

What is ASP.NET

ASP.NET is a Web application framework developed by Microsoft to build dynamic data driven Web applications and Web services.

- ASP.NET and ADO.net are subsets of .NET framework.
- A framework is a collection of classes.
- ASP.NET is the successor to classic ASP (Active Server Pages)

What is a Web Application?

A web application is an application that is accessed by users using a web browser.

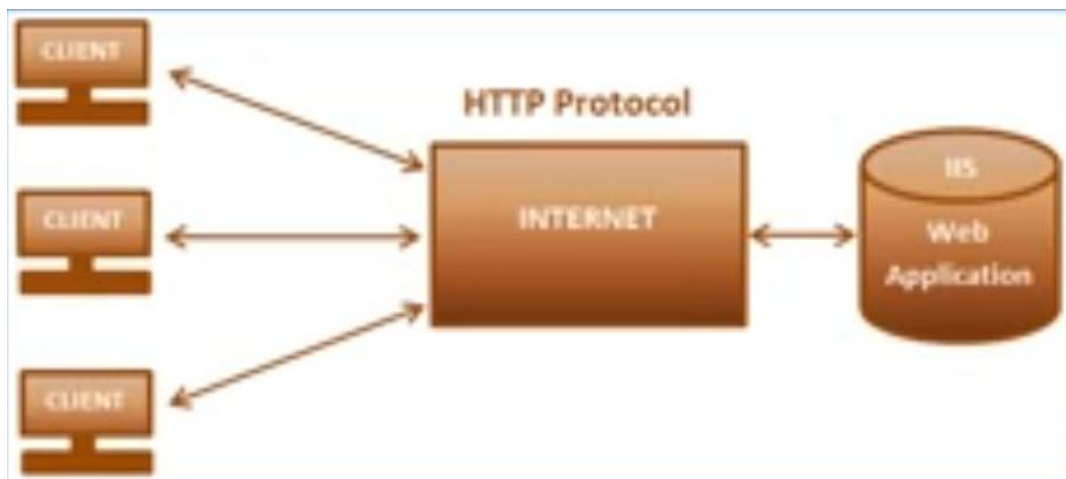
1. Microsoft Internet Explorer
2. Google Chrome
3. Mozilla Firefox
4. Apple Safari
5. Netscape Navigator
6. Edge
7. Opera

What other technologies can be used to build web applications

1. PHP
2. Java
3. CGI
4. Ruby on Rails
5. Perl

What are the advantages of Web applications?

1. Web Applications just need to be installed only on the web server, whereas desktop applications need to be installed on every computer, to access them.
2. Maintenance, support and patches are easier to provide.
3. Only a browser is required on the client machine to access a web application.
4. Accessible from anywhere, provided there is internet.
5. Cross Platform



1. Web applications work on client/server architecture

2. On the client all is needed is a browser, that can understand HTML
3. On the server side, the Web application runs under Microsoft Internet Information Services (IIS)

ViewState

Web Applications work on HTTP protocol. HTTP protocol is a stateless protocol, meaning it does not retain state between user requests.

```
public partial class WebForm1 : System.Web.UI.Page
{
    int ClicksCount = 0;
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            TextBox1.Text = "0";
        }
    }
    protected void Button1_Click(object sender, EventArgs e)
    {
        // This will always 1, as in Page_Load ClicksCount initiates to 0
        ClicksCount = ClicksCount + 1;
        TextBox1.Text = ClicksCount.ToString();
    }
}
```

In this code, after hitting Button_1, the TextBox1.Text will be 1, But no matter how many clicks it will remain 1. Because of the stateless nature of http protocol. Web pages are live only for a moment.

When first time the page loads into the browser, it goes to the web server and it is a GET request.

When the request goes to the web server, the web form which is technically a class, will create an instance. While building an application, it will convert into an IL, for web app a .dll, desktop app an .exe. The name of the assembly will be <ProjectName>.dll. When we call a web form, the assembly loaded into the memory, and will create an instance of the requested web page.

Also, whatever controls are there in a web page, they are also classes. So, the instances of the controls in the page also be created.

For the above code:

- It creates an instance of WebForm1.
- Creates an instance of ClicksCount and initiated to 0.
- Since it is a GET request, the Page_Load event will be fired.
- First time it is not a PostBack, so initializes TextBox1.Text to 0.
- Then generate the HTML and sent back to the client / browser.
- All the instances will be destroyed immediately, and the web server does not know about the Webform anymore.

Now when the button clicked, another request will go back to the web server. Since it is a button click, we are submitting or posting a request to the web server.

Now for the above code (First Click):

- Again, an instance of the `WebForm1`, `ClicksCount`, `TextBox1` and `Button1` will be created. Again, `ClicksCount` initiated to 0
- The `Page_Load` event will be fired.
- Now it is a `PostBack` event so

```
if (!IsPostBack)
{
    TextBox1.Text = "0";
}
```

This part will not be executed.

- `Button1_Click` event will be called, as we posted the page. And the code with in it will execute. For the first click here the `TextBox1.Text` will be 1.
- Then generate the HTML and sent back to the client / browser.
- All the instances will be destroyed immediately, and the web server does not know about the Webform anymore.

Now for the above code (second and onwards Click):

- Again, a new instance of the `WebForm1`, `ClicksCount`, `TextBox1` and `Button1` will be created. Again, `ClicksCount` initiated to 0
- The `Page_Load` event will be fired.
- `Button1_Click` event will be called, as we posted the page. And the code with in it will execute. For the first click here the `TextBox1.Text` will be 1.
- All the instances will be destroyed immediately, and the web server does not know about the Webform anymore.

Web forms live for barely a moment. When a request is received

1. An Instance of the requested webform is created
2. Events Processed
3. Generates the HTML & posted to the client
4. The webform is immediately destroyed

All the webforms, controls are Classes and they have objects.

To correct this problem, we need to use `ViewState`.

```
protected void Button2_Click(object sender, EventArgs e)
{
    //introduction to ViewState
    if (ViewState["Clicks"] != null)
    {
        ClicksCount = (int)ViewState["Clicks"] + 1;
    }
    TextBox1.Text = ClicksCount.ToString();
    ViewState["Clicks"] = ClicksCount;
}
```

Click the Button now, and the value gets incremented every time we click. So how is this possible now.?

It's possible because, we are using the ViewState["Clicks"] variable to preserve the data between requests. The ViewState data, travels with every request and response between the client and the web server.

In a ViewState object anything can be stored, simple, complex objects, date time anything. So, sometimes need to cast.

In page source a hidden control with id `__VIEWSTATE` and value of some random characters will generate. The random characters hold the value of the ViewState, which traverse through client and server.

In this code After first Button clicks:

- An instance of the WebForm1, ClicksCount, TextBox1 and Button2 will be created. Again, ClicksCount initiated to 0.
- Again, it is a PostBack event so

```
if (!IsPostBack)
{
    TextBox1.Text = "0";
}
```

This part will not be executed.

- Button2_Click event will be called, as we posted the page.
- Now ViewState["Clicks"] for the first click is null. So, the TextBox1.Text will be still 0.
- All the instances will be destroyed immediately, and the web server does not know about the Webform anymore.
- But the ClicksCount now stored into the ViewState. The value will now travel between Client and server as a hidden ViewState field.

After second (and onwards) Click:

- An instance of the WebForm1, ClicksCount, TextBox1 and Button2 will be created. Again, ClicksCount initiated to 0.
- Again, it is a PostBack event so

```
if (!IsPostBack)
{
    TextBox1.Text = "0";
}
```

This part will not be executed.

- Button2_Click event will be called, as we posted the page.
- Now ViewState["Clicks"] for the second click is not null and assigned a value 0 from the previous click. So now the ClicksCount will be incremented by 1.
- So, the TextBox1.Text will be 1.
- All the instances will be destroyed immediately, and the web server does not know about the Webform anymore.
- Then ViewState["Clicks"] will initiate to ClicksCount i.e. 1 now.

This happens in the Page_Init stage, which is called before the Page_Load event.

ASP.NET Server Controls & ViewState

```
protected void Button3_Click(object sender, EventArgs e)
{
    int ClicksCount = Convert.ToInt32(TextBox1.Text) + 1;
    TextBox1.Text= ClicksCount.ToString();
}
```

Upon clicking the Button, the value gets incremented correctly as expected. This is possible because, TextBox1 is an asp.net server control, that uses ViewState internally, to preserve data across PostBacks.

Because Web forms have very short lifetimes, ASP.NET takes special steps to preserve the data entered in the controls on a Web form. Data entered in controls is sent with each request and restored to controls in Page_Init. The data in these controls is then available in the Page_Load(), Button3_Click, and many more events, that occur after Page_Init() event.

- **An instance of the WebForm1, ClicksCount, TextBox1 and Button3 will be created.**
- **It initializes the TextBox1.Text to 0.**
- **Send it to the client and all the instances will be destroyed immediately.**
- **TextBox1.Text will not be destroyed as every server control uses ViewState internally, to preserve data across PostBacks. It will have the value of 0.**

In this code After first Button click:

- **Now this is again a new request.**
- **An instance of the WebForm1, ClicksCount, TextBox1 and Button3 will be created.**
- **Since it is a PostBack event so**

```
if (!IsPostBack)
{
    TextBox1.Text = "0";
}
```

This part will not be executed.

- **ClicksCount will be incremented to 1 as TextBox1.Text is 0 and the value was preserved during the PostBack.**
- **Now TextBox1.Text will be 1.**
- **Send it to the client and all the instances will be destroyed immediately.**
- **TextBox1.Text will not be destroyed as every server control uses ViewState internally, to preserve data across PostBacks. . It will have the value of 1.**

Comparison of codes

```
int ClicksCount = 0;

protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        TextBox1.Text = "0";
    }
}
```

Type	Code
No View State	<pre>protected void Button1_Click(object sender, EventArgs e) { ClicksCount = ClicksCount + 1; TextBox1.Text = ClicksCount.ToString(); }</pre>
View State	<pre>protected void Button2_Click(object sender, EventArgs e) { if (ViewState["Clicks"] != null) { ClicksCount = (int)ViewState["Clicks"] + 1; } TextBox1.Text = ClicksCount.ToString(); ViewState["Clicks"] = ClicksCount; }</pre>
Internal Textbox ViewState	<pre>protected void Button3_Click(object sender, EventArgs e) { int ClicksCount = Convert.ToInt32(TextBox1.Text) + 1; TextBox1.Text= ClicksCount.ToString(); }</pre>

Difference Between Server Controls and HTML Controls

ASP.NET server controls retain state.

HTML controls, do not retain state across post backs

An HTML control can be converted in ASP.NET server control, by adding `runat="server"` attribute in the HTML source as shown below

```
<input id="Text1" runat="server" type="text" />
```

ViewState data is serialized into base64-encoded strings, and is stored in Hidden input field.

```
<input type="hidden" name="__VIEWSTATE"
id="__VIEWSTATE"
value="PP06Jfyo8U8D4eQkz/Lo2Ynn4BiDXgjjYWfMmF+6Tu7u2kDV
zMLxj8OKHtP9eTGL1tTCaxVrCqvz+voA+ujysrY3FtLMV9Dgz+WQ0Rj
Wx8s=" />
```

Events in Life Cycle of a Web Page

In a web application, events can occur at 3 levels

1. At the Application Level(Example: Application Start)
2. At the Page Level (Example: Page Load)
3. At the Control Level (Example: Button Click)

ViewState variables are used to preserve data across page post back. By default, ViewState of one webform is not available in another webform.

Techniques to send data from one webform to another

1. QueryStrings
2. Cookies
3. Session State
4. Application State

Session state variables are available across all pages, but **only for a given single session**. Session variables are like single-user global data. Only the current session has access to its Session state.

Application State variables are available **across all pages and across all sessions**. **Application State** variables are like multi-user global data. All sessions can read and write Application State variables.

Application Level Events

In an ASP.NET web application, Global.asax file contains the application level events. In general, Application events are used to initialize data that needs to be available to all the current sessions of the application. Whereas Session events are used to initialize data that needs to be available only for a given individual session, but not between multiple sessions.

```
protected void Application_Start(object sender, EventArgs e)
{
    //Code that runs on Application Startup.
    //This event gets fired if the application is not already running.
    //The first request of a webform of an application reached IIS, and loads
the assembly into the memory.

    Application["TotalApplications"] = 0;
    Application["TotalUserSessions"] = 0;
    Application["TotalApplications"] = (int)Application["TotalApplications"] +
1;
}

protected void Session_Start(object sender, EventArgs e)
{
    ///Code that runs when new session has been created.
    ///when a new user logged into the system or connects to the application.
    ///A session is considered as a unique instance of a browser with a unique
session id.
```

```

        Application["TotalUserSessions"] = (int)Application["TotalUserSessions"]
+ 1;
    }

    protected void Application_Error(object sender, EventArgs e)
    {
        //Code that runs when an unhandled error occurs.
        //Use to handle the exception and log into a database or event viewer.
    }

    protected void Session_End(object sender, EventArgs e)
    {
        ///Code that runs when new session has been ends or a session timeout.
        ///This event raised only when the sessionstate mode is set to InProc in
web.config file.
        ///If it set to StateServer or anything else, this won't work.
    }

    protected void Application_End(object sender, EventArgs e)
    {
        //Code that runs on Application shutdown.
        //There are no more active sessions.
        //The last active session has been expired.

        Application["TotalUserSessions"] = (int)Application["TotalUserSessions"]
- 1;
    }

```

What is a Session

A session is a unique instance of the browser. A single user can have multiple sessions, by visiting an application, with multiple instances of the browser running with a different session-id on his machine.

How to get a new session-id and force the Session_Start() event to execute?

1. Close the existing browser window and then open a new instance of the browser
2. Open a new instance of a different browser
3. Use Cookie-less Sessions

```
<sessionState mode="InProc" cookieless="true"></sessionState>
```

Doing this the url will have now a unique session id (some random encoded characters).

