (ForeColor) of the control in the browser.
BorderColorBorderStyleBorderWidth	Changes the border of the control in the browser. The similarities with the CSS border properties we saw in the previous chapter are no coincidence. Each of these three ASP.NET properties maps directly to its CSS counterpart.
CssClass	Lets we define the HTML class attribute for the control in the browser.
Enabled	Determines whether the user can interact with the control in the browser. For example, with a disabled text box (Enabled="false") we cannot change its text.
Font	Allows we to define different font-related settings, such as Font-Size, Font-Names, and Font-Bold.
HeightWidth	Determines the height and width of the control in the browser.
TabIndex	Sets the HTML tabindex attribute that determines the order in which users can move through the controls in the page by pressing the Tab key.

ToolTip	Allows we to set a tooltip for the control in the browser. This tooltip, rendered as a title attribute in the HTML is shown when the user hovers the mouse over the relevant HTML element.
Visible	Determines whether the control is sent to the browser or not. We should really see this as a server-side visibility setting because an invisible control is never sent to the browser at all. This means it's quite different from the CSS display and visibility properties we saw in the previous chapter that hide the element at the client.

Page navigation in ASP.NET

Page navigation is the process of moving from one page to another page in our website. There are many ways to navigate from one page to another in ASP.NET.

- Client-side navigation
- Cross-page posting
- Client-side browser redirect
- Server-side transfer

Client-side navigation

Client-side navigation means that, our browser (client) or HTML request a web page by clicking on HyperLink. HyperLink control is the easiest way to navigate to another web page. It creates a link to another Web page. The HyperLink control renders an HTML anchor tag, <a>. The Text property is used for displaying the text in the HyperLink. We can also display an image on this control instead of text. To display an image we should set the ImageUrl property.

```
<asp:HyperLink ID="HyperLink1" runat="server" NavigateUrl="~/Welcome.aspx">
    Goto Welcome Page
</asp:HyperLink>
```

When we see the view source on browser, we will get the rendered HTML as given below:

```
<a id="HyperLink1" href="Welcome.aspx">Goto Welcome Page</a>
```

When a user clicks on HyperLink control, browser called Welcome.aspx web page. We can also use JavaScript for client side navigation. We should use HTML input button for this. The document object represents the Web page in client-side JavaScript. We can call a particular web page, by providing the name of aspx page to its Document location property.

Let us take an example, in which we will take an HTML button. onclick event of HTML button called the client-side method name Button1_onclick.

Example

```
<input id="Button1" type="button" value="Goto Welcome Page"
    onclick="return Button1_onclick()" />
```

The JavaScript code for the Button1_onclick method is written into the <head> section of the aspx page as follows:

```
<script language="javascript" type="text/javascript">
    function Button1_onclick()
    {
        document.location = "Welcome.aspx";
    }
</script>
```

When user clicks the button, Welcome.aspx page is called but no data is posted back to server.

Cross-page posting

By default, buttons and other controls that cause a postback on an ASP.NET Web page submit the page back to itself. ASP.Net provides a feature known as Cross PagePostBack that enables a web form to post-back to a different web form. A Button control supports PostBackUrl property that is used to set a Web page to which the processing occurs. The Page class has a property named PreviousPage that is used for reference of previous page. The previous page's data is available in the Page.PreviousPage property.

Example:

Let us take an example in which there are two web pages named as Home.aspx and Welcome.aspx respectively. The first page contains a Button and a Textbox control. Set the PostBackUrl property of Button control as “~/Welcome.aspx”

// Sample code for Welcome.aspx.

```
Protected void Page_Load(object sender, EventArgs e)
{
    if (Page.PreviousPage != null)
    {
        TextBox MyTextBox = (TextBox)Page.PreviousPage.FindControl("TextBox1");
        if (MyTextBox != null)
        {
            Label1.Text = MyTextBox.Text;
        }
    }
}
```

Client-side browser redirect

The Page.Response object has redirect method. The Response.Redirect method redirects a request to a new URL. It instructs the browser to initiate a request for another Web page. The redirect is not a PostBack. It is similar to the user clicking a hyperlink on a Web page. The URL

in the address bar of browser is updated according to parameter passed in Redirect method. The PreviousPage property does work when using the Redirect method. Response.Redirect takes an extra round trip to the server.

```
protected void Button1_Click(object sender, EventArgs e)
{
    Response.Redirect("TutorialRide.aspx");
}
```

Server-side transfer

The Transfer method transfers all the state information in one ASP.NET Page to a second ASP.NET Page. Server.Transfer changes the page being rendered on the browser. This happens all on the server. A redirect is not issued to the web browser. The URL of address bar of browser is not changed in this process. Server.Transfer reduces server requests and keeps the URL the same. It avoids the round trip to server.

```
protected void Button1_Click(object sender, EventArgs e)
{
    Server.Transfer("TutorialRide.aspx");
}
```

Difference between Response.Redirect and Server.Transfer

Both Response.Redirect and Server.Transfer methods are used to navigate a user from one web page to another web page. But there are some differences between both the methods as follow.

Response.Redirect()	Server.Transfer()
It sends we to a new page and update the address bar	It does not update the address bar while transfer to new page

On browser we can click back.	Back button of browser does not work
It takes additional round trips to the server on each request.	It avoids the additional round trips to the server.
We can pass the address of any web page as a parameter in Redirect method	The requested web page should be on the same server.

Data Access with List Controls in ASP.NET

List controls displays simple lists of options.

Following are the important List Controls in ASP.NET.

- DropDownList
- RadioButtonList
- ListBox
- CheckBoxList
- BulletedList controls

These classes are inherited from ListControl class. ListControl class is an abstract class that provides the common properties, methods, and events for all above five list-type controls.

Working with the DropDownList Control

DropDownList control displays list of options. We can select only one item at a time from DropDownList. SelectedIndexChanged is the default event of DropDownList. By default AutoPostBack property is false for this control. If we want the value of selected item from DropDownList, then AutoPostBack property must be set as true. In case of AutoPostBack

property true, whenever user selects the item from DropDownList, the form is automatically posted back to the server.

Consider the following table structure. We will use this table in successive example. The table name is tblCity.

	Column Name	Data Type	Allow Nulls
	CityID	int	<input type="checkbox"/>
	CityName	nvarchar(50)	<input type="checkbox"/>
<input type="checkbox"/>			<input type="checkbox"/>

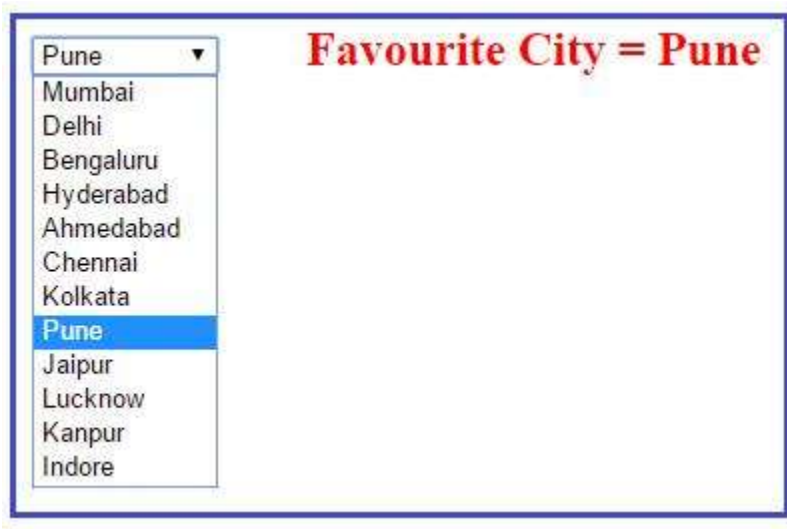
Example

```
using System;
using System.Data;
using System.Data.SqlClient;
using System.Configuration;
using System.Web.UI.WebControls;
using System.Web.UI;

public partial class ListControlsDemo : System.Web.UI.Page
{
    SqlConnection conn;
    SqlDataAdapter adapter;
    DataSet ds;
    SqlCommand cmd;
    string cs = ConfigurationManager.ConnectionStrings["conString"].ConnectionString;
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            PopulateList();
        }
    }
}
```

```
}  
protected void PopulateList()  
{  
    try  
    {  
        conn = new SqlConnection(cs);  
        adapter = new SqlDataAdapter("select * from tblCity", conn);  
        ds = new DataSet();  
        adapter.Fill(ds);  
        DropDownList1.DataSource = ds;  
        DropDownList1.DataTextField = "CityName";  
        DropDownList1.DataBind();  
    }  
    catch (Exception ex)  
    {  
        Label1.Text = "ERROR :: " + ex.Message;  
    }  
}  
protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)  
{  
    string SelectedItem = DropDownList1.SelectedItem.Text;  
    Label1.Text = "Favourite City = "+SelectedItem;  
}  
}
```

Output:



In the given example, PopulateList() is a user defined method to fill the DropDownList control. We can provide any name to populate the DropDownList control. When user selects the city name from DropDownList, SelectedIndexChanged event is fired and we will get the selected city name.

We can also use DataReader object to populate the DropDownList control as given below.

```
protected void PopulateList()
{
    cmd = new SqlCommand("select * from tblCity", conn);
    conn.Open();
    SqlDataReader reader = cmd.ExecuteReader();
    DropDownList1.DataSource = reader;
    DropDownList1.DataTextField = "CityName";
    DropDownList1.DataBind();
    conn.Close();
}
```

The DropDownList control renders an HTML <select> tag.