Difference

ViewState:

1. ViewState of a webform is available only within that webform
2. ViewState is stored on the page in a hidden field called _ViewState. Because of this, the ViewState, will be lost, if navigated away from the page, or if the browser is closed.
3. ViewState is used by all asp.net controls to retain their state acrossPostBack

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

```



```

        if (!IsPostBack)
        {
            if (ViewState["Clicks"] is null)
            {
                ViewState["Clicks"] = 0;
            }
            TextBox1.Text = ViewState["Clicks"].ToString();
        }
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        int ClicksCount = (int)ViewState["Clicks"] + 1;
        TextBox1.Text = ClicksCount.ToString();
        ViewState["Clicks"] = ClicksCount;
    }
}

```

Session State:

1. Session state variables are available across all pages, but only for a given single session. Session variables are like single-user global data.
2. Session state variables are stored on the web server.
3. Session state variables are cleared, when the user session times out. The default is 20 minutes. This is configurable in web.config

```

protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        if (Session["Clicks"] is null)
        {
            Session ["Clicks"] = 0;
        }
        TextBox1.Text = Session ["Clicks"].ToString();
    }
}

protected void Button1_Click(object sender, EventArgs e)
{
    int ClicksCount = (int) Session ["Clicks"] + 1;
    TextBox1.Text = ClicksCount.ToString();
    Session ["Clicks"] = ClicksCount;
}

```

Application State:

1. Application State variables are available across all pages and across all sessions. Application State variables are like multi-user global data.
2. Application State variables are stored on the web server.
3. Application State variables are cleared, when the process hosting the application is restarted

```

protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {

```

```

        if (Application["Clicks"] is null)
        {
            Application ["Clicks"] = 0;
        }
        TextBox1.Text = Application ["Clicks"].ToString();
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        int ClicksCount = (int) Application ["Clicks"] + 1;
        TextBox1.Text = ClicksCount.ToString();
        Application ["Clicks"] = ClicksCount;
    }
}

```

Page Life Cycles Event

In a web application, events can occur at 3 levels

1. At the Application Level(Example: Application_Start etc in Global.asax)
2. At the Page Level (Example: Page_Load, Page_init etc)
3. At the Control Level (Example: Button_Click etc)

Web applications work on a stateless protocol. Every time a request is made for a webform, the following sequence of events occur.

1. Web Application creates an instance of the requested webform.
2. Processes the events of the webform.
3. Generated the HTML, and sends the HTML back to the requested client.
4. The webform gets destroyed and removed from the memory

The following are some of the commonly used events in the life cycle of an asp.net webform. These events are shown in order of occurrence, except for, **Error event**, which occurs only if there is an unhandled exception.

Event Name	Description
Page_PreInit	As the name suggests, this event happens just before page initialization event starts. IsPostBack , IsCallback and IsCrossPagePostBack properties are set at this stage. This event allows us to set the master page and theme of a web application dynamically. PreInit is extensively used when working with dynamic controls.
Page_Init	Page_Init , event occurs after the Page_PreInit event, of all the individual controls on the webform. Use this event to read or initialize control properties. The server controls are loaded and initialized from the Web form's view state. ViewState restoration happened.
Page_InitComplete	As the name says, this event gets raised immediately after page initialization.
Page_PreLoad	Happens just before the Page Load event.

Page_Load	Page Load event, occurs before the load event of all the individual controls on that webform.
Control Events	After the Page load event, the control events like button's click, dropdownlist's selected index changed events are raised.
Page_LoadComplete	This event is raised after the control events are handled.
Page_PreRender	This event is raised just before the rendering stage of the page. (Rendering meaning sending the HTML to browser)
Page_PreRenderComplete	Raised immediately after the PreRender event.
Page_Unload	Raised for each control and then for the page. At this stage the page is, unloaded from memory.
Page_Error	This event occurs only if there is an unhandled exception and log into DB or Event Viewer

Page_Preload
 Page_Init
 Page_InitComplete
 Page_Preload
 Page_Load
 Control Events
 Page_LoadComplete
 Page_PreRender
 Page_PreRenderComplete
 Page_Unload

Server Control Events

1. At the application level (Example-Session_Start event in Global.asax)
2. At the Page or web form level (Example-Page_Load)
3. At the control level (Example-Selected Index changed event of a DropDownList)

ASP.NET server controls, such as TextBox, Button, and DropDownList has their own events. We have a set of asp.net validation controls, that has validation events. The events that these controls expose, can be broadly divided into 3 categories.

PostBack events

These events submit the Web page, immediately to the server for processing. **Click event of a button control is an example** for PostBack event.

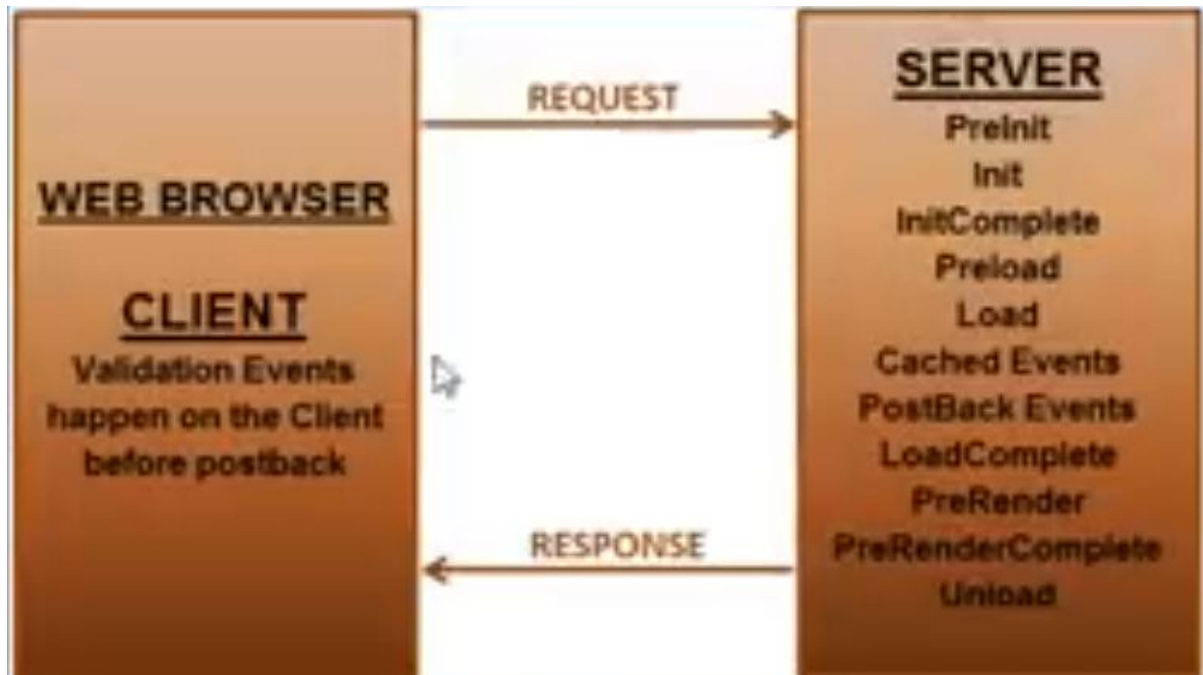
Cached events

These events are saved in the page's view state to be processed when a PostBack event occurs. **TextChanged** event of TextBox control, and **SelectedIndexChanged** event of a **DropDownList** control are examples of cached events. **Cached events can be converted into PostBack events, by setting the AutoPostBack property of the control to true.**

Validation events

These events occur on the client, before the page is posted back to the server. All validation controls use these type of events

The control events are processed after the **Page_Load** event. The picture below depicts the same. Among the control events, Cached events happen before PostBack events.



Page_PreInit
Page_Init
Page_InitComplete
Page_PreLoad
Page_Load

Button1_Clicked if **AutoPostBack="true"** then Text changed will be replaced by Button1_Click

else

TextBox1_TextChanged
Button1_Click

Page_LoadComplete
Page_PreRender
Page_PreRenderComplete

IsPostBack

IsPostBack is a Page level property, that can be used to determine whether the page is being loaded in response to a client PostBack, or if it is being loaded and accessed for the first time. This is a **Page** level event, and in the definition of Page a bool property

```
public bool IsPostBack { get; }
```

Now,

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)//or if(!Page.IsPostBack)
    {
        Response.Write("Page Loaded First Time");
    }
    LoadCity(); //This will duplicate the list items in drop down for every
click. So, should be in !IsPostBack
}

protected void Button1_Click(object sender, EventArgs e)
{
}

public void LoadCity()
{
    ListItem li = new ListItem("London");
    DropDownList1.Items.Add(li);
}
```

Now run the application. Look at the City DropDownList. The cities, (London, Sydney and Mumbai) are correctly shown as expected. Just click the button once. Notice, that the city names in the DropDownList are duplicated. So, every time clicking the button, the city names are again added to the DropDownList. (The commented `// LoadCity();` part)

What is causing duplication

We know that all ASP.NET server controls retain their state across PostBack. These controls make use of ViewState. So, the first time, when the webform loads, the cities get correctly added to the DropDownList and sent back to the client.

Now, when the client clicks the button control, and the webform is posted back to the server for processing. During the Page initialization, ViewState restoration happens. During this stage, the city names are retrieved from the ViewState and added to the DropDownList.

PageLoad event happens later in the life cycle of the webform. During page load we are again adding another set of cities. Hence, the duplication.

Solve DropDownList items duplication

There are several ways to solve this. One of the best ways to do this, is to use IsPostBack property. **(BEST APPROCH)**

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

```

        if (!IsPostBack)//or if(!Page.IsPostBack)
        {
            Response.Write("Page Loaded First Time");
            LoadCity();
        }
    }

```

Another way to solve, this problem is to simply disable the ViewState of the DropDownList control. Issues with this approach,

1. **DropDownList list, does not remember the selection acrossPostBack.**
2. **DropDownList events may not work correctly as expected.**
3. **Cached events will not work, so the DropDownList will not remember the selection, whilePostBack**

```

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

```

Another way to solve this, is to clear all the DropDownList items, before calling LoadCity() method. **But this not efficient from a performance perspective.**

```

protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)//or if(!Page.IsPostBack)
    {
        DropDownList1.items.clear();
        LoadCity();
    }
}

```

IIS

What is a web server?

In simple terms, a web server, is a software, that is used to deliver web pages to clients using the Hypertext Transfer Protocol (HTTP). For example, IIS is a web server that can be used to run asp.net web applications.

Is IIS needed to develop and test asp.net web applications?

No, Visual Studio ships with a built-in web server. If want to just build and test web applications on local machine, no need of IIS. Built-in web server will not serve requests to another computer. By default, visual studio uses the built-in web server.

How to check if IIS is installed?

1. Click on the windows Start button
2. Type INETMGR in the Run window.
3. Click OK.
4. If the IIS manager window is there, it is installed, otherwise not installed.

How to install IIS?

1. Click on the start button and select Control Panel and then click on Programs
2. Click on Turn Windows features on or Off, under Programs and features option
3. In the Windows features, Select Internet Information Services and related services

To configure a virtual directory in IIS to run asp.net web applications

1. In the IIS Manager window, double click on the IIS server name in the connections section.
2. Expand sites
3. Right click on Default Web Site, and Select Add Application.
4. Give an alias name. This is the name that can be used in the URL, when connecting to the web application.
5. Click the button next to the textbox under physical path. Select the physical web application folder.

A virtual directory can also be created from Visual Studio, on the project properties window.

1. Select Use Local IIS Web Server
2. Project URL will be populated automatically. The name of the virtual directory can be changed if wished.
3. Click Create Virtual Directory button.
4. After a few seconds the virtual directory was successfully created message will appear.
5. Click OK

TextBox Controls

The TextBox control is used to get the input from the user of the web application. An asp.net textbox has several properties, that we need to be aware of as a developer.

Properties of a TextBox control

1. **TextMode** Property - SingleLine, MultiLine and Password.

When set the TextMode to MultiLine, use Rows property to control the number of lines to display for a Multiline TextBox.

2. **Text** - Use this property to set or get the Text from the TextBox.
3. **MaxLength** - The maximum number of characters that a user can enter.
4. **ReadOnly** - Set this property to true if don't want the user to change the text in the TextBox.
5. **ToolTip** - The tooltip is displayed when the mouse is over the control.
6. **Columns** - Use this property to specify the width of the TextBox in characters
7. **Height** - Set the height
8. **Width** - Set the width

9. **AutoPostBack** - Automatically PostBack when text is changed.

Events of TextBox:

TextChanged-This event is fired, when the text is changed.

Methods of a TextBox:

Focus-Set input focus onto the control.

To view the properties of the TextBox, Right click on the control, and select Properties. In the properties window, also the events can be found that supported by the control.

All these properties can be set at the design time, or at runtime using code.

Radio Button Control

Radio Button control is used, when the user to select only one option from the available choices. For example, the gender of a person. A person can be Male or Female. He cannot be both. So, if the user has first selected Male, and if tries to select Female, the initial Male selection he made should automatically get de-selected. Another example, would be when the user to select his or her favourite colour.

In short, if provide the user with mutually exclusive options, then choose a Radio Button Control

Properties

1. **Checked**-This is a Boolean property, that is used to check if the button is checked or not.
2. **Text**-This is string property used to get or set the text associated with the radio button control
3. **TextAlign**-right or left. On which side of the radio button the text should appear
4. **AutoPostBack**-Set this property to true, if the webform to be posted immediately when the checked status of the radio button changes.
5. **Group Name**-By default, the individual radio button selections, are not mutually exclusive. If there is a group of radio buttons, and if want the selections among the group to be mutually exclusive, then use the same group name for all the radio button controls.

Events:

CheckedChanged - This event is fired when the checked status of the radio button control is changed.

CheckBox

CheckBox Control is used, when want the user to select more than one option from the available choices. For example, the education of a person. A person can have a graduate degree, post graduate degree and a doctorate. In this case the user selects all the 3 checkboxes. Whereas a person, may just have a graduate degree, in which case he only selects, the graduate checkbox. Another example, would be when the user to select the days of his availability.

In short, if want to provide the user with more than one option to select from, then choose a check box Control.

Properties:

1. **Checked**-This is a Boolean property, that is used to check if the check box is checked or not.
2. **Text**-This is a string property used to get or set the text associated with the check box control
3. **TextAlign**-right or left. On which side of the check box the text should appear
4. **AutoPostBack**-Set this property to true, if want the webform to be posted immediately when the checked status of the check box changes.

Methods:

Focus()-Just like TextBox, checkbox also supports, Focus() method. If want to set 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 check button control is changed.

Code (for StringBuilder)

StringBuilder to store the selections

```
<asp:CheckBox ID="CheckBox1" Text="Graduate" runat="server" />
<asp:CheckBox ID="CheckBox2" Text="Post Graduate" runat="server" />
<asp:CheckBox ID="CheckBox3" Text="Doctorate" runat="server" />

<asp:Button ID="btn" Text="Click" runat="server" OnClick="btn_Click" />
```

```
using System;
using System.Text; )    // for StringBuilder
```

```
namespace WebApplication1
{
    public partial class CheckBox : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }

        protected void btn_Click(object sender, EventArgs e)
        {
            StringBuilder userChoice = new StringBuilder();

            if (CheckBox1.Checked)
            {
                userChoice.Append(CheckBox1.Text);
            }
            if (CheckBox2.Checked)
            {
                userChoice.Append( CheckBox2.Text);
            }
            if (CheckBox3.Checked)
```

```

        {
            userChoice.Append( CheckBox3.Text);
        }
        Response.Write(userChoice);
    }
}
}

```

HyperLink

The ASP.NET Hyperlink control is used to create a link to another Web page.

Properties:

Text-The link text that will be shown to the user

Navigate URL-The URL of the page to which the user will be sent

ImageUrl-The URL of the image, that will be displayed for the link. If specify both the Text and ImageUrl, the image will be displayed instead of the text. If for some reason, the image is not unavailable, the text will be displayed.

Target-If target is not specified, the web page to which the hyperlink is linked, will be displayed in the same window. If set the Target to _blank, the web page will be opened in a new window.

Methods:

Focus()-Call this method to Set the input focus when the page loads.

Events:

No events specific to HyperLink control

Button, LinkButton & ImageButton

The Button, LinkButton and ImageButton controls in ASP.NET are used to post a page to the server.

1. **Button**- The Button control is used to display a push button. Use the Text property to change the Text on the Button control.
2. **LinkButton**- LinkButton displays the button like a HyperLink. Use the Text property to change the Link Text. But this redirected to the same page in general
3. **ImageButton**- ImageButton provides the flexibility of associating an Image with the button, using the ImageURL property.

Properties

1. CommandName
2. CommandArgument
3. CausesValidation
4. ValidationGroup
5. PostBackURL
6. **OnClientClick**- Generally used for JavaScript functions to execute in client side, and before server side PostBacks or before OnClick property, like alert(),confirm().

Command Event

ASP.NET button control exposes 2 events-Click and Command events. When the Button is clicked, both the events are raised. Button1_Click and Button1_Command Event.

Click event happens before the Command event.

Note: Eventhandlers can be associated to the events of a control in 2 ways.

1. Declaratively at design time in the HTML

```
<asp:Button ID="Button1" runat="server" Text="Button" OnClick="Button1_Click"
OnCommand="Button1_Command" />
```

```
protected void Button1_Click(object sender, EventArgs e)
{
    Response.Write("Button Clicked <br/>");
}

protected void Button1_Command(object sender, CommandEventArgs e)
{
    Response.Write("Button Command <br/>");
}
```

```
//Output on Button click
Button Clicked
Button Command
```

2. Programmatically using delegates

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

```
protected void Page_Load(object sender, EventArgs e)
{
    Button1.Click += new EventHandler(Button1_Click);
    Button1.Command += new CommandEventHandler(Button1_Command);
}
```

```
//Here EventHandler and CommandEventHandler are delegates. Instead declaring the
properties in .aspx, declared using delegates programmatically.
}
```

```
protected void Button1_Click(object sender, EventArgs e)
{
    Response.Write("Button Clicked <br/>");
}

protected void Button1_Command(object sender, CommandEventArgs e)
{
    Response.Write("Button Command <br/>");
}
```

```
//Output on Button click
Button Clicked
Button Command
```

When to use the Command event

If there are multiple button controls on a webform, and if need to programmatically determine which Button control is clicked, then can make use of Command event, along with CommandName and CommandArgument properties.

Command event, makes it possible to have a single event handler method responding to the click event of multiple buttons. The Command event, CommandName and CommandArgument properties are extremely useful when working with data-bound controls like Repeater, GridView, DataList. All the 3 button controls expose Command event, the CommandName and CommandArgument properties.

```
<asp:Button ID="Print" runat="server" Text="Print"
OnCommand="OnCommandButton_Click" CommandName="Print"/>
<asp:Button ID="Delete" runat="server" Text="Delete"
OnCommand="OnCommandButton_Click" CommandName="Delete"/>
<asp:Button ID="Top100" runat="server" Text="Top100"
OnCommand="OnCommandButton_Click" CommandName="Show" CommandArgument="Top10"/>
<asp:Button ID="Bottom10" runat="server" Text="Bottom10"
OnCommand="OnCommandButton_Click" CommandName="Show" CommandArgument="Bottom10"/>
<asp:Label ID="Output" runat="server"></asp:Label>
```

```
protected void OnCommandButton_Click(object sender, CommandEventArgs e)
{
    switch(e.CommandName)
    {
        case "Print":
            Output.Text = "Print Clicked";
            break;
        case "Delete":
            Output.Text = "Delete Clicked";
            break;
        case "Show":
            if(e.CommandArgument.ToString()== "Top10") {
                Output.Text = "Show Top 10 Clicked";
            }
            if (e.CommandArgument.ToString() == "Bottom10")
            {
                Output.Text = "Show Bottom 10 Clicked";
            }
            break;
        default:
            Output.Text = "Nothing Clicked";
            break;
    }
}
```

Here instead of using 4 button clicks , everything handled into a single command event.

DropDownList

1. DropDownList is a collection of ListItem objects.
2. The Listitems can be added to the DropDownList at the design time or at the runtime.
3. If want a specific ListItem to be selected in the DropDownList, set the **Selected** property of the ListItem object to **true**.
4. To hide a ListItem in the DropDownList, set the **Enabled** property to **False**.

5. If adding ListItem objects, to the DropDownList in the **Page_Load** event, make to do only when the page is loaded for the first time. Otherwise, every time, post the page back, by clicking a button, the list items will be added again causing duplication.

A DropDownList is a collection of ListItem objects. Along the same lines, the following controls are also a collection of ListItem objects. So, adding items to these controls is also very similar to DropDownList.

1. CheckBoxList
2. RadioButtonList
3. BulletedList
4. ListBox

Adding DropDownList Items using code behind

Binding from Design

```
<asp:DropDownList runat="server" ID="DropDownList3">
  <asp:ListItem Value="0" Text="SSC"></asp:ListItem>
  <asp:ListItem Value="1" Text="UPSC"></asp:ListItem>
  <asp:ListItem Value="2" Text="IPS"></asp:ListItem>
</asp:DropDownList>
```

Binding from Code Behind

```
protected void Page_Load(object sender, EventArgs e)
{
    ListItem l1 = new ListItem("Male", "M");
    ListItem l2 = new ListItem("Female", "F");
    ListItem l3 = new ListItem("Unknown", "U");
    ListItem l4 = new ListItem("Others", "O", false); //Setting enabled false,
i.e. Hidden

    if(!IsPostBack)
    {
        DropDownList1.Items.Add(l1);
        DropDownList1.Items.Add(l2);
        DropDownList1.Items.Add(l3);
        DropDownList1.Items.Add(l4);
    }
}
```

Binding from Database

```
<asp:DropDownList runat="server" ID="DropDownList2"></asp:DropDownList><br />

string connectionString = ConfigurationManager.AppSettings["DBConnectionString"];
using (OracleConnection sCon = new OracleConnection(connectionString))
{
    OracleCommand cmd = new OracleCommand("SELECT GCI_CITY_CD
CITY_CD,GCI_CITY_NAME CITY_NAME FROM GA_CITY_MASTER Order by GCI_CITY_CD", sCon);
    sCon.Open();
    DropDownList2.DataSource = cmd.ExecuteReader();
    DropDownList2.DataTextField = "CITY_NAME";
    DropDownList2.DataValueField = "CITY_CD";
    DropDownList2.DataBind();
}
}
```

```

<appSettings>
  <add key="DBConnectionString" value="Data Source=TMCTEST;User
ID=tmclive;pwd=tmclive;integrated security = no" />
</appSettings>

```

OR

```

<asp:DropDownList runat="server" DataTextField="CITY_NAME" DataValueField="CITY_CD"
ID="DropDownList2"></asp:DropDownList><br />

```

```

string connectionString =
ConfigurationManager.AppSettings["DBConnectionString"];
using (OracleConnection sCon = new OracleConnection(connectionString))
{
    OracleCommand cmd = new OracleCommand("SELECT GCI_CITY_CD
CITY_CD,GCI_CITY_NAME CITY_NAME FROM GA_CITY_MASTER Order by GCI_CITY_CD", sCon);
    sCon.Open();
    DropDownList2.DataSource = cmd.ExecuteReader();
    //DropDownList2.DataTextField = "CITY_NAME";
    //DropDownList2.DataValueField = "CITY_CD";
    DropDownList2.DataBind();
}

```

If **DataTextField** and **DataValueField** are not defined, then the DropDownList will populate the objects but not the values. The both fields should be declared either in design or in Code behind before **DataBind()**.

Binding from XML

ReadXml() method of the DataSet object can be used, to read the data from the XML file into a DataSet.

Server.MapPath() method return the physical path for a given virtual path.

To insert a ListItem at a specific location use the **Insert()** method.

```

<?xml version="1.0" encoding="utf-8" ?>
<Countries>
  <Country>
    <CountryId>101</CountryId>
    <CountryName>India</CountryName>
  </Country>
  <Country>
    <CountryId>102</CountryId>
    <CountryName>USA</CountryName>
  </Country>
  <Country>
    <CountryId>103</CountryId>
    <CountryName>Austria</CountryName>
  </Country>

```

```

<Country>
  <CountryId>104</CountryId>
  <CountryName>UK</CountryName>
</Country>
</Countries>

```

```

DataSet ds = new DataSet();
ds.ReadXml(Server.MapPath("~/contents/Country.xml"));
DropDownList4.DataSource = ds;
DropDownList4.DataTextField = "CountryName";
DropDownList4.DataValueField = "CountryId";
DropDownList4.DataBind();
DropDownList4.Items.Insert(0, "~Select~");

```

Inserting an element in nth Position

```

DropDownList1.Items.Insert(0, "~Select~");
DropDownList2.Items.Insert(0, "~Select~");
DropDownList3.Items.Insert(0, "~Select~");
DropDownList4.Items.Insert(0, "~Select~");

```

Retrieving Value and text

Selected Item Text: DropDownList1.SelectedItem.Text

Selected Item Value: DropDownList1.SelectedItem.Value

OR

Selected Item Value: DropDownList1.SelectedValue

Selected Item Index: DropDownList1.SelectedIndex

The SelectedIndex and SelectedValue properties of the DropDownList can also be used to have a list item selected in the DropDownList

Cascading DropDownList

```

<asp:DropDownList runat="server" ID="ddlCountry" AutoPostBack="true"
OnSelectedIndexChanged="ddlCountry_SelectedIndexChanged">

    </asp:DropDownList><br /><br /><br />
    <asp:DropDownList runat="server" ID="ddlState" AutoPostBack="true"
Enabled="false" OnSelectedIndexChanged="ddlState_SelectedIndexChanged">

        </asp:DropDownList><br /><br /><br />
        <asp:DropDownList runat="server" ID="ddlCity" AutoPostBack="true"
Enabled="false">

            </asp:DropDownList><br /><br /><br />

```

```

using System;
using System.Data;
using System.Data.OracleClient;
using System.Configuration;
using System.Web.UI.WebControls;

namespace ASPDotNET_Tutorial_for_Beginners
{

```

```

public partial class CascadingDropDownList : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            ddlCountry.DataSource = GetData("P_GET_COUNTRY", null);
            ddlCountry.DataValueField = "COUNTRY_CD";
            ddlCountry.DataTextField = "COUNTRY_DESC";
            ddlCountry.DataBind();
            ddlCountry.Items.Insert(0, "~Select Country~");
            ddlState.Items.Insert(0, "~Select State~");
            ddlCity.Items.Insert(0, "~Select City~");
        }
    }

    public DataSet GetData(String SPName, OracleParameter SPPParameter)
    {
        string connectionString =
ConfigurationManager.AppSettings["DBConnectionString"];
        OracleConnection sCon = new OracleConnection(connectionString);
        OracleDataAdapter da = new OracleDataAdapter(SPName, connectionString);
        da.SelectCommand.CommandType = CommandType.StoredProcedure;
        if (SPPParameter != null)
        {
            da.SelectCommand.Parameters.Add(SPPParameter);
        }
        OracleParameter refCursor = new OracleParameter("RC1", OracleType.Cursor);
        refCursor.Direction = ParameterDirection.Output;
        da.SelectCommand.Parameters.Add(refCursor);
        DataSet ds = new DataSet();
        da.Fill(ds);
        return ds;
    }

    protected void ddlCountry_SelectedIndexChanged(object sender, EventArgs e)
    {
        if (ddlCountry.SelectedValue.ToString() != "~Select Country~")
        {
            ddlState.Enabled = true;
            ddlCity.Enabled = false;
            ddlCity.ClearSelection();
            ddlCity.Items.Insert(0, "~Select City~");
            OracleParameter parameter = new OracleParameter("P_COUNTRY_CD",
ddlCountry.SelectedValue.ToString());
            ddlState.DataSource = GetData("P_GET_STATE", parameter);
            ddlState.DataTextField = "state_desc";
            ddlState.DataValueField = "State_cd";
            ddlState.DataBind();
            ddlState.Items.Insert(0, "~Select State~");
        }
        else
        {
            ddlState.Items.Insert(0, "~Select State~");
            ddlCity.Items.Insert(0, "~Select City~");
            ddlState.Enabled = false;
            ddlCity.Enabled = false;
        }
    }
}

```



```

protected void ddlState_SelectedIndexChanged(object sender, EventArgs e)
{
    if (ddlState.SelectedValue.ToString() != "~Select State~")
    {
        ddlCity.Enabled = true;
        OracleParameter parameter = new OracleParameter("P_STATE_CD",
ddlState.SelectedValue.ToString());
        ddlCity.DataSource = GetData("P_GET_CITY", parameter);
        ddlCity.DataTextField = "CITY_DESC";
        ddlCity.DataValueField = "CITY_CD";
        ddlCity.DataBind();

    }
    else
    {
        ddlCity.Enabled = false;
        ddlCity.ClearSelection();
        ddlCity.Items.Insert(0, "~Select City~");
    }
}
}
}

```

CheckBoxList

1. CheckBoxList is collection of ListItem objects.
2. Items can be added to the CheckBoxList in the HTML source or in the code behind file
3. CheckBoxList can be bound to a database table or an xml file

DropDownList is generally used, when want to present the user with multiple choices, from which to select only one option. Whereas if want the user to select more than one option, then a CheckBoxList control can be used

Useful Properties

1. RepeatDirection : By default, Vertical, can be changed to horizontal
2. RepeatColumns : The CheckBoxList items will be distributed into number columns/rows
3. Enabled: If set to false, then that item will read only..
4. SelectedIndex
5. SelectedValue
6. SelectedItem

```

if (checkboxListEducation.SelectedItem is null)
{
    Response.Write(checkboxListEducation.SelectedItem.Text);
}

```

Check for null, when using SelectedItem property of a CheckBoxList control.

```

<asp:CheckBoxList ID="CheckBoxList1" AutoPostBack="true" runat="server"
RepeatDirection="Horizontal"
OnSelectedIndexChanged="CheckBoxList1_SelectedIndexChanged">
    <asp:ListItem Value="SSC">Madhyamik</asp:ListItem>
    <asp:ListItem Value="HS">Higher Secondary</asp:ListItem>
    <asp:ListItem Value="UG">Under Graduate</asp:ListItem>
    <asp:ListItem Value="G">Graduate</asp:ListItem>
    <asp:ListItem Value="PG">Post Graduate</asp:ListItem>
</asp:CheckBoxList><br />
<asp:Button text="Click" runat="server" ID="btnClick"
OnClick="btnClick_Click"/>