Important properties of DropDownList

- **Attributes:** It provides the collection of attributes.
- **DataTextField:** This property is used to set the column value of database table.
- **DataValueField:** It provides the value of each list item.
- **SelectedIndex:** It provides the index of selected item.
- **SelectedItem:** Gets the selected item from list.
- **SelectedValue:** Provides the value of selected item from the list.
- **Items:** Provides the collection of items from list.
- **Visible:** It is a Boolean property that is used to determine whether the control is visible on UI or not.

Working with the RadioButtonList Control

User can select only one item from RadioButtonList Control. This control works same as DropDownList but the UI is totally different. The RadioButtonList control displays a list of radio buttons. It can be arranged either horizontally or vertically.

```
protected void PopulateList()
{
    try
    {
        conn = new SqlConnection(cs);
        adapter = new SqlDataAdapter("select * from tblCity", conn);
        ds = new DataSet();
        adapter.Fill(ds);
        RadioButtonList1.DataSource = ds;
        RadioButtonList1.DataTextField = "CityName";
        RadioButtonList1.DataBind();
    }
    catch (Exception ex)
```

```

    {
        Label1.Text = "ERROR :: " + ex.Message;
    }
}

protected void RadioButtonList1_SelectedIndexChanged(object sender, EventArgs e)
{
    string SelectedItem = RadioButtonList1.SelectedItem.Text;
    Label2.Text = "Favourite City = " + SelectedItem;
}

```

For complete code please refer the previous code. SelectedIndexChanged is the default event of RadioButtonList. When we select any item from RadioButtonList, this event is fired. AutoPostBack property must be set as true otherwise we will not get the function of SelectedIndexChanged event. We can display any column value from database table as a list of item in the RadioButtonList control. For displaying the item in RadioButtonList control, set the value of DataTextField property. In the given example, we used "CityName" as the DataTextField property. We can provide any other column name as DataTextField property, according to need of application.



A screenshot of a web application interface. It features a vertical list of 12 Indian cities, each preceded by a radio button. The cities are: Mumbai, Delhi, Bengaluru, Hyderabad, Ahmedabad, Chennai, Kolkata, Pune, Jaipur, Lucknow, Kanpur, and Indore. The radio button for 'Jaipur' is selected, indicated by a filled circle. Below the list, a text label displays 'Favourite City = Jaipur' in a bold, red font.

ListBox Control

The ListBox control is similar to the DropDownList control but we can select more than one item from the list at a time. ListBox control has SelectionMode property that enables we to select multiple items from ListBox control. By default SelectionMode property is set as single. If we want to select multiple items from the ListBox, then set SelectionMode property value as Multiple and press Ctrl or Shift key when clicking more than one list item. Let us take one real world example. There are two ListBox control. One list box is populated with the city name that comes from the database table name tblCity. A user can select one or more item from the first ListBox and transfer into the second ListBox and vice-versa. User can also move the all item from ListBox one into another.

```
protected void PopulateList()
{
    try
    {
        conn = new SqlConnection(cs);
        adapter = new SqlDataAdapter("select * from tblCity", conn);
        ds = new DataSet();
        adapter.Fill(ds);
        ListBox1.DataSource = ds;
        ListBox1.DataTextField = "CityName";
        ListBox1.DataBind();
    }
    catch (Exception ex)
    {
        Label1.Text = "ERROR :: " + ex.Message;
    }
}

protected void Button1_Click(object sender, EventArgs e)
{
    foreach (ListItem item in ListBox1.Items)
    {
```

```
        if (item.Selected)
        {
            if (ListBox2.Items.Contains(item))
            {
                Label1.Text = "Item already present in second list";
            }
            else
            {
                ListBox2.Items.Add(item);
            }
        }
    }
}

protected void Button2_Click(object sender, EventArgs e)
{
    foreach (ListItem item in ListBox2.Items)
    {
        if (item.Selected)
        {
            if (ListBox1.Items.Contains(item))
            {
                Label1.Text = "Item already present in first list";
            }
            else
            {
                ListBox1.Items.Add(item);
            }
        }
    }
}

protected void Button3_Click(object sender, EventArgs e)
```

```
{
    while (ListBox1.Items.Count > 0)
    {
        for (int i = 0; i < ListBox1.Items.Count; i++)
        {
            ListBox2.Items.Add(ListBox1.Items[i]);
            ListBox1.Items.Remove(ListBox1.Items[i].Text);
        }
    }
}

protected void Button4_Click(object sender, EventArgs e)
{
    while (ListBox2.Items.Count > 0)
    {
        for (int i = 0; i < ListBox2.Items.Count; i++)
        {
            ListBox1.Items.Add(ListBox2.Items[i]);
            ListBox2.Items.Remove(ListBox2.Items[i].Text);
        }
    }
}
```

Output:



BulletedList Control

The BulletedList control enables we to display the list of items either a bulleted or ordered (numbered) list. By default BulletedList control displays the list of items in bulleted format. We can display each list item as plain text, a LinkButton control, or a link to another web page. BulletedList control supports the BulletStyle property. We can use this property to change the style of Bullets. BulletStyle property supports the following styles. Circle, CustomImage, Disc, LowerAlpha, LowerRoman, NotSet, Numbered, Square, UpperAlpha, UpperRoman.

Example

```
protected void PopulateList()
{
    try
    {
        conn = new SqlConnection(cs);
        adapter = new SqlDataAdapter("select * from tblCity", conn);
        ds = new DataSet();
        adapter.Fill(ds);
        BulletedList1.DataSource = ds;
```

```
BulletedList1.DataTextField = "CityName";  
BulletedList1.DataBind();  
}  
catch (Exception ex)  
{  
    Label1.Text = "ERROR :: " + ex.Message;  
}  
}
```



Applications and Virtual Directories in IIS

We can create a virtual directory that points to an ASP.NET application, and then convert the virtual directory to an application. Alternatively, we can directly convert a folder in the Connections pane of IIS Manager to an ASP.NET application without first creating a virtual directory. For information about how to open IIS Manager, see [IIS 7.0: Open IIS Manager](#)

To create a virtual directory with IIS Manager for an ASP.NET application

1. In IIS Manager, expand the local computer and the **Sites** folder.
2. Right-click the site or folder where we want to create the virtual directory and then click **Add Virtual Directory**.
3. In the **Add Virtual Directory** dialog box, specify the following information:

- **Alias.** Enter a name for the virtual directory.
 - **Physical Path.** Enter or browse to the physical directory that contains the virtual directory. We can select an existing folder or create a new one to contain the content for the virtual directory.
4. To provide credentials to connect to a UNC path, click the **Connect as** button.
 5. Click **OK**.
 6. Right-click the virtual directory that we created earlier and then click **Convert to Application**.

Configuring an ASP.NET Application

The default root location for IIS content is the %systemdrive%\inetpub\wwwroot directory. The instructions that follow assume that we have copied the ASP.NET application to the %systemdrive%\inetpub\wwwroot directory. If we want to specify another file location for the content, we must first create a virtual directory to the content. For more information, see the preceding procedure.

To configure an ASP.NET application under the root Web site.

1. In IIS Manager, expand the node for the local computer and then expand the **Sites** folder.
2. Right-click the folder that we want to convert to an application and then click **Convert to Application**.

The **Add Application** dialog box is displayed.

3. Click **OK**.

Accessing Data using ADO.NET, Connecting to Data, Executing Commands

ADO.NET data provider examples

The following code listings demonstrate how to retrieve data from a database using ADO.NET data providers. The data is returned in a `DataReader`. For more information, see [Retrieving Data Using a DataReader](#).

SqlClient

The code in this example assumes that we can connect to the Northwind sample database on Microsoft SQL Server. The code creates a [SqlCommand](#) to select rows from the Products table, adding a [SqlParameter](#) to restrict the results to rows with a UnitPrice greater than the specified parameter value, in this case 5. The [SqlConnection](#) is opened inside a using block, which ensures that resources are closed and disposed when the code exits. The code executes the command by using a [SqlDataReader](#), and displays the results in the console window. If we're using `System.Data.SqlClient`, we should consider upgrading to `Microsoft.Data.SqlClient` as it's where future investments and new feature developments are being made.

```
using System;
```

```
using System.Data;
```

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

```
class Program
```

```
{
```

```
    static void Main()
```

```
    {
```

```
        string connectionString =
```

```
            "Data Source=(local);Initial Catalog=Northwind;"
```

```
            + "Integrated Security=true";
```

```
        // Provide the query string with a parameter placeholder.
```

```
        string queryString =
```

```

"SELECT ProductID, UnitPrice, ProductName from dbo.products "
+ "WHERE UnitPrice > @pricePoint "
+ "ORDER BY UnitPrice DESC;";

// Specify the parameter value.
int paramValue = 5;

// Create and open the connection in a using block. This
// ensures that all resources will be closed and disposed
// when the code exits.
using (SqlConnection connection =
    new SqlConnection(connectionString))
{
    // Create the Command and Parameter objects.
    SqlCommand command = new SqlCommand(queryString, connection);
    command.Parameters.AddWithValue("@pricePoint", paramValue);

    // Open the connection in a try/catch block.
    // Create and execute the DataReader, writing the result
    // set to the console window.
    try
    {
        connection.Open();
        SqlDataReader reader = command.ExecuteReader();
        while (reader.Read())
        {
            Console.WriteLine("{0}\t{1}\t{2}",
                reader[0], reader[1], reader[2]);
        }
        reader.Close();
    }
}

```

```

        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
        Console.ReadLine();
    }
}

```

OleDb

The code in this example assumes that we can connect to the Microsoft Access Northwind sample database. The code creates a [OleDbCommand](#) to select rows from the Products table, adding a [OleDbParameter](#) to restrict the results to rows with a UnitPrice greater than the specified parameter value, in this case 5. The [OleDbConnection](#) is opened inside of a using block, which ensures that resources are closed and disposed when the code exits. The code executes the command by using a [OleDbDataReader](#), and displays the results in the console window.

```

using System;
using System.Data;
using System.Data.OleDb;

class Program
{
    static void Main()
    {
        // The connection string assumes that the Access
        // Northwind.mdb is located in the c:\Data folder.
        string connectionString =
            "Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
            + "c:\\Data\\Northwind.mdb;User Id=admin;Password=";

        // Provide the query string with a parameter placeholder.
    }
}

```

```

string queryString =
    "SELECT ProductID, UnitPrice, ProductName from products "
    + "WHERE UnitPrice > ? "
    + "ORDER BY UnitPrice DESC;";

// Specify the parameter value.
int paramValue = 5;

// Create and open the connection in a using block. This
// ensures that all resources will be closed and disposed
// when the code exits.
using (OleDbConnection connection =
    new OleDbConnection(connectionString))
{
    // Create the Command and Parameter objects.
    OleDbCommand command = new OleDbCommand(queryString, connection);
    command.Parameters.AddWithValue("@pricePoint", paramValue);

    // Open the connection in a try/catch block.
    // Create and execute the DataReader, writing the result
    // set to the console window.
    try
    {
        connection.Open();
        OleDbDataReader reader = command.ExecuteReader();
        while (reader.Read())
        {
            Console.WriteLine("\t{0}\t{1}\t{2}",
                reader[0], reader[1], reader[2]);
        }
        reader.Close();
    }
}

```

```

    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    Console.ReadLine();
}
}

```

Odbc

The code in this example assumes that we can connect to the Microsoft Access Northwind sample database. The code creates a [OdbcCommand](#) to select rows from the Products table, adding a [OdbcParameter](#) to restrict the results to rows with a UnitPrice greater than the specified parameter value, in this case 5. The [OdbcConnection](#) is opened inside a using block, which ensures that resources are closed and disposed when the code exits. The code executes the command by using a [OdbcDataReader](#), and displays the results in the console window.

```

using System;
using System.Data;
using System.Data.Odbc;

class Program
{
    static void Main()
    {
        // The connection string assumes that the Access
        // Northwind.mdb is located in the c:\Data folder.
        string connectionString =
            "Driver={Microsoft Access Driver (*.mdb)};"
            + "Dbq=c:\\Data\\Northwind.mdb;Uid=Admin;Pwd=";
    }
}

```



```

// Provide the query string with a parameter placeholder.
string queryString =
    "SELECT ProductID, UnitPrice, ProductName from products "
    + "WHERE UnitPrice > ? "
    + "ORDER BY UnitPrice DESC;";

// Specify the parameter value.
int paramValue = 5;

// Create and open the connection in a using block. This
// ensures that all resources will be closed and disposed
// when the code exits.
using (OdbcConnection connection =
    new OdbcConnection(connectionString))
{
    // Create the Command and Parameter objects.
    OdbcCommand command = new OdbcCommand(queryString, connection);
    command.Parameters.AddWithValue("@pricePoint", paramValue);

    // Open the connection in a try/catch block.
    // Create and execute the DataReader, writing the result
    // set to the console window.
    try
    {
        connection.Open();
        OdbcDataReader reader = command.ExecuteReader();
        while (reader.Read())
        {
            Console.WriteLine("{0}\t{1}\t{2}",
                reader[0], reader[1], reader[2]);
        }
    }
}

```

```

        reader.Close();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
    Console.ReadLine();
}
}
}

```

OracleClient

The code in this example assumes a connection to DEMO.CUSTOMER on an Oracle server. We must also add a reference to the System.Data.OracleClient.dll. The code returns the data in an [OracleDataReader](#).

```

using System;
using System.Data;
using System.Data.OracleClient;

class Program
{
    static void Main()
    {
        string connectionString =
            "Data Source=ThisOracleServer;Integrated Security=yes;";
        string queryString =
            "SELECT CUSTOMER_ID, NAME FROM DEMO.CUSTOMER";
        using (OracleConnection connection =
            new OracleConnection(connectionString))
    }
}

```

```

{
    OracleCommand command = connection.CreateCommand();
    command.CommandText = queryString;

    try
    {
        connection.Open();

        OracleDataReader reader = command.ExecuteReader();

        while (reader.Read())
        {
            Console.WriteLine("\t{0}\t{1}",
                reader[0], reader[1]);
        }
        reader.Close();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}
}

```

Entity Framework examples

The following code listings demonstrate how to retrieve data from a data source by querying entities in an Entity Data Model (EDM). These examples use a model based on the Northwind sample database. For more information about Entity Framework,

LINQ to Entities

The code in this example uses a LINQ query to return data as Categories objects, which are projected as an anonymous type that contains only the CategoryID and CategoryName properties. For more information, see [LINQ to Entities Overview](#).

```
using System;
using System.Linq;
using System.Data.Objects;
using NorthwindModel;

class LinqSample
{
    public static void ExecuteQuery()
    {
        using (NorthwindEntities context = new NorthwindEntities())
        {
            try
            {
                var query = from category in context.Categories
                            select new
                            {
                                categoryID = category.CategoryID,
                                categoryName = category.CategoryName
                            };

                foreach (var categoryInfo in query)
                {
                    Console.WriteLine("\t{0}\t{1}",
                                      categoryInfo.categoryID, categoryInfo.categoryName);
                }
            }
        }
    }
}
```

```

        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
}

```

Typed ObjectQuery

The code in this example uses an Object Query<T> to return data as Categories objects.

```

using System;
using System.Data.Objects;
using NorthwindModel;

class ObjectQuerySample
{
    public static void ExecuteQuery()
    {
        using (NorthwindEntities context = new NorthwindEntities())
        {
            ObjectQuery<Categories> categoryQuery = context.Categories;

            foreach (Categories category in
                categoryQuery.Execute(MergeOption.AppendOnly))
            {
                Console.WriteLine("\t{0}\t{1}",
                    category.CategoryID, category.CategoryName);
            }
        }
    }
}

```

EntityClient

The code in this example uses an [EntityCommand](#) to execute an Entity SQL query. This query returns a list of records that represent instances of the Categories entity type. An [EntityDataReader](#) is used to access data records in the result set.

```
using System;
using System.Data;
using System.Data.Common;
using System.Data.EntityClient;
using NorthwindModel;

class EntityClientSample
{
    public static void ExecuteQuery()
    {
        string queryString =
            @"SELECT c.CategoryID, c.CategoryName
            FROM NorthwindEntities.Categories AS c";

        using (EntityConnection conn =
            new EntityConnection("name=NorthwindEntities"))
        {
            try
            {
                conn.Open();
                using (EntityCommand query = new EntityCommand(queryString, conn))
                {
                    using (DbDataReader rdr =
                        query.ExecuteReader(CommandBehavior.SequentialAccess))
                    {
                        while (rdr.Read())
```

```

        {
            Console.WriteLine("\t{0}\t{1}", rdr[0], rdr[1]);
        }
    }
}
}
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
}
}
}

```

LINQ to SQL

The code in this example uses a LINQ query to return data as Categories objects, which are projected as an anonymous type that contains only the CategoryID and CategoryName properties. This example is based on the Northwind data context.

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Northwind;

class LinqSqlSample
{
    public static void ExecuteQuery()
    {
        using (NorthwindDataContext db = new NorthwindDataContext())
        {

```