```

```

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

    }

    protected void btnClick_Click(object sender, EventArgs e)
    {
        if (CheckBoxList1.SelectedItem is null)
        {
            Response.Write("Nothing Selected.");
        }
        else
        {
            foreach (ListItem li in CheckBoxList1.Items)
            {
                if (li.Selected)
                {
                    Response.Write("Value is - " + li.Value + "<br/>");
                    Response.Write("Text is - " + li.Text + "<br/>");
                    Response.Write("Index is - " + CheckBoxList1.Items.IndexOf(li)
+ "<br/>");
                }
            }
        }
    }

    protected void CheckBoxList1_SelectedIndexChanged(object sender, EventArgs e)
    {
        if (CheckBoxList1.SelectedItem is null)
        {
            Response.Write("Nothing Selected.");
        }
        else
        {
            foreach (ListItem li in CheckBoxList1.Items)
            {
                if (li.Selected)
                {
                    Response.Write("Value is - " + li.Value + "<br/>");
                    Response.Write("Text is - " + li.Text + "<br/>");
                    Response.Write("Index is - " + CheckBoxList1.Items.IndexOf(li)
+ "<br/>");
                }
            }
        }
    }
}

```

```

<asp:CheckBoxList ID="CheckBoxList1" AutoPostBack="true" runat="server"
RepeatDirection="Horizontal"
OnSelectedIndexChanged="CheckBoxList1_SelectedIndexChanged">
    <asp:ListItem Value="SSC">Madhyamik</asp:ListItem>

```

```

        <asp:ListItem Value="HS">Higher Secondary</asp:ListItem>
        <asp:ListItem Value="UG">Under Graduate</asp:ListItem>
        <asp:ListItem Value="G">Graduate</asp:ListItem>
        <asp:ListItem Value="PG">Post Graduate</asp:ListItem>
    </asp:CheckBoxList><br />
    <asp:Button text="Print" runat="server" ID="btnClick"
OnCommand="btnClick_Click" CommandName="Print"/>
    <asp:Button text="Select All" runat="server" ID="btnSelectAll"
OnCommand="btnClick_Click" CommandName="SelectAll"/>
    <asp:Button text="De Select All" runat="server" ID="btnDeselectAll"
OnCommand="btnClick_Click" CommandName="DeSelectAll"/>

```

```

protected void btnClick_Click(object sender, CommandEventArgs e)
{
    switch (e.CommandName)
    {
        case "SelectAll":
            foreach (ListItem li in CheckBoxList1.Items)
            {
                li.Selected = true;
            }
            break;

        case "DeSelectAll":
            foreach (ListItem li in CheckBoxList1.Items)
            {
                li.Selected = false;
            }
            break;

        case "Print":
            if (CheckBoxList1.SelectedItem is null)
            {
                Response.Write("Nothing Selected.");
            }
            else
            {
                foreach (ListItem li in CheckBoxList1.Items)
                {
                    if (li.Selected)
                    {
                        Response.Write("Value is - " + li.Value +
"
<br/>");
                        Response.Write("Text is - " + li.Text + "
<br/>");
                        Response.Write("Index is - " +
CheckBoxList1.Items.IndexOf(li) + "
<br/>");
                    }
                }
            }
            break;
        default:
            break;
    }
}

```

ListBox

Just like DropDownList and CheckBoxList

1. ListBox is also a collection of ListItem objects.
2. Items can be added to the ListBox in the HTML source or in the code behind file
3. ListBox can be bound to a database table or an xml file

Useful ListBox Properties

1. Rows : Set to show number of rows of the ListBox.
2. SelectionMode: By default, single. In that case only a value can be selected. Resetting it to multiple, more than one items can be selected.

CheckBoxList to ListBox

```
<asp:CheckBoxList ID="CheckBoxList1" AutoPostBack="true" runat="server"
RepeatDirection="Horizontal"
OnSelectedIndexChanged="CheckBoxList1_SelectedIndexChanged">
    <asp:ListItem Value="SSC">Madhyamik</asp:ListItem>
    <asp:ListItem Value="HS">Higher Secondary</asp:ListItem>
    <asp:ListItem Value="UG">Under Graduate</asp:ListItem>
    <asp:ListItem Value="G">Graduate</asp:ListItem>
    <asp:ListItem Value="PG">Post Graduate</asp:ListItem>
</asp:CheckBoxList><br />
<asp:ListBox ID="ListBox1" AutoPostBack="true" runat="server"
SelectionMode="Multiple" Height="180px" Width="260px"></asp:ListBox><br />
<asp:Label ID="Label1" runat="server"></asp:Label>
```

```
protected void CheckBoxList1_SelectedIndexChanged(object sender, EventArgs e)
{
    ListBox1.Items.Clear();/// Not clearing, will add repeated items in the
ListBox
    foreach (ListItem li in CheckBoxList1.Items)
    {
        if (li.Selected)
        {
            /// ListBox1.Items.Add(li);      // Not including .Text Property,
will make the ListBox items selected.
            ListBox1.Items.Add(li.Text);
        }
    }
    if (ListBox1.Items.Count == 0)
    {
        Label1.Text = "0 Items Selected";
        Label1.ForeColor = System.Drawing.Color.Red;
        Label1.Font.Bold = true;
    }
    else
    {
        Label1.Text = ListBox1.Items.Count + " Items Selected";
        Label1.ForeColor = System.Drawing.Color.Black;
        Label1.Font.Bold = true;
    }
}
```

RadioButtonList

In ASP.net there are several controls like

1. DropDownList
2. CheckBoxList
3. BulletedList
4. ListBox
5. RadioButtonList

Just like every other list control

1. RadioButtonList is also a collection of ListItem objects
2. Items can be added to the RadioButtonList in the HTML or in the code behind file
3. RadioButtonList like any other list control supports data binding

For example, RadioButtonList can be bound to a database table or an xml file

CheckBoxList is generally used, when want to present the user with multiple choices, from which to select one or more options.

Whereas, if want the user to select only one option, then a RadioButtonList control can be used, i.e. RadioButtonList is commonly used to present mutually exclusive choices.

Properties

1. RepeatDirection
2. RepeatColumns
3. RepeatLayout
4. SelectedItem
5. Selectedindex

BulletedList

Just like every other list control

1. Bulleted List is also a collection of ListItem objects
2. Items can be added to the Bulleted List in the HTML or in the code behind file
3. BulletedList like any other list control supports data binding.

For example, Bulleted List can be bound to a database table or an xml file

Properties:

1. BulletStyle
2. FirstBulletNumber
3. DisplayMode-Text, HyperLink or LinkButton. The default is Text
4. Target-Used when DisplayMode is HyperLink

```
<asp:BulletedList ID="RadioButtonList1" runat="server"
RepeatDirection="Horizontal" BulletStyle="UpperAlpha" DisplayMode="LinkButton"
OnClick="RadioButtonList1_Click" Target="_blank">
    <asp:ListItem Value="https://www.google.co.in">Google</asp:ListItem>
    <asp:ListItem Value="https://www.msn.com/en-in">MSN</asp:ListItem>
</asp:BulletedList>
```

```
protected void RadioButtonList1_Click(object sender, BulletedListEventArgs e)
{
    ListItem li = RadioButtonList1.Items[e.Index];
    Response.Write("Value is - " + li.Value + "<br/>");
    Response.Write("Text is - " + li.Text + "<br/>");
    Response.Write("Index is - " + e.Index.ToString() + "<br/>");
}
```

ListControl

In ASP.NET there are several list controls, like

1. DropDownList
2. CheckBoxList
3. RadioButtonList
4. ListBox
5. BulletedList

All these controls inherit from ListControl class.

1. Collection of ListItem objects
2. ListItems can be added in the HTML source or in the code behind file
3. Supports Databinding

Since all the list controls inherit from List Control class, AddListItems() method can be used to add ListItems to any list control. A parent class reference variable can point to a derived class object. This fact allows us to pass any list control into the AddListItems() method as a parameter.

```
<asp:CheckBoxList ID="CheckBoxList2" runat="server">
</asp:CheckBoxList>
<asp:DropDownList ID="DropDownList2" runat="server">
</asp:DropDownList>
<asp:RadioButtonList ID="RadioButtonList2" runat="server">
</asp:RadioButtonList>
<asp:ListBox ID="ListBox2" SelectionMode="Multiple" runat="server"></asp:ListBox>
<asp:BulletedList ID="BulletedList2" runat="server" DisplayMode="LinkButton">
</asp:BulletedList>
<asp:Button runat="server" Text="Click" OnClick="Unnamed1_Click"/>
```

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        AddListItems(CheckBoxList2);
        AddListItems(DropDownList2);
        AddListItems(RadioButtonList2);
        AddListItems(ListBox2);
        AddListItems(BulletedList2);
    }
}

public void AddListItems(ListControl listControls)
{
}
```

```

        ListItem L1 = new ListItem("Item 1", "1");
        ListItem L2 = new ListItem("Item 2", "2");
        ListItem L3 = new ListItem("Item 3", "3");

        listControls.Items.Add(L1);
        listControls.Items.Add(L2);
        listControls.Items.Add(L3);
    }

```

Retrieving

ListBox (If SelectionMode-Multiple) and CheckBoxList allows user to select multiple items. So, to retrieve all the selected ListItem's Text, Value and Index use a foreach loop. Reusable method that can be used with any control that derives from ListControl class, but works best with controls that allows multiple selections.

```

public void RetrieveMultipleData(ListControl listControls)
{
    foreach (ListItem li in listControls.Items)
    {
        if (li.Selected)
        {
            Response.Write(listControls + "<br/>");
            Response.Write("Value is - " + li.Value + "<br/>");
            Response.Write("Text is - " + li.Text + "<br/>");
            Response.Write("Index is - " + ListBox1.Items.IndexOf(li) +
"<br/>");
        }
    }
}

```

ListBox (If SelectionMode-Single), RadioButtonList and DropDownList allows user to select only one item. So, use SelectedIndex and SelectedItem properties to retrieve the Text, Value and Index of the selected ListItem. Reusable method that can be used with any control that derives from ListControl class, but works best with controls that allows single selection.

```

public void RetrieveSingleData(ListControl listControls)
{
    if (listControls.SelectedIndex != -1)
    {
        Response.Write(listControls + "<br/>");
        Response.Write("Value is - " + listControls.SelectedItem.Value +
"<br/>");
        Response.Write("Text is - " + listControls.SelectedItem.Text +
"<br/>");
        Response.Write("Index is - " + listControls.SelectedIndex.ToString() +
"<br/>");
    }
}

protected void Unnamed1_Click(object sender, EventArgs e)
{
    RetrieveMultipleData(CheckBoxList2);
    RetrieveMultipleData(ListBox2);
    RetrieveMultipleData(BulletedList2);
    RetrieveSingleData(ListBox2);
    RetrieveSingleData(DropDownList2);
    RetrieveSingleData(RadioButtonList2);
}

```

```
RetrieveSingleData(BulletedList2);  
}
```

Mapping Virtual Path to Physical path

Virtual Path is in other word the URL, Physical path the location where the file is in HDD.

Server.MapPath() Method

Server.MapPath() returns the physical path for a given virtual path. This method can be used in several different ways, depending on the characters that we use in the virtual path.

- `Server.MapPath(".")` → Current physical directory of the page that is running
- `Server.MapPath("..")` → Parent physical directory of the page that is running
- `Server.MapPath("~/")` → The physical path of the root directory of the application

OUTPUT

`Server.MapPath(".")` - D:\472124\T Factor New\PragimTech\Module 4\1. ASP.NET Tutorial for Beginners\ASP.NET Tutorial for Beginners\ASP.NET Tutorial for Beginners\9. Mapping\ChildOfMapping --- **This is the actual Physical Path of the .aspx**

`Server.MapPath("..")` - D:\472124\T Factor New\PragimTech\Module 4\1. ASP.NET Tutorial for Beginners\ASP.NET Tutorial for Beginners\ASP.NET Tutorial for Beginners\9. Mapping --- **This is the Physical Path of the parent folder of the .aspx**

`Server.MapPath("~/")` - D:\472124\T Factor New\PragimTech\Module 4\1. ASP.NET Tutorial for Beginners\ASP.NET Tutorial for Beginners\ASP.NET Tutorial for Beginners\ --- **This is the root directory of the .aspx. To get root directory always use this.**

`Server.MapPath("../")` - D:\472124\T Factor New\PragimTech\Module 4\1. ASP.NET Tutorial for Beginners\ASP.NET Tutorial for Beginners\ASP.NET Tutorial for Beginners --- **This is the root directory of the .aspx. This could be an error if we are already in root or in the first level. ../.. means the actual file is in 2nd level form root.**

Now let say we are in a file from root directory. In that case

OUTPUT

`Server.MapPath(".")` - D:\472124\T Factor New\PragimTech\Module 4\1. ASP.NET Tutorial for Beginners\ASP.NET Tutorial for Beginners\ASP.NET Tutorial for Beginners. --- **This is the actual Physical Path of the.aspx**

`Server.MapPath("../")` - **This is the Physical Path of the parent folder. Error in this case, as we are already in root folder and this does not have any parent. `System.Web.HttpException: 'Cannot use a leading .. to exit above the top directory.'`**

`Server.MapPath("~/")` - D:\472124\T Factor New\PragimTech\Module 4\1. ASP.NET Tutorial for

Beginners\ASP.NET Tutorial for Beginners\ASP.NET Tutorial for Beginners\--- **This is the root directory.**

Server.MapPath("../..") - **This is the root directory of the .aspx. Error in this case, as we are already in root folder and this does not have any parent. `System.Web.HttpException: 'Cannot use a leading .. to exit above the top directory.'`**

FileUpload