```

try
{
    var query = from category in db.Categories
                select new
                {
                    categoryID = category.CategoryID,
                    categoryName = category.CategoryName
                };

    foreach (var categoryInfo in query)
    {
        Console.WriteLine("vbTab {0} vbTab {1}",
            categoryInfo.categoryID, categoryInfo.categoryName);
    }
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
}
}

```

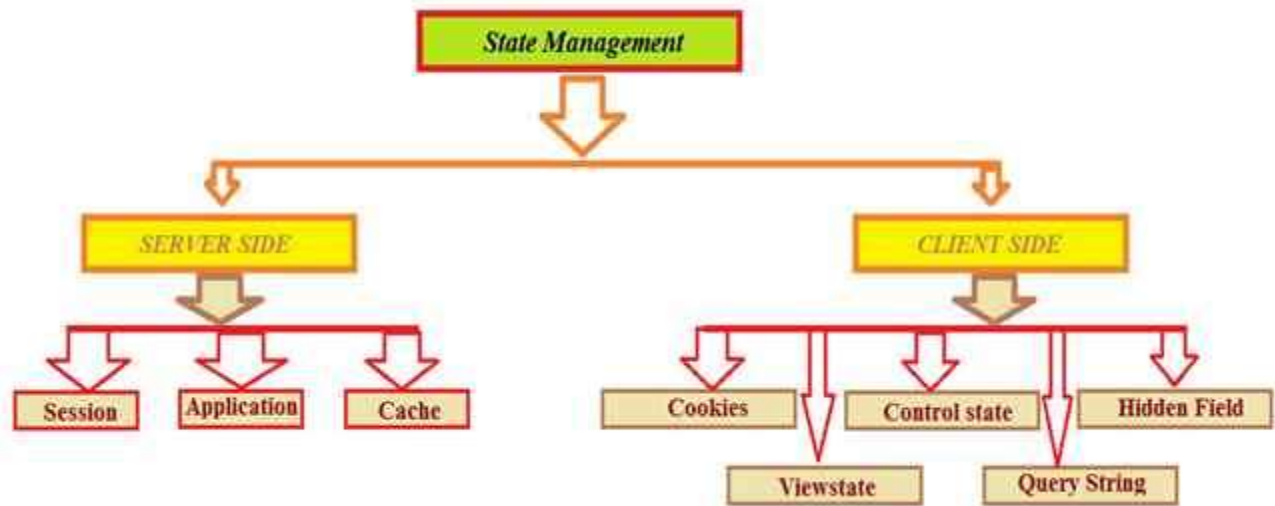
State management Page-Level state,

State management is very important and useful in ASP.NET. It is also asked in many interviews to fresher and experienced developers.

ASP.NET State management is a preserve state control and object in an application because ASP.NET web applications are stateless. A new instance of the Web page class is created each time the page is posted to the server. If a user enters information into a web application, that information would be lost in the round trip from the browser (MSDN).

In a single line, State management maintains and stores the information of any user till the end of the user session.

Two types of State Management techniques are available in ASP.NET as in the following figure,



Server side

Session

Session is a very important technique to maintain state. Normally session is used to store information and identity. The server stores information using Sessionid.

Set User Session

```
1. protected void btnSubmit_Click(object sender, EventArgs e)
2. {
3.     Session["UserName"] = txtName.Text;
4.
5.     Response.Redirect("Home.aspx");
6. }
```

Session Event

Session event can be seen in project *Global.asax* file.

Two types of Session Events

Session_Start

The Session_start event is raised every time a new user requests without a session ID.

Session_End

The Session_End event is raised when session is ended by a user or a time out using Session end method.

```
void Session_End(object sender, EventArgs e)
{
    Response.Write("Session_End");
}
```

The session is stored in the following for ways in ASP.NET.

- *InProcMode*

It is a default session mode and a value store in web server memory (IIS). In this the session value stored with server start and it ends when the server is restarted.

- *State Server Mode*

In this mode session data is stored in separate server.

- *SQL Server Mode*

In this session is stored in the database. It is a secure mode.

- *Custom Mode*

Generally under session data is stored in InProc, Sql Server, State server, etc. If we store session data with other new techniques then provide ASP.NET.

Application

Application State is a server side management state. It is also called application level state management. In this mainly store user activity in server memory and application event shown in Global.asax file.

There are three types of applications in ASP.NET.

Application_Start

This event begins with domain start.

```
Void Application_Start(object sender, EventArgs e)
{
    Application["AppstartMessage"] = "Welcome to CSharp Corner Developer Communtiy"
}
```

Application_Error

In this section manage unhandled exception error.

```
void Application_Error(object sender, EventArgs e)
{
    // Write an unhandled error code exception
}
```

Application_End

This ends with domain or restarts IIS.

```
Void Application_End(object sender, EventArgs e)
```

```
{  
    Application["AppEndMessage"] = "Application Closed";  
}
```

Cache

Cache is stored on server side. It implements Page Caching and data caching. Cache is use to set expiration polices

```
Response.Cache.SetExpiresTime(DateTime.Now.AddDays(1));
```

Client Side

Now here I am explaining client side state management one by one:

Also state management has the following four important parts available on the client side,

Control state

Control state technique is developed to maintain data work properly in order. We can use view state but suppose view state is disabled by the user, the control will not work as expected. For expected results of the control we have to use Control State. In application, the Viewstate is by default true. Sometimes we need to use custom control to manage application properly.

```
if (!IsPostBack)  
{  
    lblmsg1.Text = "Welcome to C# corner";  
    lblmsg2.Text = "Welcome to C# corner community";  
}
```

When two messages are displayed on a Postback event, then control which one is displayed by using customized control state.

Hidden Field

Hidden fields are used to store value to client side. Hidden field is not displayed on the browser, but it works on a request.

```
if (HiddenField1.Value != null)
{
    int val = Convert.ToInt32(HiddenField1.Value) + 1;
    HiddenField1.Value = val.ToString();
    Label1.Text = val.ToString();
}
```

Viewstate

Viewstate is a very useful client side property. It is used for page level state management. Viewstate stores any type of data and used for sending and receiving information,

Example 1 - Count.

ViewState Demo Count: <asp:Label runat="server" id="lblcount" />

<asp:Button runat="server" id="Submit" onclick="Submit_Click" text="show" />

```
protected void Page_Load(object sender, EventArgs e)
```

```
{
```

```
if (IsPostBack)

{

    if (ViewState["count"] != null)

    {

        int ViewstateVal = Convert.ToInt32(ViewState["count"]) + 1;

        lblcount.Text = ViewstateVal.ToString();

        ViewState["count"] = ViewstateVal.ToString();

    }

    else

    {

        ViewState["count"] = "1";

    }

}

}
```

```
protected void Submit_Click(object sender, EventArgs e)
```

```
{

    lblcount.Text = ViewState["count"].ToString();

}
```

Example 2 - Set/Get user.

```
if (ViewState["UserName"] != null)

    lblName.Text = ViewState["UserName"].ToString();
```

OR,

```
ViewState["UserName"] = txtUserName.Text;
```

Viewstate is easy to apply and does not need access to any server resources. In a Viewstate, do not store big data, only store small values. Viewstate enables and disables on page level control. It also supports Encryption and Decryption and data/value is stored in hashed format. So we are not storing important data such as password, account information, etc. When more data is stored in this, then the page becomes heavy.

Query String

Query string stores the value in URL.

```
Response.Redirect("ShowStringValue.aspx?Username=" + txtUsername.Text);
```

It is visible to all the users in url as in the following link,



Cookies to preserve state

Cookie is a small and an important part of ASP.NET. In this store user information, session and application. It can be created constant and temporary and they work with browser request. Cookies are store on client side. The server can read cookies and abstract data.

Two types of cookies are available,

Persistence

this type of cookie works with Date and time.

```
Response.Cookies["CookieName"].Value = "Test Cookies";  
  
//set expire time  
  
Response.Cookies["CookieName"].Expires = DateTime.Today.AddHours(1);
```

Non-Persistence

This is a temporary cookie. It is created with access application and discards the close application.

```
Response.Cookies["CookieName"].Value = "Test Cookies";
```

ASP.NET Cookie is a small bit of text that is used to store user-specific information. This information can be read by the web application whenever user visits the site.

When a user requests for a web page, web server sends not just a page, but also a cookie containing the date and time. This cookie stores in a folder on the user's hard disk.

When the user requests for the web page again, browser looks on the hard drive for the cookie associated with the web page. Browser stores separate cookie for each different sites user visited.

Note: The Cookie is limited to small size and can be used to store only 4 KB (4096 Bytes) text.

There are two ways to store cookies in ASP.NET application.

- Cookies collection
- HttpCookie

We can add Cookie either to Cookies collection or by creating instance of HttpCookie class. both work same except that HttpCookie require Cookie name as part of the constructor.

HttpCookie Example

In the following example, we are creating and adding cookie with the help of HttpCookie class.

```
// CookieExample.aspx
<% @ Page Language="C#" AutoEventWireup="true"
CodeBehind="CookieExample.aspx.cs" Inherits="CookieExample.CookieExample" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
        </div>
    </form>
</body>
</html>
```

Code

```
// CookieExample.aspx.cs
using System;
using System.Web;
namespace WebFormsControlls
{
    public partial class CookieExample : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
```

```

//----- Creating Cookie -----//
// Creating HttpCookie instance by specifying name "student"
    HttpCookie cokie = new HttpCookie("student");
// Assigning value to the created cookie
    cokie.Value = "Rahul Kumar";
// Adding Cookie to the response instance
    Response.Cookies.Add(cokie);
//----- Fetching Cookie -----//
    var co_val = Response.Cookies["student"].Value;
    Label1.Text = co_val;
}
}
}

```

In the following example, we are adding cookie directly to the Cookies collection.