```
<asp:FileUpload ID="FileUpload1" runat="server" /><br />
<asp:Button runat="server" id="Upload" Text="Click" OnClick="Upload_Click"/>
<asp:Label runat="server" id="lblStatus" />

protected void Upload_Click(object sender, EventArgs e)
{
    if (FileUpload1.HasFile)
    {
        string fileExtension =
System.IO.Path.GetExtension(FileUpload1.FileName);
        if (fileExtension == ".docx")
        {
            int fileSize = FileUpload1.PostedFile.ContentLength;
            if (fileSize <= 20000)
            {
                FileUpload1.SaveAs(Server.MapPath("~/contents/" +
FileUpload1.FileName));
                lblStatus.Text = "Success";
                lblStatus.ForeColor = System.Drawing.Color.Red;
            }
            else {
                lblStatus.Text = "Size exceeds";
                lblStatus.ForeColor = System.Drawing.Color.DarkRed;
            }
        }
        else {
            lblStatus.Text = "Upload word only";
            lblStatus.ForeColor = System.Drawing.Color.DeepSkyBlue;
        }
    }
    else
    {
        lblStatus.Text = "Select a file";
        lblStatus.ForeColor = System.Drawing.Color.GreenYellow;
    }
}
```

AdRotater

AdRotater control is used to display random ads. The ads information can be stored in an xml file or in a database table.

XML file attributes

ImageUrl -The URL of the image to display

NavigateUrl -The URL to navigate to, when the ad is clicked

Alternate Text-The text to use if the image is missing

Keyword -Used by the AdRotator control to filter ads

Impressions-A numeric value (a weighting number) that indicates the likelihood of how often the ad is displayed.

To open the target web page in a separate browser window, set Target="_blank"

Use Keyword attribute to filter ads

The Keyword Filter and Advertisement File properties can be changed at runtime also.

Changing the Keyword Filter at runtime could be very useful. For example, when the AdRotator control is on a master page, and if want to change the Keyword Filter on each content page.

1. Aspx

```
<asp:AdRotator runat="server" KeywordFilter="Google" Target="_blank"
AdvertisementFile=~\contents/AddData.xml"></asp:AdRotator>
```

2. Xml

```
<?xml version="1.0" encoding="utf-8" ?>
<Advertisements>
  <Ad>
    <ImageUrl>~/Images/Google.png</ImageUrl>
    <NavigateUrl>http://google.com</NavigateUrl>
    <AlternateText>Please visit http://www.Google.com</AlternateText>
    <Impressions>10</Impressions>
    <Keyword>Google</Keyword>
  </Ad>
  <Ad>
    <ImageUrl>~/Images/Pragim.png</ImageUrl>
    <NavigateUrl>https://www.pragimtech.com/</NavigateUrl>
    <AlternateText>Please visit http://www.pragistech.com</AlternateText>
    <Impressions>30</Impressions>
    <Keyword>Pragim</Keyword>
  </Ad>
  <Ad>
    <ImageUrl>~/Images/Youtube.png</ImageUrl>
    <NavigateUrl>http://Youtube.com</NavigateUrl>
    <AlternateText>Please visit http://www.Youtube.com</AlternateText>
    <Impressions>20</Impressions>
    <Keyword>Google</Keyword>
  </Ad>
</Advertisements>
```

Here Impressions is set to determine how often the ad will appear. Greater the number, more often it will come/appear.

The KeywordFilter="Google" is used to filter which ads will appear. In this case, only keywords containing 'Google' will be displayed. 'Pragim' will never appear, no matter how many times the page is refreshed.

These can be done in code behind also.

```
addRotater.KeywordFilter = "Google";
```

Calendar Control

This can be Auto formatted from the design view.

Sample code

1. aspx

```
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<asp:ImageButton ID="ImageButton1" runat="server" Height="28px"
ImageUrl="~/Images/Calendar.png" OnClick="ImageButton1_Click" Width="29px" />
<asp:Calendar ID="Calendar1" runat="server" BackColor="#FFFFCC"
BorderColor="#FFCC66" BorderWidth="1px" DayNameFormat="Shortest" Font-Names="Verdana"
Font-Size="8pt" ForeColor="#663399" Height="200px" OnDayRender="Calendar1_DayRender"
OnSelectionChanged="Calendar1_SelectionChanged" ShowGridLines="True" Width="220px">
    <DayHeaderStyle BackColor="#FFCC66" Font-Bold="True" Height="1px" />
    <NextPrevStyle Font-Size="9pt" ForeColor="#FFFFCC" />
    <OtherMonthDayStyle ForeColor="#CC9966" />
    <SelectedDayStyle BackColor="#CCCCFF" Font-Bold="True" />
    <SelectorStyle BackColor="#FFCC66" />
    <TitleStyle BackColor="#990000" Font-Bold="True" Font-Size="9pt"
ForeColor="#FFFFCC" />
    <TodayDayStyle BackColor="#FFCC66" ForeColor="White" />
</asp:Calendar>
```

2. aspx.cs

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

namespace ASP.NET_Tutorial_for_Beginners
{
    public partial class Calendar : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                Calendar1.Visible = false;
            }
        }

        protected void ImageButton1_Click(object sender, ImageClickEventArgs e)
        {
            if (Calendar1.Visible)
            {
                Calendar1.Visible = false;
            }
            else
            {
                Calendar1.Visible = true;
            }
        }

        protected void Calendar1_SelectionChanged(object sender, EventArgs e)
        {
            TextBox1.Text = Calendar1.SelectedDate.ToString();
            Calendar1.Visible = false;
        }
    }
}
```

```

//-----Formats-----
Response.Write("ToString() -" + DateTime.Now.ToString() + "<br/>");
Response.Write("ToLongDateString() -" + DateTime.Now.ToLongDateString() +
"<br/> ");
Response.Write("ToShortDateString() -" + DateTime.Now.ToShortDateString() +
"<br/>");
Response.Write("ToLongTimeString() -" + DateTime.Now.ToLongTimeString() +
"<br/>");
Response.Write("ToShortTimeString() -" + DateTime.Now.ToShortTimeString() +
"<br/>");

    ///Output
    //ToString()-09/30/2024 11:49:14PM
    //ToLongDateString()-Monday, September 30, 2024
    //ToShortDateString()-09/30/2024
    //ToLongTimeString()-11:49:14PM
    //ToShortTimeString()-11:49PM

Response.Write("-----<br/>");

Response.Write("d-" + DateTime.Now.ToString("d") + "<br/> ");
Response.Write("D-" + DateTime.Now.ToString("D") + "<br/>");
Response.Write("dd/MM/yyyy-" + DateTime.Now.ToString("dd/MM/yyyy")
+ "<br/>");
Response.Write("dd/MMMM/yyyy - " + DateTime.Now.ToString("dd/MMMM/yyyy") +
"<br/>");
Response.Write("dd/MMMM/yy - " + DateTime.Now.ToString("dd/MMMM/yy") +
"<br/>");
Response.Write("MM/dd/yy - " + DateTime.Now.ToString("MM/dd/yy") +
"<br/>");

    ///Output
    //d-09/30/2024
    //D-Monday, September 30, 2024
    //dd/MM/yyyy-30/09/2024
    //dd/MMMM/yyyy-30/September/2024
    //dd/MMMM/yy-30/September/24
    //MM/dd/yy-09/30/24

```

Reference: <https://learn.microsoft.com/en-us/dotnet/standard/base-types/standard-date-and-time-format-strings>

```

}

protected void Calendar1_DayRender(object sender, DayRenderEventArgs e)
{
    //This event render for every day of the calendar
    Response.Write(e.Day.DayNumberText + " "); // This will show all the
    visible dates. From previous month to next months.
    if (e.Day.IsOtherMonth)
    {
        e.Day.IsSelectable = false; // disable other month date
    }

    if (e.Day.IsWeekend)
    {
        e.Day.IsSelectable = false; // disable weekends
        e.Cell.BackColor = System.Drawing.Color.Red; // change color
    }

    if (e.Day.Date.Day % 2 == 0) // for the even dates
    {

```

```

        e.Cell.BackColor = System.Drawing.Color.Green; // change color
        e.Cell.Text = "XX"; // change text
        e.Cell.ToolTip = "Booked";
        e.Cell.ForeColor = System.Drawing.Color.White;
    }
}
}

```

Useful Properties

1. **Caption**- This is a string read/write property.
2. **CaptionAlign**-Used to align the caption.
3. **DayHeaderStyle**-Style properties that can be used to customize the look and feel of the day header in the calendar
4. **DayNameFormat**-Can be Full, Short, FirstLetter, First TwoLetters, Shortest DayStyle-Style properties that can be used to customize the look and feel of the day in the calendar
5. **FirstDayOfWeek**-Which day of the week is displayed first
6. **NextPrevFormat**-Can be ShortMonth, FullMonth, Custom Text
7. **NextMonthText**-The text to use for the next month button.
8. **PrevMonthText**-The text to use for the previous month button.
9. **Select`ionMode**-Can be Day, DayWeek, DayWeekMonth. Determines if Days, Weeks and Months are selectable.

In case a user selects the entire date or an entire month, then the retrieval can be done as--

```

protected void Calendar1_SelectionChanged(object sender, EventArgs e)
{
    foreach (DateTime dt in Calendar1.SelectedDates)
    {
        Response.Write(dt.ToString("dd/MM/yyyy") + " ");
    }
}

protected void Calendar1_VisibleMonthChanged(object sender, MonthChangedEventArgs e)
{
    //This event fires when he visible month changed
    Response.Write("Month Changed from " + getMonthName(e.PreviousDate.Month)+
    " to " + getMonthName(e.NewDate.Month));
    //Output --   Month Changed from 10/01/2024 12:00:00 AM to 11/01/2024
    12:00:00 AM
}

private string getMonthName(int MonthNumber)
{
    switch (MonthNumber)
    {
        case 1:
            return "January";
        case 2:
            return "February";
        case 3:
            return "March";
    }
}

```

```

        case 4:
            return "April";
        case 5:
            return "May";
        case 6:
            return "June";
        case 7:
            return "July";
        case 8:
            return "August";
        case 9:
            return "September";
        case 10:
            return "October";
        case 11:
            return "November";
        case 12:
            return "December";
        default:
            return "No Month";
    }
}

```

HiddenField

The Hidden Field control is used to store a value that needs to be persisted across posts to the server, but don't want the control or its value visible to the user. For example, when editing and updating an employee record, we don't want the user to see the Employee Id. So, we will store the Employee Id in a Hidden Field, so that it can then be used on the server to update the correct employees record.

HiddenField:

1. Value property of the HiddenField is used to Get or set the value.
2. The value is stored as string
3. ViewState uses HiddenField to maintain state acrossPostBack
4. HiddenField is rendered as an `<input type="hidden"/>` element

Alternatives for Hidden Field:

- View state, QueryStrings, session state, and cookies can be used as an alternative for HiddenField. Session state and cookies will be accessible from other pages as well, and will be available until their timeout has reached. Whereas ViewState and Hidden Field data, is available only on that page and the data is lost when navigate to away from the page.
- HiddenField data is lost when navigate away from the page. Doesn't require any explicit clean-up task.
- HiddenField is accessible to client-side scripts

```

<script type="text/javascript">
    function GetHiddenFieldValue() {
        alert(document.getElementById('HiddenField1').value);
    }
</script>

```

Disadvantage of Hidden Field:

Hidden field data can be seen, by viewing the page source. Never, use HiddenField to store confidential data.

MultiView Controls

As the name states, a Multiview is made up of multiple view controls, and each view control in turn can have controls inside it .

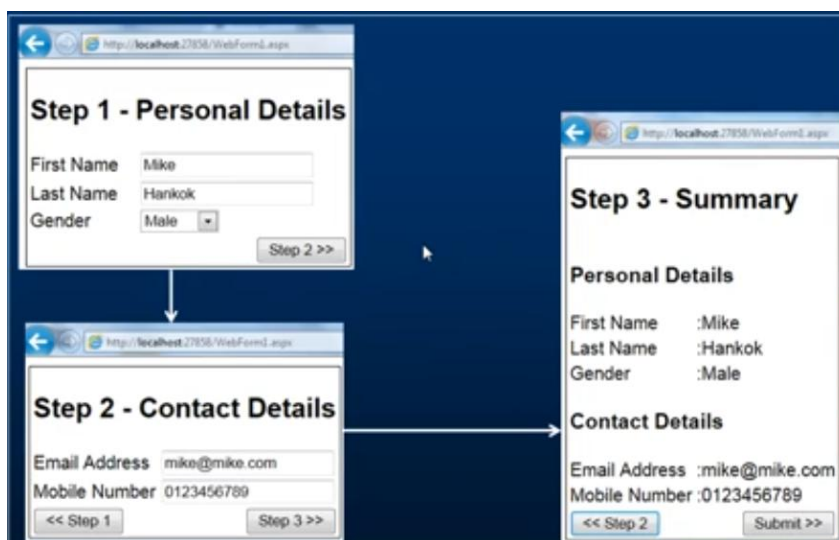
```
<asp:MultiView ID="Multiview1" runat="server">
  <asp:View ID="View1" runat="server"></asp:View>
  <asp:View ID="View2" runat="server"></asp:View>
  <asp:View ID="View3" runat="server"></asp:View>
</asp:MultiView>
```

Let's create a simple example, where we want to capture employee information on a step by step basis.

1. First capture Employee Personal details
2. Next capture Employee contact details
3. Show summary for confirmation. Upon confirmation, save the data to a database table

ActiveViewIndex property of the Multiview control is used to determine, the view that is visible or active.

This can also be achieved using the WIZARD control or by creating multiple webforms and passing data between webforms using Cookies, Query Strings or Session variables



Aspx

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="17. MultiView.aspx.cs"
Inherits="ASP.NET_Tutorial_for_Beginners.MultiView" %>
```

```
<!DOCTYPE html>
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
```

```

<title></title>
</head>

<body>

<form id="form1" runat="server">
  <asp:MultiView ID="Multiview1" runat="server">

    <asp:View ID="ViewPersonalDetails" runat="server">
      <table style="border: 1px solid black">
        <tr>
          <td colspan="2">
            <h2>Step 1 - Personal Details</h2>
          </td>
        </tr>
        <tr>
          <td>First Name</td>
          <td>
            <asp:TextBox ID="txtFN" runat="server"></asp:TextBox></td>
        </tr>
        <tr>
          <td>Last Name</td>
          <td>
            <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox></td>
        </tr>
        <tr>
          <td>Gender</td>
          <td>
            <asp:DropDownList ID="ddlGender" runat="server">
              <asp:ListItem Value="M" Text="Male"></asp:ListItem>
              <asp:ListItem Value="F" Text="Feale"></asp:ListItem>
            </asp:DropDownList></td>
        </tr>
        <tr>
          <td colspan="2" style="text-align: right">
            <asp:Button ID="ButtonStep2" Text="Step 2 >>" runat="server"
OnClick="ButtonStep2_Click"></asp:Button></td>
          </tr>
        </table>
      </asp:View>

      <asp:View ID="ViewContactdetails" runat="server">
        <table style="border: 1px solid black">
          <tr>
            <td colspan="2">
              <h2>Step2 - Contact Details</h2>
            </td>
          </tr>
          <tr>
            <td>Mobile Number</td>
            <td>
              <asp:TextBox ID="TextBox3" runat="server"></asp:TextBox></td>
          </tr>
          <tr>
            <td>Email Address</td>
            <td>
              <asp:TextBox ID="TextBox4" runat="server"></asp:TextBox></td>
          </tr>
          <tr>
            <td colspan="2">
              <asp:Button ID="ButtonStep3" Text="Step 3 >>" runat="server"
OnClick="ButtonStep3_Click"></asp:Button></td>
          </tr>
        </table>
      </asp:View>
    </asp:MultiView>
  </form>

```



```

        <td colspan="2" style="text-align: left">
            <asp:Button ID="ButtonStep1" Text="<< Step 1" runat="server"
OnClick="ButtonStep1_Click" ></asp:Button></td>
        <td style="text-align: right">
            <asp:Button ID="ButtonStep3" Text="Step 3 >>" runat="server"
OnClick="ButtonStep3_Click" ></asp:Button></td>
    </tr>
</table>
</asp:View>

<asp:View ID="ViewSummary" runat="server">
    <table style="border: 1px solid black">
        <tr>
            <td colspan="2">
                <h2>Step 3 - Summary </h2>
            </td>
        </tr>
        <tr>
            <td>
                <h2>Personal Details</h2>
            </td>
        </tr>
        <tr>
            <td>First Name</td>
            <td><asp:Label runat="server" ID="Label1"></asp:Label></td>
        </tr>
        <tr>
            <td>Last Name</td>
            <td><asp:Label runat="server" ID="Label2"></asp:Label></td>
        </tr>
        <tr>
            <td>Gender</td>
            <td><asp:Label runat="server" ID="Label3"></asp:Label></td>
        </tr>
        <tr>
            <td>
                <h2>Contact Details</h2>
            </td>
        </tr>
        <tr>
            <td>Email Address</td>
            <td><asp:Label runat="server" ID="LblEmail"></asp:Label></td>
        </tr>
        <tr>
            <td>Mobile Number</td>
            <td><asp:Label runat="server" ID="lblMobile"></asp:Label></td>
        </tr>
        <tr>
            <td colspan="2" style="text-align: left">
                <asp:Button ID="ButtonStep21" Text="<< Step 2" runat="server"
OnClick="ButtonStep2_Click"></asp:Button></td>
            <td style="text-align: right">
                <asp:Button ID="btnSubmit" Text="Submit >>" runat="server"
OnClick="btnSubmit_Click" ></asp:Button></td>
        </tr>
    </table>
</asp:View>

</asp:MultiView>
</form>

```

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

Aspx.cs

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

namespace ASP.NET_Tutorial_for_Beginners
{
    public partial class MultiView : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                Multiview1.ActiveViewIndex = 0; // The ActiveViewIndex is important,
                as it will decide to display the view. else, there will be nothing in browser
            }
        }

        protected void ButtonStep2_Click(object sender, EventArgs e)
        {
            Multiview1.ActiveViewIndex = 1; // The ActiveViewIndex is important, as
            it will decide to display the view. else, there will be nothing in browser
        }

        protected void ButtonStep1_Click(object sender, EventArgs e)
        {
            Multiview1.ActiveViewIndex = 0; // The ActiveViewIndex is important, as
            it will decide to display the view. else, there will be nothing in browser
        }

        protected void ButtonStep3_Click(object sender, EventArgs e)
        {
            Multiview1.ActiveViewIndex = 2; // The ActiveViewIndex is important, as
            it will decide to display the view. else, there will be nothing in browser
            Label1.Text = txtFN.Text;
            Label2.Text = TextBox2.Text;
            Label3.Text = ddlGender.SelectedItem.Text;
            lblEmail.Text = TextBox3.Text;
            lblMobile.Text = TextBox4.Text;

            // The controls are in a single webform, but used accrossed different views. Hence the
            multiview can access all these controls. So, this act as multiple page display, inside
            a single page.
        }

        protected void btnSubmit_Click(object sender, EventArgs e)
        {
            Response.Redirect("http://www.google.co.in");
        }
    }
}
```

Wizard

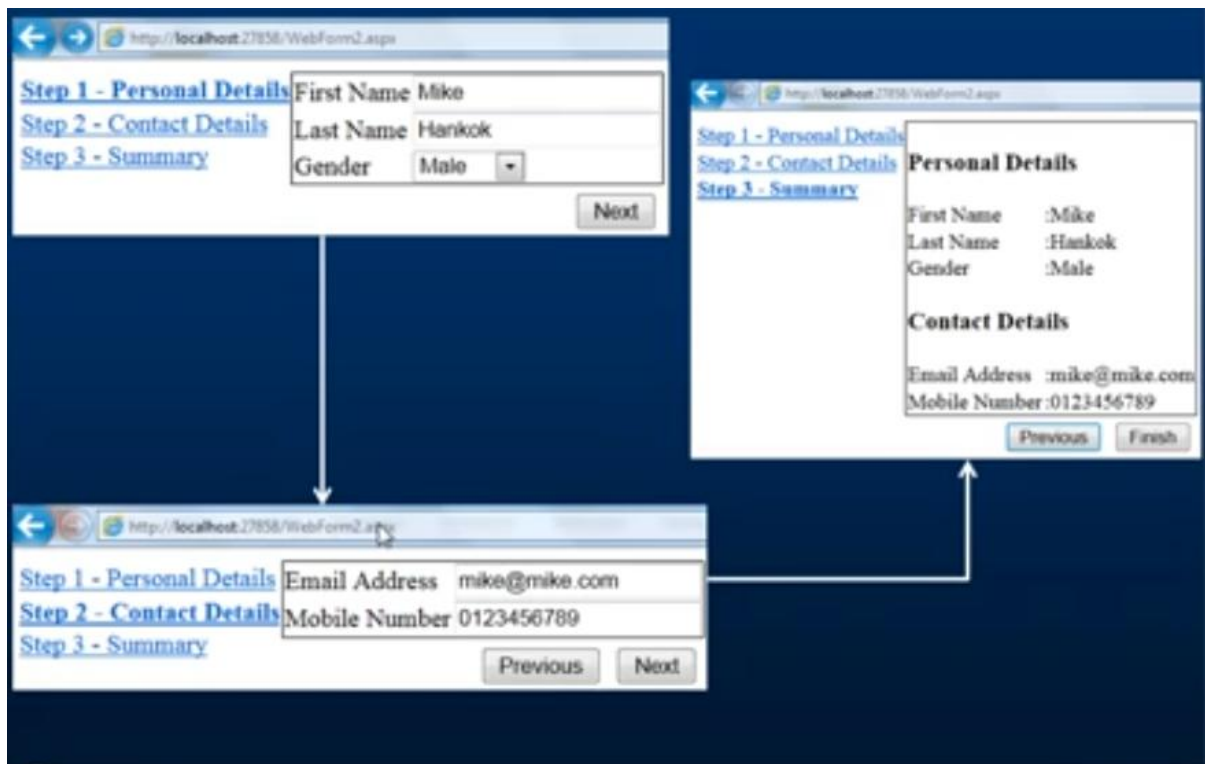
Wizard control enables creation of multi-step user interface. Wizard control provides with built-in previous/next functionality.

Let's create a simple example, where we want to capture employee information on a step by step basis.

First capture Employee Personal details

1. Next capture Employee contact details
2. Show summary for confirmation. Upon confirmation, save the data to a database table

A wizard is a collection of WizardSteps. The StepType property of WizardStep determines the correct previous/next buttons to show.



Multiview controls do have multiple views, whereas wizard have multiple wizard steps. In Multiview we need to create the NEXT PREVIOUS button manually, whereas in wizard controls those are inbuilt.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="18. WizardControl.aspx.cs" Inherits="ASP.NET_Tutorial_for_Beginners._18_WizardControl" %>
```

```
<!DOCTYPE html>
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title></title>
</head>
<body>
```

```

<form id="form1" runat="server">
    <div>
        <asp:Wizard ID="Wizard1" runat="server"
OnFinishButtonClick="Wizard1_FinishButtonClick"
OnNextButtonClick="Wizard1_NextButtonClick1">
            <SideBarStyle HorizontalAlign="Justify" VerticalAlign="Top" />
            <WizardSteps>
                <asp:WizardStep ID="WizardStep1" runat="server" Title="Step 1 --
Personal Details">
                    <table style="border: 1px solid black">
                        .....Same as MultiView View1.....
                        <%--<tr> Not Required

                            <td colspan="2" style="text-align: right">
                                <asp:Button ID="ButtonStep2" Text="Step 2 >>" runat="server"
OnClick="ButtonStep2_Click"></asp:Button></td>
                            </tr>--%>

                        </table>
                    </asp:WizardStep>
                <asp:WizardStep ID="WizardStep2" runat="server" Title="Step 2 -- Contact
Details">
                    <table style="border: 1px solid black">
                        .....Same as MultiView View2.....
                        <%--<tr> Not required

                            <td colspan="2" style="text-align: left">
                                <asp:Button ID="ButtonStep1" Text="<< Step 1" runat="server"
OnClick="ButtonStep1_Click" ></asp:Button></td>
                                <td style="text-align: right">
                                    <asp:Button ID="ButtonStep3" Text="Step 3 >>" runat="server"
OnClick="ButtonStep3_Click" ></asp:Button></td>
                                </tr>--%>

                            </table>
                        </asp:WizardStep>
                    <asp:WizardStep ID="WizardStep3" runat="server" Title="Step 3 --
Summary">
                        <table style="border: 1px solid black">
                            .....Same as MultiView View3.....
                            <%--<tr> Not Required

                                <td colspan="2" style="text-align: left">
                                    <asp:Button ID="ButtonStep21" Text="<< Step 2" runat="server"
OnClick="ButtonStep2_Click"></asp:Button></td>
                                    <td style="text-align: right">
                                        <asp:Button ID="btnSubmit" Text="Submit >>" runat="server"
OnClick="btnSubmit_Click" ></asp:Button></td>
                                    </tr>--%>

                                </table>
                            </asp:WizardStep>
                        </WizardSteps>
                    </asp:Wizard>
                </div>
            </form>
        </body>
    </html>
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;

```

```

using System.Web.UI.WebControls;

namespace ASP.NET_Tutorial_for_Beginners
{
    public partial class _18_WizardControl : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }

        protected void Wizard1_NextButtonClick1(object sender,
        WizardNavigationEventArgs e)
        {
            if (e.NextStepIndex == 2)
            {
                Label1.Text = txtFN.Text;
                Label2.Text = TextBox2.Text;
                Label3.Text = ddlGender.SelectedItem.Text.ToString();
                LblEmail.Text = TextBox3.Text;
                lblMobile.Text = TextBox4.Text;
            }
        }

        protected void Wizard1_FinishButtonClick(object sender,
        WizardNavigationEventArgs e)
        {
            Response.Redirect("https://www.google.co.in");
        }
    }
}

```

Property

ActiveStepIndex :

```

protected void Page_Load(object sender, EventArgs e)
{

    Wizard1.ActiveStepIndex = 1;

}

```

While page load the 2nd wizard will be visible by default.

Cancel Button :

```

<asp:Wizard ID="Wizard1" runat="server" CancelDestinationPageUrl="~/17.
MultiView.aspx" DisplayCancelButton="True">

```

DisplayCancelButton will place a cancel button in each wizard. **CancelDestinationPageUrl** will redirect to that page on cancel click. The **CancelButtonType** can be **button** or **image** or **link**.

SideBar :

Can be customize the sidebar

```

<SideBarStyle HorizontalAlign="Justify" VerticalAlign="Top" />

```

SideBar can be hidden also,

```

<asp:Wizard ID="Wizard1" runat="server" DisplaySideBar="False">

```

FinishCompleteButtonType, FinishPreviousButtonType :

These can be [button](#) or [image](#) or [link](#).

HeaderText:

Header text of the wizard. The header text can be changed programmatically, in Page_PreRender stage of the page.

```
protected void Page_PreRender(object sender, EventArgs e)
{
    if (Wizard1.ActiveStepIndex == 1)
    {
        Wizard1.HeaderText = "Wizard Demo -- Contact";
    };

    if (Wizard1.ActiveStepIndex == 2)
    {
        Wizard1.HeaderText = "Wizard Demo -- Summary";
    };

    if (Wizard1.ActiveStepIndex == 0)
    {
        Wizard1.HeaderText = "Wizard Demo -- Personal";
    };
}
```

Etc.....

Wizard Control events

ActiveStepChanged	Occurs when the user switches to a new step in the control.
CancelButtonClick	Occurs when the Cancel button is clicked.
FinishButtonClick	Occurs when the Finish button is clicked.
NextButtonClick	Occurs when the Next button is clicked.
PreviousButtonClick	Occurs when the Previous button is clicked.
SideBarButtonClick	Occurs when a button in the sidebar area is clicked.

Wizard Control Templates

1. Set focus to the first control in the wizard step when the page loads

```
// this will not work
protected void Page_Load(object sender, EventArgs e)
{
    // this will not work
    //if (!IsPostBack) // even giving this will also not work.
    //{
    if (Wizard1.ActiveStepIndex == 0)
    {
        TextBox1.Focus();
    }
    else if (Wizard1.ActiveStepIndex == 1)
    {
        TextBox2.Focus();
    }
}
```

```

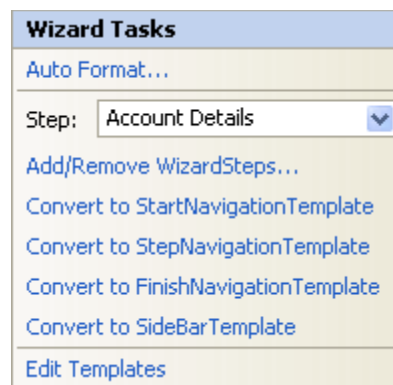
else if (Wizard1.ActiveStepIndex == 2)
{
    TextBox3.Focus();
}
//}

}
// this should be done in PreRender

protected void Page_PreRender(object sender, EventArgs e)
{
    if (Wizard1.ActiveStepIndex == 0)
    {
        TextBox1.Focus();
    }
    else if (Wizard1.ActiveStepIndex == 1)
    {
        TextBox2.Focus();
    }
    else if (Wizard1.ActiveStepIndex == 2)
    {
        TextBox3.Focus();
    }
}

```

2. Attach javascript to the Next, Previous and Finish buttons, to display a confirmation dialog box before the user moves to the next step. The Step Navigation buttons are automatically created based on the step type. For the first step only, Next Button will be there and for the last step Previous and submit buttons will be there. For the rest Next and Previous buttons will be there. But There is no declaration in HTML. Now to customize those buttons, got to **Smart Tag** of Wizard Control, then **Convert Start/Step/FinishNavigationTemplate**



After Converting the following html will be created inside Wizard Control

```
<asp:Wizard ID="Wizard1" runat="server" >
```

```

<FinishNavigationTemplate>
    <asp:Button ID="FinishPreviousButton" runat="server"
CausesValidation="False" CommandName="MovePrevious" Text="Previous"
/>
    <asp:Button ID="FinishButton" runat="server"
CommandName="MoveComplete" Text="Finish" />

```

```

        </FinishNavigationTemplate>
        <SideBarStyle HorizontalAlign="Center"
VerticalAlign="Middle" Wrap="False" />
        <SideBarTemplate>
            <asp:DataList ID="SideBarList" runat="server">
                <ItemTemplate>
                    <asp:LinkButton ID="SideBarButton"
runat="server"></asp:LinkButton>
                </ItemTemplate>
                <SelectedItemStyle Font-Bold="True" />
            </asp:DataList>
        </SideBarTemplate>
        <StartNavigationTemplate>
            <asp:Button ID="StartNextButton" runat="server"
CommandName="MoveNext" Text="Next" />
        </StartNavigationTemplate>
        <StepNavigationTemplate>
            <asp:Button ID="StepPreviousButton" runat="server"
CausesValidation="False" CommandName="MovePrevious" Text="Previous"
/>
            <asp:Button ID="StepNextButton" runat="server"
CommandName="MoveNext" Text="Next" />
        </StepNavigationTemplate>

```

```

<WizardSteps>
    <asp:WizardStep ID="WizardStep1" runat="server" Title="Step 1">
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
    </asp:WizardStep>
    <asp:WizardStep ID="WizardStep2" runat="server" Title="Step 2">
        <asp:TextBox ID="TextBox2" runat="server"></asp:TextBox>
    </asp:WizardStep>
    <asp:WizardStep ID="WizardStep3" runat="server" Title="Step 3">
        <asp:TextBox ID="TextBox3" runat="server"></asp:TextBox>
    </asp:WizardStep>
</WizardSteps>
</asp:Wizard>

```

Here we can modify, like adding javascript etc.