```

// Default.aspx
<%@ Page Title="Home Page" Language="C#" AutoEventWireup="true" CodeBehind="Default
t.aspx.cs"
Inherits="CookieExample._Default" %>
<form id="form1" runat="server">
    <asp:Label ID="Label1" runat="server" Text="Select Brand Preferences"></asp:Label>
    <br />    <br />
    <asp:CheckBox ID="apple" runat="server" Text="Apple" />
    <br />
    <asp:CheckBox ID="dell" runat="server" Text="Dell" />
    <br />
    <br />
    <asp:CheckBox ID="acer" runat="server" Text="Acer" />
    <br />
    <asp:CheckBox ID="sony" runat="server" Text="Sony" />
<br />

```

```

<asp:CheckBox ID="wipro" runat="server" Text="Wipro" />
<br />
<br />
<asp:Button ID="Button1" runat="server" OnClick="Button1_Click" Text="Submit" />
<p>
    <asp:Label ID="Label2" runat="server"></asp:Label>
</p>
</form>

```

CodeBehind

// Default.aspx.cs

```

using System;
using System.Web.UI;
namespace CookieExample
{
    public partial class _Default : Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            // Setting expiring date and time of the cookies
            Response.Cookies["computer"].Expires = DateTime.Now.AddDays(-1);
        }

        protected void Button1_Click(object sender, EventArgs e)
        {
            Label2.Text = "";
            // ----- Adding Cookies -----//
            if (apple.Checked)
                Response.Cookies["computer"]["apple"] = "apple";
            if (dell.Checked)
                Response.Cookies["computer"]["dell"] = "dell";
            if (lenevo.Checked)

```

```

        Response.Cookies["computer"]["lenevo"] = "lenevo";
    if (acer.Checked)
        Response.Cookies["computer"]["acer"] = "acer";
    if (sony.Checked)
        Response.Cookies["computer"]["sony"] = "sony";
    if (wipro.Checked)
        Response.Cookies["computer"]["wipro"] = "wipro";
    // ----- Fetching Cookies -----//
    if (Request.Cookies["computer"].Values.ToString() != null)
    {
        if (Request.Cookies["computer"]["apple"] != null)
            Label2.Text += Request.Cookies["computer"]["apple"] + " ";
        if (Request.Cookies["computer"]["dell"] != null)
            Label2.Text += Request.Cookies["computer"]["dell"] + " ";
        if (Request.Cookies["computer"]["lenevo"] != null)
            Label2.Text += Request.Cookies["computer"]["lenevo"] + " ";
        if (Request.Cookies["computer"]["acer"] != null)
            Label2.Text += Request.Cookies["computer"]["acer"] + " ";
        if (Request.Cookies["computer"]["sony"] != null)
            Label2.Text += Request.Cookies["computer"]["sony"] + " ";
        if (Request.Cookies["computer"]["wipro"] != null)
            Label2.Text += Request.Cookies["computer"]["wipro"] + " ";
    } else Label2.Text = "Please select wer choice";
    Response.Cookies["computer"].Expires = DateTime.Now.AddDays(-1);
}
}
}

```

Output:

This example will store selected values as cookie.

http://localhost:52385/Default.aspx

localhost:52385 120% Search

Select Brand Preferences

- ☐ Apple
- ☐ Dell
- ☐ Lenevo
- ☐ Acer
- ☐ Sony
- ☐ Wipro

Submit

http://localhost:52385/Default.aspx

localhost:52385 120% Search

Select Brand Preferences

- ☐ Apple
- ☐ Dell
- ☒ Lenevo
- ☐ Acer
- ☐ Sony
- ☒ Wipro

Submit

lenevo wipro

ASP.NET Session State, Storing Object in Session State, Configuring Session State

ASP.NET Session

In ASP.NET session is a state that is used to store and retrieve values of a user.

It helps to identify requests from the same browser during a time period (session). It is used to store value for the particular time session. By default, ASP.NET session state is enabled for all ASP.NET applications.

Each created session is stored in **SessionStateItemCollection** object. We can get current session value by using **Session** property of **Page** object. Let's see an example, how to create an access session in asp.net application.

In the following example, we are creating a session and storing user email. This example contains the following files.

// Default.aspx

```
<% @ Page Title="Home Page" Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="SessionExample._Default" %>

<head>
    <style type="text/css">
        .auto-style1 {
            width: 100%;
        }
        .auto-style2 {
            width: 105px;
        }
    </style>
</head>

<form id="form1" runat="server">
    <p>Provide Following Details</p>
```

```

<table class="auto-style1">
  <tr>
    <td class="auto-style2">Email</td>
    <td>
      <asp:TextBox ID="email" runat="server" TextMode="Email"></asp:TextBox>
    </td>
  </tr>
  <tr>
    <td class="auto-style2">Password</td>
    <td>
      <asp:TextBox ID="password" runat="server" TextMode="Password"></asp:TextBo>
    </td>
  </tr>
  <tr>
    <td class="auto-style2"> </td>
    <td>
      <asp:Button ID="login" runat="server" Text="Login" OnClick="login_Click" />
    </td>
  </tr>
</table>
<br />
<asp:Label ID="Label3" runat="server"></asp:Label>
<br />
<asp:Label ID="Label4" runat="server"></asp:Label>

```

// Default.aspx.cs

```

using System;
using System.Web.UI;
namespace SessionExample
{
    public partial class _Default : Page

```

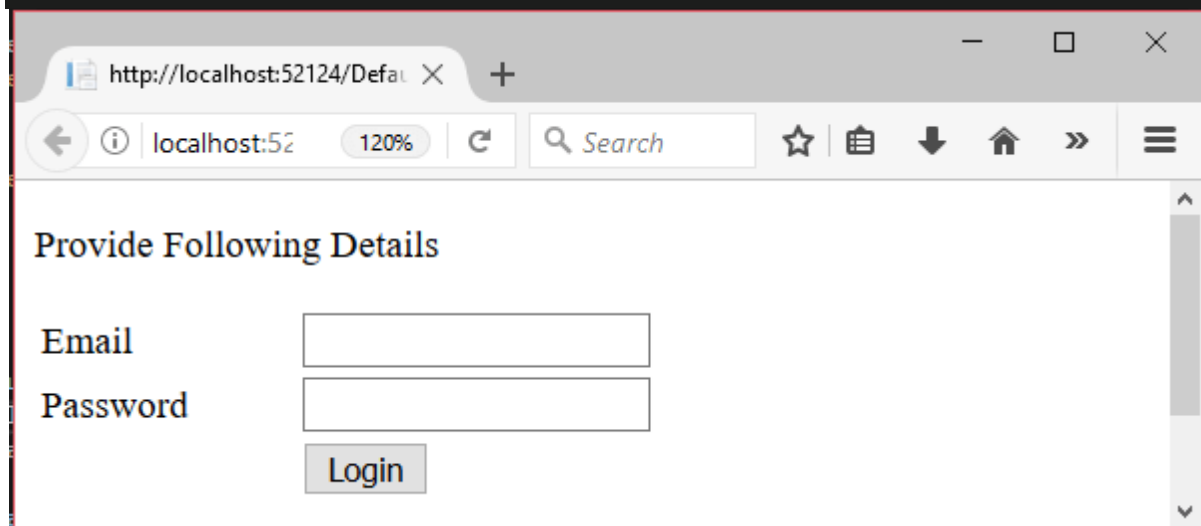
```

{
    protected void login_Click(object sender, EventArgs e)
    {
        if (password.Text=="qwe123")
        {
            // Storing email to Session variable
            Session["email"] = email.Text;
        }
        // Checking Session variable is not empty
        if (Session["email"] != null)
        {
            // Displaying stored email
            Label3.Text = "This email is stored to the session.";
            Label4.Text = Session["email"].ToString();
        }
    }
}

```

Output:

This application will store user email to the session when user login.



The screenshot shows a web browser window with the address bar displaying 'http://localhost:52124/Default.aspx'. The page content includes a heading 'Provide Following Details' and a login form. The form consists of two text input fields labeled 'Email' and 'Password', and a 'Login' button positioned below the 'Password' field. The browser's address bar also shows 'localhost:52' and a '120%' zoom level. The browser interface includes standard navigation icons like back, forward, and home, as well as a search bar.

It will show stored session value, user email.

Provide Following Details

Email

Password

This email is stored to the session.
kapil@abc.com

Validation

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

ASP.NET provides the following validation controls:

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

BaseValidator Class

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

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

RequiredFieldValidator Control

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

The syntax of the control is as given:

```
<asp:RequiredFieldValidator ID="rfvcandidate"
    runat="server" ControlToValidate ="ddlcandidate"
    ErrorMessage="Please choose a candidate"
    InitialValue="Please choose a candidate">
```

```
</asp:RequiredFieldValidator>
```

RangeValidator Control

The RangeValidator control verifies that the input value falls within a predetermined range.

It has three specific properties:

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

The syntax of the control is as given:

```
<asp:RangeValidator ID="rvclass" runat="server" ControlToValidate="txtclass"
  ErrorMessage="Enter wer class (6 - 12)" MaximumValue="12"
  MinimumValue="6" Type="Integer">

</asp:RangeValidator>
```

CompareValidator Control

The CompareValidator control compares a value in one control with a fixed value or a value in another control.

It has the following specific properties:

Properties	Description
Type	It specifies the data type.

ControlToCompare	It specifies the value of the input control to compare with.
ValueToCompare	It specifies the constant value to compare with.
Operator	It specifies the comparison operator, the available values are: Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual, and DataTypeCheck.

The basic syntax of the control is as follows:

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

</asp:CompareValidator>
```

RegularExpressionValidator

The RegularExpressionValidator allows validating the input text by matching against a pattern of a regular expression. The regular expression is set in the ValidationExpression property.

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

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

\	Escape character.
---	-------------------

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

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

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

Quantifier	Description
*	Zero or more matches.
+	One or more matches.
?	Zero or one matches.

{N}	N matches.
{N,}	N or more matches.
{N,M}	Between N and M matches.