```

        <StartNavigationTemplate>
            <asp:Button ID="StartNextButton" runat="server" CommandName="MoveNext"
Text="Next" OnClick="return confirm('Are you sure to go to next
step');"/>

```

```

        <StepNavigationTemplate>
            <asp:Button ID="StepPreviousButton" runat="server"
CausesValidation="False" CommandName="MovePrevious" Text="Previous"
OnClick="return alert('You are going back');"/>

```



```

        <asp:Button ID="StepNextButton" runat="server" CommandName="MoveNext"
Text="Next" OnClientClick="return confirm('Are you sure to go to next
step');"/>
    </StepNavigationTemplate>

```

This also can be done from backend. But in backend these controls cannot be accessed directly, as the controls are inside a Template. So, we need to use **FindControl**. The control hierarchy can be found in the page source. E.g. for [StepPreviousButton](#)

```

<input type="submit" name="Wizard1$FinishNavigationContainerID$FinishPreviousButton"
value="Next" onclick="return confirm(&#39;Are you sure to go to next step&#39;);"
id="Wizard1_FinishNavigationContainerID_FinishPreviousButton " />

```

So, the hierarchy is Wizard1 → StartNavigationTemplateContainerID → StartNextButton

```

protected void Page_Load(object sender, EventArgs e)
{
    Button btn =
(Button)Wizard1.FindControl("FinishNavigationTemplateContainerID").FindControl("Finish
PreviousButton");// this will return a control. We need to convert that into a Button
control
    btn.OnClientClick = "return confirm('Are you sure to go to previous step
from finish?');";
}

```

3. Setting **UseSubmitBehavior="true"** for the Previous button in the wizard control.

```

<asp:Button ID="StepPreviousButton" UseSubmitBehavior="false" runat="server"
CausesValidation="False" CommandName="MovePrevious" Text="Previous"
OnClientClick="return alert('You are going back');"/>

```

UseSubmitBehavior Property

The **UseSubmitBehavior** property specifies if the Button control uses the browser's built- in submit function or the ASP.NET PostBack mechanism.

This property is **TRUE by default**. When: set to FALSE, ASP.NET adds a client-side script to post the form. To view the client-side script added by the ASP.NET, right click on the browser and view source.

```

<asp:Literal ID="Literal1" runat="server" Text="Name"></asp:Literal>
<asp:TextBox ID="txtName" runat="server"></asp:TextBox>
<asp:Button ID="btnClear" runat="server" OnClick="btnClear_Click"
Text="Clear">
<asp:Button ID="btnSubmit" runat="server" OnClick="btnSubmit_Click"
Text="Submit" />
<br />
<asp:Label ID="lblMsg" runat="server" Font-Bold="True"
ForeColor="#009933"></asp:Label>
</div>

```

```
using System;
```

```

namespace ASP.NET_Tutorial_for_Beginners
{
    public partial class _18_UseSubmitBehavior_Property : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)

```

```

    {
    }

    protected void btnClear_Click(object sender, EventArgs e)
    {
        txtName.Text = string.Empty;
    }

    protected void btnSubmit_Click(object sender, EventArgs e)
    {
        lblMsg.Text = "You Entered: " + txtName.Text;
    }
}

```

Name

In this code, by hitting enter the clear button will work, as `UseSubmitBehavior="true"` by default. So, the entered text will be cleared. To change this behaviour, we can do `UseSubmitBehavior="false"` to change the default behaviour. Now by hitting enter Submit will be clicked. If a button has the `UseSubmitBehavior="false"`, then the default **submit behavior** of the browser is changed by an automated javascript, and that can be viewed in page source.

```

<input type="button" name="btnClear" value="Clear"
onclick="javascript:__doPostBack('&#39;btnClear&#39;,&#39;&#39;)" id="btnClear" />

```

```

function __doPostBack(eventTarget, eventArgument) {
    if (!theForm.onsubmit || (theForm.onsubmit() != false)) {
        theForm.__EVENTTARGET.value = eventTarget;
        theForm.__EVENTARGUMENT.value = eventArgument;
        theForm.submit();
    }
}

```

Literal Control

1. In many ways a Literal control is similar to a Label control. Both of these controls are used to display Text on a webform. The Text property can be set in the HTML or in the code-behind.

```

protected void Page_Load(object sender, EventArgs e)
{
    Label1.Text = "Text from Label";
    Literal1.Text = "Text from Literal";
}

```

2. Label control wraps the text in a span tag when rendered. Any style that is applied to the Label control, will be rendered using the style property of the span tag. A literal control, doesn't output any surrounding tags. The Text is displayed as is.

In Page source—

```

<span id="Label1">Text from Label</span>      This is for label, Inside Span

```

Text from Literal **This is for Literal, Normal Text. No tag**

3. To apply any styles to a literal control, include them in the Text of the literal control.

```
<asp:Label runat="server" ID="Label1" Font-Bold="true" ForeColor="Red">
```

```
public partial class Literal : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        Label1.Text = "Text from Label";
        Literal1.Text = "<b><font color='Red'>Text from Literal</font></b>";
    }
}
```

```
</span id="Label1" style="color:Red;font-weight:bold;">Text from
Label</span> This is for label, Inside Span
```

```
<b><font color='Red'>Text from Literal</font></b> This is for Literal,
Normal Text. No tag
```

4. If just want the text to be displayed without any styles, then use Literal control, else use Label control.
5. By default, both the Label and Literal Control's does not encode the text they display.

```
<asp:Label runat="server" ID="Label1" Font-Bold="true" ForeColor="Red"
Text="<script>alert('Label')</script>"> </asp:Label>
<br />
<asp:Literal ID="Literal1" runat="server"
Text="<script>alert('Literal')</script>"></asp:Literal>
```

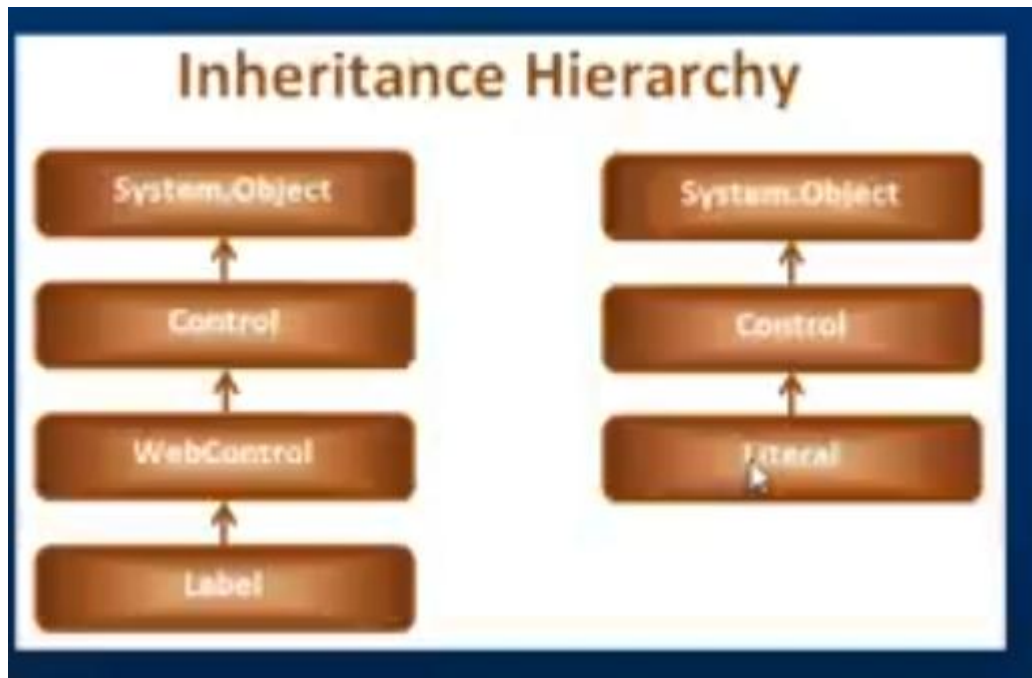
Here we are injecting javascript in both the label and Literal controls. In this case, the texts do not appear as normal text in browser, rather they will appear as alert box.

6. To HTML encode the Label Text, Server.HtmlEncode() method can be used, and for Literal control, Mode property can be used.

```
<asp:Label runat="server" ID="Label1" Font-Bold="true" ForeColor="Red">
    <%= Server.HtmlEncode("<script>alert('Label')</script>") %>
</asp:Label>
<br />
<asp:Literal ID="Literal1" mode="Encode" runat="server"
Text="<script>alert('Literal')</script>"></asp:Literal>
</div>
```

In this case they will print the text as normal text.

7. Literal control is a light weight control, when compared with the Label control.



Panel Control

The panel control is used as a container for other controls.

A panel control is very handy, when want to group controls, and then show or hide, all the controls in the group.

Panel control, is also very useful, when adding controls to the webform dynamically.

using the Panel control to group controls, and then toggle their visibility, using the Panel control's Visible property.

Let say in a page there are some controls for admin users and some for non admin users.

Let say the following code

-----ASPX-----

```

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="21. Panel.aspx.cs"
Inherits="ASP.NET_Tutorial_for_Beginners.Panel" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:DropDownList ID="DropDownList1" runat="server"
AutoPostBack="true" OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged">
                <asp:ListItem Text="Select User" Value="-1"></asp:ListItem>

```

```

        <asp:ListItem Text="Admin User" Value="Admin"></asp:ListItem>
        <asp:ListItem Text="Non admin User" Value="Non
Admin"></asp:ListItem>
    </asp:DropDownList>

    <table>
    <tr>
        <td colspan="2">
            <asp:Label ID="AdminGreeting" runat="server" Font-
Size="XX-Large" Text="You are logged in as an administrator"></asp:Label>
        </td>
    </tr>
    <tr>
        <td>
            <asp:Label ID="AdminsameLabel" runat="server" Text="Admin
Name:"></asp:Label>
        </td>
        <td>
            <asp:TextBox ID="AdminNameTextBox" runat="server"
Text="Tom"></asp:TextBox>
        </td>
    </tr>
    <tr>
        <td>
            <asp:Label ID="AdminRegionLabel" runat="server"
Text="Admin Region:"></asp:Label></td>
        <td>
            <asp:TextBox ID="AdminkegionTextBox" runat="server"
Text="Asia"></asp:TextBox>
        </td>
    </tr>
    <tr>
        <td>
            <asp:Label ID="AdminActionstabel" runat="server"
Text="Actions:"></asp:Label>
        </td>
        <td>
            <asp:TextBox ID="TextBox1" runat="server" Font-
Size="Medium" TextMode="MultiLine"
Text="There are 4 user queries to be answered by the
end of December 25th 2024." Font-Bold="True"></asp:TextBox>
        </td>
    </tr>
    </table>

    <table>
    <tr>
        <td colspan="2">
            <asp:Label ID="NonAdminGreeting" runat="server" Font-
Size="XX-Large" Text="Welcome Guest!"></asp:Label>
        </td>
    </tr>
    <tr>
        <td>
            <asp:Label ID="nonAdminLabel" runat="server" Text="User
Name:"></asp:Label>

```

```

        </td>

        <td>
            <asp:TextBox ID="nonAdminText" runat="server"
Text="Guest"></asp:TextBox>
        </td>
    </tr>
    <tr>
        <td>
            <asp:Label ID="NonAdminRegionLabel" runat="server"
Text="User Region:"></asp:Label></td>

            <td>
                <asp:TextBox ID="NonAdminRegionText" runat="server"
Text="UK"></asp:TextBox>
            </td>
        </tr>
        <tr>
            <td>
                <asp:Label ID="NonAdminCityLabel" runat="server"
Text="City"></asp:Label>
            </td>
            <td>
                <asp:TextBox ID="NonAdminCityText" runat="server" Font-
Size="Medium"
                Text="London"></asp:TextBox>
            </td>
        </tr>
    </tr>
</table>
</div>

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

```

-----ASPX.CS

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace ASP.NET_Tutorial_for_Beginners
{
    public partial class Panel : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {

```

```

        AdminGreeting.Visible = false;
        AdminsameLabel.Visible = false;
        AdminNameTextBox.Visible = false;
        AdminRegionLabel.Visible = false;
        AdminkegionTextBox.Visible = false;
        AdminActionstabel.Visible = false;
        TextBox1.Visible = false;

        NonAdminGreeting.Visible = false;
        nonAdminLabel.Visible = false;
        nonAdminText.Visible = false;
        NonAdminRegionLabel.Visible = false;
        NonAdminRegionText.Visible = false;
        NonAdminCityLabel.Visible = false;
        NonAdminCityText.Visible = false;
    }
}

protected void DropDownList1_SelectedIndexChanged(object sender,
EventArgs e)
{
    if (DropDownList1.SelectedValue == "-1")
    {
        AdminGreeting.Visible = false;
        AdminsameLabel.Visible = false;
        AdminNameTextBox.Visible = false;
        AdminRegionLabel.Visible = false;
        AdminkegionTextBox.Visible = false;
        AdminActionstabel.Visible = false;
        TextBox1.Visible = false;

        NonAdminGreeting.Visible = false;
        nonAdminLabel.Visible = false;
        nonAdminText.Visible = false;
        NonAdminRegionLabel.Visible = false;
        NonAdminRegionText.Visible = false;
        NonAdminCityLabel.Visible = false;
        NonAdminCityText.Visible = false;
    }
    else if (DropDownList1.SelectedValue == "Admin")
    {
        AdminGreeting.Visible = true;
        AdminsameLabel.Visible = true;
        AdminNameTextBox.Visible = true;
        AdminRegionLabel.Visible = true;
        AdminkegionTextBox.Visible = true;
        AdminActionstabel.Visible = true;
        TextBox1.Visible = true;

        NonAdminGreeting.Visible = false;
        nonAdminLabel.Visible = false;
        nonAdminText.Visible = false;
        NonAdminRegionLabel.Visible = false;
        NonAdminRegionText.Visible = false;
        NonAdminCityLabel.Visible = false;
        NonAdminCityText.Visible = false;
    }
    else if (DropDownList1.SelectedValue == "Non Admin")
    {