The syntax of the control is as given:

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

</asp:RegularExpressionValidator>
```

CustomValidator

The CustomValidator control allows writing application specific custom validation routines for both the client side and the server side validation.

The client side validation is accomplished through the ClientValidationFunction property. The client side validation routine should be written in a scripting language, such as JavaScript or VBScript, which the browser can understand.

The server side validation routine must be called from the control's ServerValidate event handler. The server side validation routine should be written in any .Net language, like C# or VB.Net.

The basic syntax for the control is as given:

```
<asp:CustomValidator ID="CustomValidator1" runat="server"
  ClientValidationFunction=.cvf_func. ErrorMessage="CustomValidator">

</asp:CustomValidator>
```

ValidationSummary

The ValidationSummary control does not perform any validation but shows a summary of all errors in the page. The summary displays the values of the ErrorMessage property of all validation controls that failed validation.

The following two mutually inclusive properties list out the error message:

- **ShowSummary** : shows the error messages in specified format.
- **ShowMessageBox** : shows the error messages in a separate window.

The syntax for the control is as given:

```
<asp:ValidationSummary ID="ValidationSummary1" runat="server"  
    DisplayMode = "BulletList" ShowSummary = "true" HeaderText="Errors:" />
```

Validation Groups

Complex pages have different groups of information provided in different panels. In such situation, a need might arise for performing validation separately for separate group. This kind of situation is handled using validation groups.

To create a validation group, we should put the input controls and the validation controls into the same logical group by setting their *ValidationGroup* property.

Example

The following example describes a form to be filled up by all the students of a school, divided into four houses, for electing the school president. Here, we use the validation controls to validate the user input.

This is the form in design view:

The content file code is as given:

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

    <table style="width: 66%;">

        <tr>
            <td class="style1" colspan="3" align="center">
                <asp:Label ID="lblmsg"
                    Text="President Election Form : Choose wer president"
                    runat="server" />
            </td>
        </tr>

        <tr>
            <td class="style3">
                Candidate:
            </td>

            <td class="style2">
                <asp:DropDownList ID="ddlcandidate" runat="server" style="width:239px">
                    <asp:ListItem>Please Choose a Candidate</asp:ListItem>
                    <asp:ListItem>M H Kabir</asp:ListItem>
                </td>
        </tr>
    </table>
</form>
```



```

        <asp:ListItem>Steve Taylor</asp:ListItem>
        <asp:ListItem>John Abraham</asp:ListItem>
        <asp:ListItem>Venus Williams</asp:ListItem>
    </asp:DropDownList>
</td>

<td>
    <asp:RequiredFieldValidator ID="rfvcandidate"
        runat="server" ControlToValidate ="ddlcandidate"
        ErrorMessage="Please choose a candidate"
        InitialValue="Please choose a candidate">
    </asp:RequiredFieldValidator>
</td>
</tr>

<tr>
    <td class="style3">
        House:
    </td>

    <td class="style2">
        <asp:RadioButtonList ID="rblhouse" runat="server" RepeatLayout="Flow">
            <asp:ListItem>Red</asp:ListItem>
            <asp:ListItem>Blue</asp:ListItem>
            <asp:ListItem>Yellow</asp:ListItem>
            <asp:ListItem>Green</asp:ListItem>
        </asp:RadioButtonList>
    </td>

    <td>
        <asp:RequiredFieldValidator ID="rfvhouse" runat="server"

```

```

        ControlToValidate="rblhouse" ErrorMessage="Enter wer house name" >
    </asp:RequiredFieldValidator>

    <br />

</td>

</tr>

<tr>

    <td class="style3">
        Class:
    </td>

    <td class="style2">
        <asp:TextBox ID="txtclass" runat="server"></asp:TextBox>
    </td>

    <td>
        <asp:RangeValidator ID="rvclass"
            runat="server" ControlToValidate="txtclass"
            ErrorMessage="Enter wer class (6 - 12)" MaximumValue="12"
            MinimumValue="6" Type="Integer">
        </asp:RangeValidator>
    </td>

</tr>

<tr>

    <td class="style3">
        Email:
    </td>

    <td class="style2">
        <asp:TextBox ID="txtemail" runat="server" style="width:250px">

```

```

        </asp:TextBox>
    </td>

    <td>
        <asp:RegularExpressionValidator ID="remail" runat="server"
            ControlToValidate="txtemail" ErrorMessage="Enter wer email"
            ValidationExpression="\w+([-+.']\w+)*@\w+([-.']\w+)*\.\w+([-.']\w+)*">
        </asp:RegularExpressionValidator>
    </td>
</tr>

<tr>
    <td class="style3" align="center" colspan="3">
        <asp:Button ID="btnsubmit" runat="server" onClick="btnsubmit_Click"
            style="text-align: center" Text="Submit" style="width:140px" />
    </td>
</tr>
</table>
<asp:ValidationSummary ID="ValidationSummary1" runat="server"
    DisplayMode ="BulletList" ShowSummary ="true" HeaderText="Errors:" />
</form>

```

The code behind the submit button:

```

protected void btnsubmit_Click(object sender, EventArgs e)
{
    if (Page.IsValid)
    {
        lblmsg.Text = "Thank We";
    }
    else
    {

```

```
lblmsg.Text = "Fill up all the fields";  
}  
}
```

IIS URL Authorization

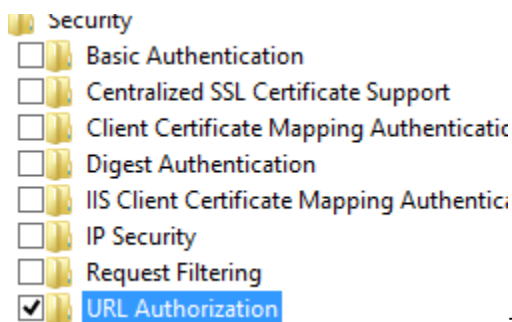
The <authorization> element allows you to configure the user accounts that can access your site or application. Use authorization in combination with authentication to secure access to content on your server. Authentication confirms the identity of a user, while authorization determines what resources users can or cannot access.

IIS defines two types of authorization rules, Allow rules and Deny rules:

- Allow rules let us define the user accounts or user groups that can access a site, an application, or all the sites on a server.
- Deny rules let us define the user accounts or user groups that cannot access a site, an application, or all the sites on a server.

Windows 8 or Windows 8.1

1. On the **Start** screen, move the pointer all the way to the lower left corner, right-click the **Start** button, and then click **Control Panel**. - In **Control Panel**, click **Programs and Features**, and then click **Turn Windows features on or off**. - Expand **Internet Information Services**, expand **World Wide Web Services**, expand **Security**, and then select **URL Authorization**.



- Click **OK**.

2. Click **Close**.

How to add an authorization rule

1. Open **Internet Information Services (IIS) Manager**:

- If you are using Windows Server 2012 or Windows Server 2012 R2:
 - On the taskbar, click **Server Manager**, click **Tools**, and then click **Internet Information Services (IIS) Manager**.
- If you are using Windows 8 or Windows 8.1:
 - Hold down the **Windows** key, press the letter **X**, and then click **Control Panel**.
 - Click **Administrative Tools**, and then double-click **Internet Information Services (IIS) Manager**.
- If you are using Windows Server 2008 or Windows Server 2008 R2:
 - On the taskbar, click **Start**, point to **Administrative Tools**, and then click **Internet Information Services (IIS) Manager**.
- If you are using Windows Vista or Windows 7:
 - On the taskbar, click **Start**, and then click **Control Panel**.
 - Double-click **Administrative Tools**, and then double-click **Internet Information Services (IIS) Manager**.

2. In the **Connections** pane, expand the server name, expand **Sites**, and then navigate to the site or application on which you want to configure authorization.

3. In the **Home** pane, double-click **Authorization Rules**.



4. To add a new authorization rule, in the **Actions** pane click **Add Allow Rule...** or **Add Deny Rule...**

5. Apply the authorization settings needed for your site or application, and then click **OK**. For example:

- Example #1: Adding an Allow rule for all users for specific HTTP verbs:

☐ Specified user:

Example: User1, User2

☒ Apply this rule to specific verbs:

GET,HEAD,POST

Example: GET, POST

- Example #2: Adding a Deny rule for a specific user for all HTTP verbs:

Add Deny Authorization Rule

Deny access to this Web content to:

☐ All users

☐ All anonymous users

☐ Specified roles or user groups:

Configuration

We can configure the <authorization> element at the server level in the ApplicationHost.config file, or at the site or application level in the appropriate Web.config file.

We can set default authorization rules for the entire server by configuring authorization rules at the server level. We can remove, clear, or override these rules by configuring more specific rules for your sites or applications.

Attributes

Attribute	Description
bypassLoginPage	Optional Boolean attribute. Specifies whether to skip authorization check for the page specified as the login page for Forms authentication. This enables unauthenticated users to access the login page to log on. The default value is true.

Child Elements

Element	Description
add	Optional element. Adds an authorization rule to the collection of authorization rules.
remove	Optional element. Removes a reference to an authorization rule to the collection of authorization rules.
clear	Optional element. Removes all references to authorization rules from the collection of authorization rules.

Configuration Sample

The following configuration example, when included in a Web.config file, removes the default IIS authorization settings, which allows all users' access to Web site or application content. It then configures an authorization rule that allows only users with administrator privileges to access the content.

```
<configuration>
  <system.webServer>
```

```
<security>
  <authorization>
    <remove users="*" roles="" verbs="" />
    <add accessType="Allow" users="" roles="Administrators" />
  </authorization>
</security>
</system.webServer>
</configuration>
```

```
using System;

using System.Text;

using Microsoft.Web.Administration;

internal static class Sample
{
    private static void Main()
    {
        using (ServerManager serverManager = new ServerManager())
        {
            Configuration config = serverManager.GetWebConfiguration("Contoso"); ConfigurationSection
            authorizationSection = config.GetSection("system.webServer/security/authorization");

            ConfigurationElementCollection authorizationCollection = authorizationSection.GetCollection();

            ConfigurationElement addElement = authorizationCollection.CreateElement("add");

            addElement["accessType"] = @"Allow";

            addElement["roles"] = @"administrators";
```



```
        authorizationCollection.Add(addElement);  
  
        serverManager.CommitChanges();  
    }  
}  
}
```

Forms Authentication

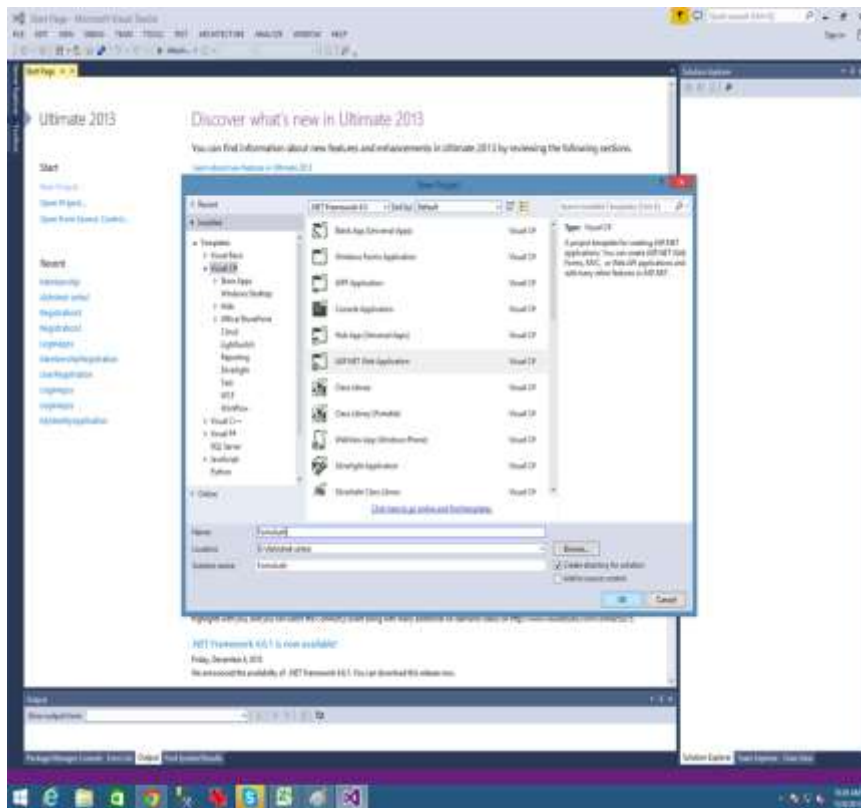
Form authentication is used for internet web application. The advantage of form authentication is that users do not have to be member of a domain-based network to have access to your application. So the number of web application uses the form authentication in their web application.

There are three types of authentication in ASP.NET,

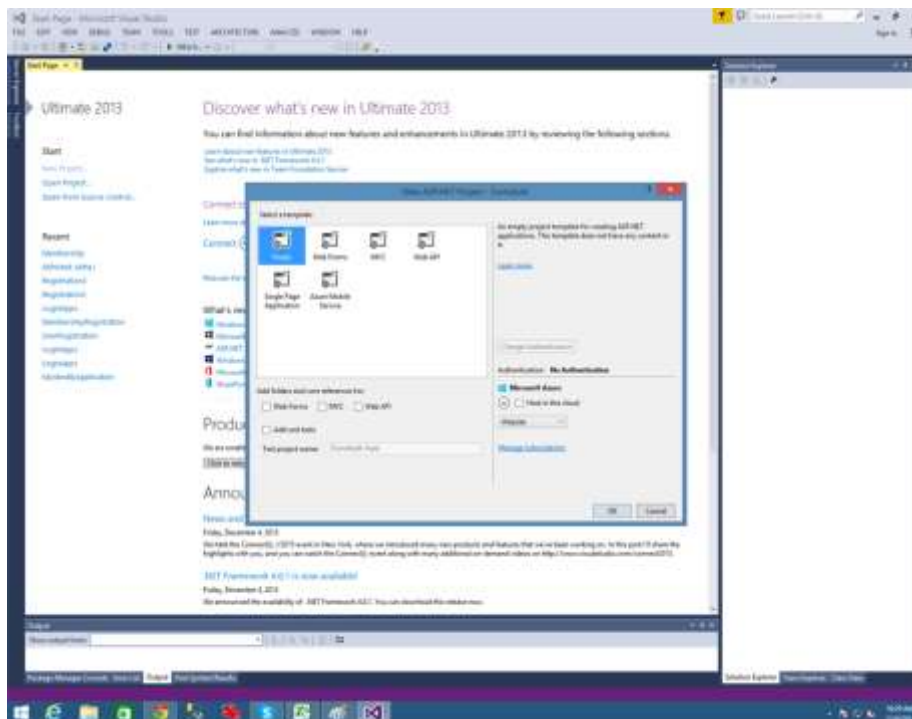
- Windows authentication
- Forms authentication
- Passport Authentication

These are the followings steps to use forms authentication in our web application.

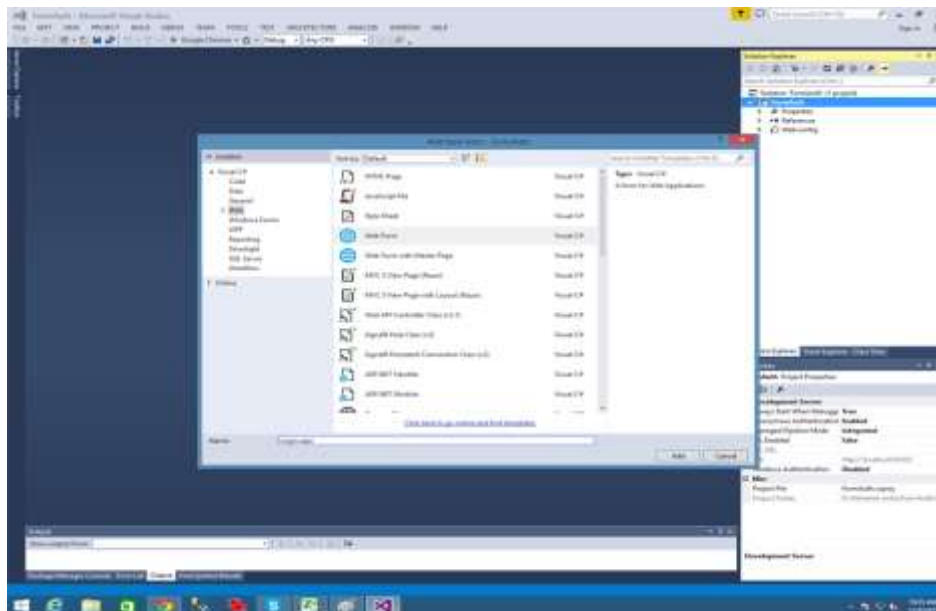
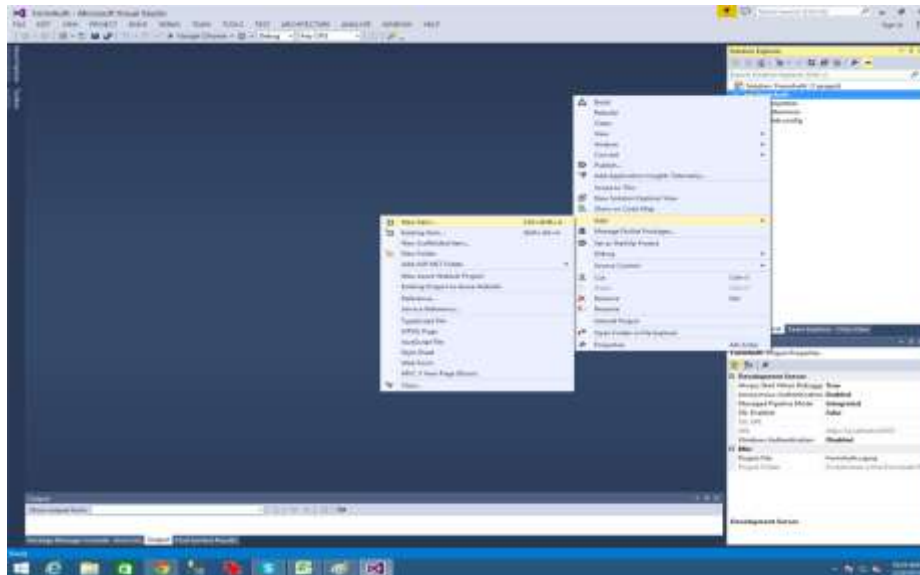
Step 1: Open Visual Studio then go to the File Menu where we click New, then Project and select the ASP.NET web application and assign the name of the application in pop up menu.



Step 2: After selecting the web application select an empty template here.



Step 3: In our web application here we add two pages one login.aspx and another welcome.aspx.



Here we added two web form login.aspx and welcome.aspx.

Step 4: Now we set web.config file to implement the authentication of web application.

```
<configuration>
  <appSettings>
    <add key="ValidationSettings:UnobtrusiveValidationMode" value="None" />
  </appSettings>
  <system.web>
    <compilation debug="true" targetFramework="4.5" />
    <authentication mode="Forms">
      <forms loginUrl="login.aspx" defaultUrl="welcome.aspx">
        <credentials passwordFormat="Clear">
          <user name="chhetra" password="chhetra@123"/>
          <user name="bishnu" password="bishnu@123" />
        </credentials>
      </forms>
    </authentication>
    <authorization>
      <deny users="?" />
    </authorization>
    <httpRuntime targetFramework="4.5" />
  </system.web>
</configuration>
```

Here we add Forms authentication mode and loginUrl is my login page where we create our login form and we also assign two users for authentication which is hard coded. So only two users by this name can authenticate our application page. If any other users try to access our web application then it will deny the users that are defined under the authorization tag.

Steps 5: Now we create a login form in *login.aspx* page. For that we will write the code for login form here,

```
<% @ Page Language="C#" AutoEventWireup="true" CodeBehind="Login.aspx.cs" Inherits="FormAuth.Login" %>
```

```
<!DOCTYPE html>
```

```
<html
```

```
  xmlns="http://www.w3.org/1999/xhtml">
```

```
  <head runat="server">
```

```
    <title></title>
```

```
  </head>
```

```
  <body>
```

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

```
      <h3>
```

```
        Login Page</h3>
```

```
      <table>
```

```
        <tr>
```

```
          <td>
```

```
            UserName:</td>
```

```
          <td>
```

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

```
          </td>
```

```
          <td>
```

```
            <asp:RequiredFieldValidator ID="RequiredFieldValidator1"
```

```
              ControlToValidate="UserName"
```

```
              Display="Dynamic"
```

ErrorMessage="Cannot be empty."

runat="server" />

</td>

</tr>

<tr>

<td>

Password:</td>

<td>

<asp:TextBox ID="UserPass" TextMode="Password"

runat="server" />

</td>

<td>

<asp:RequiredFieldValidator ID="RequiredFieldValidator2"

ControlToValidate="UserPass"

ErrorMessage="Cannot be empty."

runat="server" />

</td>

</tr>

<tr>

<td>

Remember me?</td>

<td>

<asp:CheckBox ID="chkboxPersist" runat="server" />

</td>

</tr>

```
</table>

<asp:Button ID="Submit1" OnClick="Login_Click" Text="Log In"
runat="server" />

<p>

    <asp:Label ID="Msg" ForeColor="red" runat="server" />

</p>

</form>

</body>

</html>
```

Step 6: Here we write the code for Login button in *login.aspx.cs* page.

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.UI;

using System.Web.UI.WebControls;

using System.Web.Security;

namespace FormAuth {

    public partial class Login: System.Web.UI.Page {

        protected void Page_Load(object sender, EventArgs e) {

        }

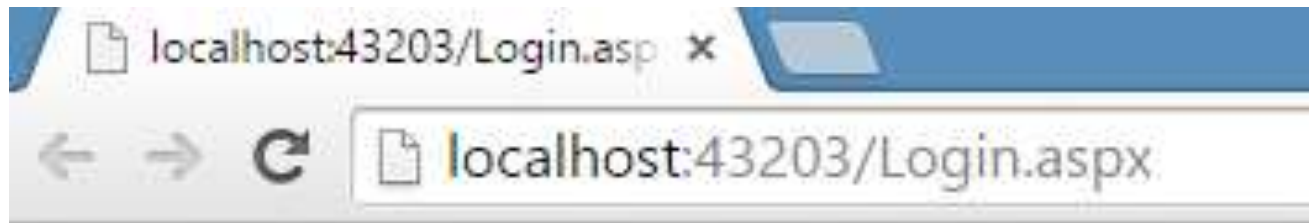
        protected void Login_Click(object sender, EventArgs e) {
```

```
        if (FormsAuthentication.Authenticate(UserName.Text, UserPass.Text)) {  
FormsAuthentication.RedirectFromLoginPage(UserName.Text, chkboxPersist.Checked);  
        } else {  
            Msg.Text = "Invalid User Name and/or Password";  
        }  
    }  
}  
}
```

Before writing the login button code user can directly redirect to the welcome page. If user type the Welcome page URL in browser, we need to prevent the direct access to the welcome page. For that we write the condition for authenticating the users on button click.

FormsAuthentication.Authenticate which takes two parameter username and password which authenticate the user and redirect the page after successful login.

Step 7: After writing the code of these pages now we are ready to execute the web application. To run the application we click Ctrl+F5 . After debugging the application our login page open in the browser.



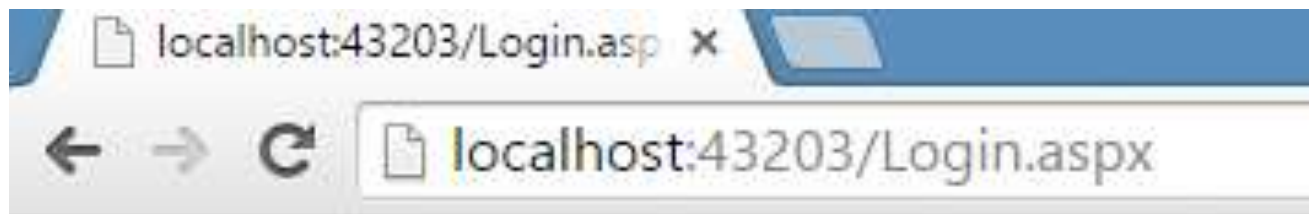
Login Page

UserName:

Password:

Remember me? ☐

Here we enter the valid Username and Password which we assigned in our web.config file. If we enter any invalid user then it will throw an exception.



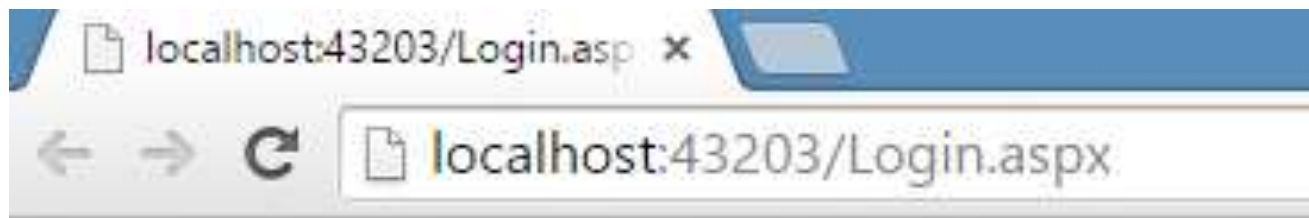
Login Page

UserName:	<input type="text" value="Surbhi"/>
Password:	<input type="password"/>
Remember me?	<input type="checkbox"/>
<input type="button" value="Log In"/>	

Invalid User Name and/or Password

Step 8: By entering the valid username and password our application will successfully authenticate the user which will redirect to the next welcome page after successfully login.

Here we checked the **Remember me** check box which means let's say we logged in and close the browser. We again open the browser with welcome page then we can access the authorized page without login. That means our browser cookies are enabled which store the user data in text file until the browser is cleared.



Login Page

UserName:

Password:

Remember me? ☒



!!!! WELCOME YOU ARE SUCCESSFULLY LOGED IN !!!!

Config File encryption

ASP.NET offers the possibility to encrypt sections in the web.config automatically. It seems it is not possible for WinForm applications to do that for the app.config. And this is true for a part: WinForms does not offer tools to configure it. But it can be done. It is all .NET. Using the Code

The app.config and web.config are divided into sections. The encrypting and decrypting operations are performed on sections and not on the file as a whole.

Developers can extend a configuration file by defining custom sections. This can be done by adding a section tag to the configSections element or the sectionGroup element like in the example below. The name attribute of section element specifies the name of the new section. The type attribute specifies the handler that processes the configuration section: it gets the data out of the section. As you can see in the example below, I implemented both scenarios.

```
<?xml version="1.0" encoding="utf-8" ?>

<configuration>

  <configSections>

    <section name="Vault"

      type="System.Configuration.NameValueSectionHandler" />

    <sectionGroup name="applicationSettings"

      type="System.Configuration.ApplicationSettingsGroup, System,

        Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" >

      <section name="EncryptConnStringsSection.My.MySettings"

        type="System.Configuration.ClientSettingsSection, System,

          Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"

          requirePermission="false" />

      </sectionGroup>

    </configSections>

    <connectionStrings>

      <add name="EncryptConnStringsSection.My.MySettings.testConn"

        connectionString="Data Source=someserver;Initial

          Catalog=ProjectX_Dev;Integrated Security=True" />

    </connectionStrings>
```

Now that I have explained how to create sections in the app.config, let's go on to show how to encrypt a section. It is really a simple operation. And once a section has been encrypted, you do not have to worry about decrypting it. The .NET Framework does it automatically for you. It is a transparent operation and works as if you did not encrypt the section.

The configuration namespace contains a class that represents a section. This class is called `ConfigurationSection`. A member of this class is the `ElementInformation` property. This property gets information about a section and it has the method `ProtectSection` defined on it. This method encrypts the section. Out of the box, there are two encryption algorithms supported via providers:

`DPAPIProtectedConfigurationProvider`

`RSAProtectedConfigurationProvider`