```

```

        AdminGreeting.Visible = false;
        AdminsameLabel.Visible = false;
        AdminNameTextBox.Visible = false;
        AdminRegionLabel.Visible = false;
        AdminkegionTextBox.Visible = false;
        AdminActionstabel.Visible = false;
        TextBox1.Visible = false;

        NonAdminGreeting.Visible = true;
        nonAdminLabel.Visible = true;
        nonAdminText.Visible = true;
        NonAdminRegionLabel.Visible = true;
        NonAdminRegionText.Visible = true;
        NonAdminCityLabel.Visible = true;
        NonAdminCityText.Visible = true;
    }
    else { }
}

}

}

//or    ALTERNATE .cs
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace ASP.NET_Tutorial_for_Beginners
{
    public partial class Panel : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                setAdminHide();

                setNonAdminhide();
            }
        }

        protected void DropDownList1_SelectedIndexChanged(object sender,
        EventArgs e)
        {
            if (DropDownList1.SelectedValue == "-1")
            {
                setAdminHide();
                setNonAdminhide();
            }
            else if (DropDownList1.SelectedValue == "Admin")
            {
                setAdminShow();
            }
        }
    }
}

```



```

        setNonAdminhide();
    }
    else if (DropDownList1.SelectedValue == "Non Admin")
    {
        setAdminHide();

        setNonAdminShow();
    }
    else { }
}

private void setAdminShow()
{
    AdminGreeting.Visible = true;
    Adminsamelabel.Visible = true;
    AdminNameTextBox.Visible = true;
    AdminRegionLabel.Visible = true;
    AdminregionTextBox.Visible = true;
    AdminActionstabel.Visible = true;
    TextBox1.Visible = true;
}

private void setAdminHide()
{
    AdminGreeting.Visible = false;
    Adminsamelabel.Visible = false;
    AdminNameTextBox.Visible = false;
    AdminRegionLabel.Visible = false;
    AdminregionTextBox.Visible = false;
    AdminActionstabel.Visible = false;
    TextBox1.Visible = false;
}

private void setNonAdminShow()
{
    NonAdminGreeting.Visible = true;
    nonAdminLabel.Visible = true;
    nonAdminText.Visible = true;
    NonAdminRegionLabel.Visible = true;
    NonAdminRegionText.Visible = true;
    NonAdminCityLabel.Visible = true;
    NonAdminCityText.Visible = true;
}

private void setNonAdminhide()
{
    NonAdminGreeting.Visible = false;
    nonAdminLabel.Visible = false;
    nonAdminText.Visible = false;
    NonAdminRegionLabel.Visible = false;
    NonAdminRegionText.Visible = false;
    NonAdminCityLabel.Visible = false;
    NonAdminCityText.Visible = false;
}
}
}

```

Either way these are a lot of code. So ,we can use panel to reduce the code. Just put the tables inside a panel.

-----New Panel ASPX-----

```

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="21. Panel.aspx.cs"
Inherits="ASP.NET_Tutorial_for_Beginners.Panel" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:DropDownList ID="DropDownList1" runat="server"
AutoPostBack="true" OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged">
                <asp:ListItem Text="Select User" Value="-1"></asp:ListItem>
                <asp:ListItem Text="Admin User" Value="Admin"></asp:ListItem>
                <asp:ListItem Text="Non admin User" Value="Non
Admin"></asp:ListItem>
            </asp:DropDownList>

            <asp:Panel ID="AdminPanel" runat="server">                                <%--only
changed, Rest ate the same--%>
                <table>
                    <tr>
                        <td colspan="2">
                            <asp:Label ID="AdminGreeting" runat="server" Font-
Size="XX-Large" Text="You are logged in as an administrator"></asp:Label>
                        </td>
                    </tr>
                    <tr>
                        <td>
                            <asp:Label ID="AdminsameLabel" runat="server"
Text="Admin Name:"></asp:Label>
                        </td>
                        <td>
                            <asp:TextBox ID="AdminNameTextBox" runat="server"
Text="Tom"></asp:TextBox>
                        </td>
                    </tr>
                    <tr>
                        <td>
                            <asp:Label ID="AdminRegionLabel" runat="server"
Text="Admin Region:"></asp:Label></td>
                        <td>
                            <asp:TextBox ID="AdminkegionTextBox" runat="server"
Text="Asia"></asp:TextBox>
                        </td>
                    </tr>
                    <tr>
                        <td>
                            <asp:Label ID="AdminActionstabel" runat="server"
Text="Actions:"></asp:Label>
                        </td>
                        <td>
                            <asp:TextBox ID="TextBox1" runat="server" Font-
Size="Medium" TextMode="MultiLine"
Text="There are 4 user queries to be answered by
the end of December 25th 2024." Font-Bold="True"></asp:TextBox>
                        </td>
                    </tr>
                </table>
            </asp:Panel>
        </div>
    </form>
</body>
</html>

```

```

        </tr>

    </table>

    </asp:Panel>                                <!--only changed, Rest ate the same--%>

    <asp:Panel ID="NonAdminPanel" runat="server">                                <!--only
changed, Rest ate the same--%>
        <table>
            <tr>
                <td colspan="2">
                    <asp:Label ID="NonAdminGreeting" runat="server" Font-
Size="XX-Large" Text="Welcome Guest!"></asp:Label>
                </td>
            </tr>
            <tr>
                <td>
                    <asp:Label ID="nonAdminLabel" runat="server"
Text="User Name:"></asp:Label>
                </td>
                <td>
                    <asp:TextBox ID="nonAdminText" runat="server"
Text="Guest"></asp:TextBox>
                </td>
            </tr>
            <tr>
                <td>
                    <asp:Label ID="NonAdminRegionLabel" runat="server"
Text="User Region:"></asp:Label></td>
                <td>
                    <asp:TextBox ID="NonAdminRegionText" runat="server"
Text="UK"></asp:TextBox>
                </td>
            </tr>
            <tr>
                <td>
                    <asp:Label ID="NonAdminCityLabel" runat="server"
Text="City"></asp:Label>
                </td>
                <td>
                    <asp:TextBox ID="NonAdminCityText" runat="server"
Font-Size="Medium"
Text="London"></asp:TextBox>
                </td>
            </tr>
        </table>

    </asp:Panel>                                <!--only changed, Rest ate the same--%>
</div>

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

```

-----NEW ASPX.CS-----

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace ASP.NET_Tutorial_for_Beginners
{
    public partial class Panel : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                AdminPanel.Visible = false;
                NonAdminPanel.Visible = false;
            }

            protected void DropDownList1_SelectedIndexChanged(object sender,
            EventArgs e)
            {
                if (DropDownList1.SelectedValue == "-1")
                {
                    AdminPanel.Visible = false;
                    NonAdminPanel.Visible = false;
                }
                else if (DropDownList1.SelectedValue == "Admin")
                {
                    AdminPanel.Visible = true;
                    NonAdminPanel.Visible = false;
                }
                else if (DropDownList1.SelectedValue == "Non Admin")
                {
                    AdminPanel.Visible = false;
                    NonAdminPanel.Visible = true;
                }
                else { }
            }
        }
    }
}

```

Creating control Dynamically

Control Type ☐ Label ☒ TextBox ☒ Button How Many 3

Label Controls

TextBox Controls

TextBox - 1 TextBox - 2 TextBox - 3

Button Controls

Button - 1 Button - 2 Button - 3

-----ASPX

```
<%@ Page Language="C#" AutoEventWireup="true"
CodeBehind="22.DynamicControl.aspx.cs"
Inherits="ASP.NET_Tutorial_for_Beginners._22_DynamicControl" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <table>
            <tr>
                <td>
                    <b>control Type</b>
                </td>
                <td>
                    <asp:CheckBoxList ID="chkBoxListcontroltype" runat="server"
RepeatDirection="Horizontal">
                        <asp:ListItem Text="Label" Value="Label"></asp:ListItem>
                        <asp:ListItem Text="TextBox"
Value="TextBox"></asp:ListItem>
                        <asp:ListItem Text="Button"
Value="Button"></asp:ListItem>
                    </asp:CheckBoxList>
                </td>
                <td><b>How Many</b></td>
                <td>
                    <asp:TextBox ID="TextControlsCount" runat="server"
Width="40px"></asp:TextBox>
                </td>
                <td>
                    <asp:Button ID="btnGenerateControl" runat="server"
Text="Generate Controls" style="height: 29px" OnClick="btnGenerateControl_Click"
/>
                </td>
            </tr>
            <tr>
                <td colspan="5">
                    <h3>Label Controls</h3>
                </td>
            </tr>
            <tr>
                <td colspan="5">
                    <h3>TextBox Controls</h3>
                </td>
            </tr>
            <tr>
                <td colspan="5">
                    <h3>Button Controls</h3>
                </td>
            </tr>
        </table>
    </form>
</body>
</html>
```

```

        <td colspan="5">
            <asp:Panel runat="server" ID="pnlLabels"></asp:Panel>
        </td>
    </tr>
    <tr>
        <td colspan="5">
            <h3>TextBox Controls</h3>
        </td>
    </tr>
    <tr>
        <td colspan="5">
            <asp:Panel runat="server" ID="pnlTxtBoxes"></asp:Panel>
        </td>
    </tr>
    <tr>
        <td colspan="5">
            <h3>Button Controls</h3>
        </td>
    </tr>
    <tr>
        <td colspan="5">
            <asp:Panel runat="server" ID="pnlButton"></asp:Panel>
        </td>
    </tr>
</table>
<div>
</div>
</form>
</body>
</html>

```

-----ASPX.CS

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;

namespace ASP.NET_Tutorial_for_Beginners
{
    public partial class _22_DynamicControl : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {

        }

        protected void btnGenerateControl_Click(object sender, EventArgs e)
        {
            int count = Convert.ToInt32(TextControlsCount.Text);
            foreach (ListItem lst in chkBoxListcontroltype.Items)
            {
                if (lst.Selected)
                {
                    if (lst.Value == "Label")
                    {
                        for (int i = 1; i <= count; i++)
                        {
                            Label lbl = new Label();
                            lbl.Text = "Label - " + i.ToString();
                            pnlLabels.Controls.Add(lbl);
                        }
                    }
                }
            }
        }
    }
}

```

Panel is not the only controls where we can create or add dynamic controls. There is other way also. In stead panels we can also use

But in this case there will be an error like,

In this case when the page tries to add some controls, it tries to add the controls outside the form tag. To get rid of the error, we can add the controls in the form itself.

```
this.form1.Controls.Add(lbl);  
// pnlLabels.Controls.Add(lbl);  
//this.Page.Controls.Add(txt);  
this.form1.Controls.Add(txt);  
  
//this.Page.Controls.Add(btn);  
this.form1.Controls.Add(btn);
```

In this case the controls will be created, but at the bottom of the page, just before the closing tag of `</Form>`.

Also, instead of using the panel controls, we can also make the `<td>`s to be `runat="server"` and do the same like

```
<tr>
    <td colspan="5" runat="server" id="tdTxtBoxes">
        <!--<asp:Panel runat="server" ID="pnlTxtBoxes"></asp:Panel>--%>
    </td>
</tr>

//this.form1.Controls.Add(txt);
tdTxtBoxes.Controls.Add(txt);
```

Another control we can use is Placeholder control

```
<tr>
    <td colspan="5">
        <!--<asp:Panel runat="server" ID="pnlButton"></asp:Panel>--%>
        <asp:Placeholder ID="Placeholder1"
runat="server"></asp:Placeholder>
    </td>
</tr>

Placeholder1.Controls.Add(btn);
```

RequiredFieldValidator

Validation controls are used to ensure if, the data, entered by the user is valid. Microsoft asp.net framework, provides 6 built-in validation controls.

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

These validation controls can be used to perform both client side and server side validation.

Browsers understand only client scripts and HTML. In the past to perform client side validation, developers had to write themselves the required javascript code. With validation controls, we don't have to write javascript, we can use the built-in validation controls, which will generate the required javascript for us.

For DropDownList RequiredFieldValidator, will not work as expected, as this already have some value into it, like `~Select~` etc. So, in this case we need to use the `InitialValue` property to place the value of the initial value may be `~Select~` in this case and the value in the dropdown list item, May be 0 or -1 like that.


```

        <asp:DropDownList ID="DropDownList1" runat="server"
AutoPostBack="true" OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged">
        <asp:ListItem Text="Select User" Value="-1"></asp:ListItem>
        <asp:ListItem Text="Admin User" Value="Admin"></asp:ListItem>
        <asp:ListItem Text="Non admin User" Value="Non
Admin"></asp:ListItem>
        </asp:DropDownList>

<asp:RequiredFieldValidator ID="rfvUser" ControlToValidate="DropDownList1"
runat="server" ErrorMessage="User is selected" InitialValue="-1" ForeColor="Red"
></asp:RequiredFieldValidator>

```

The properties **ControlToValidate**, **ErrorMessage**, **Display**, **SetFocusOnError**, **EnableClientScript** property **Text**. These are common for all the validation controls . **InitialValue** is only for RFV.

Client scripts can spread viruses and cause security concerns. Because of this, users may disable javascript on their browser. If this happens, client side validation is skipped. That is why, it is always a good practice to have server side validation as well.

To get rid off this we can check if all the validations are working perfectly, while clicking the button, on where the validations will be fired.

```

if (Page.IsValid)
{
    // .....Conditions.....
}
else
{
    // .....Conditions.....
}

```

RangeValidator

RangeValidator is used to check if the value is within a specified range of values. For example, to check if the age falls between 1 and 100.

Properties specific to RangeValidator control:

- **Type**-Data type of the value to check. (Currency, Date, Double, Integer, String)
- **MinimumValue**-The minimum value allowed
- **MaximumValue**-The maximum value allowed

Display property is supported by all validation controls.

- **None**-Error message not rendered and displayed next to the control. Used to show the error message only in the Validation Summary control
- **Static**-The error message is displayed next to the control if validation fails. Space is reserved on the page for the message even if validation succeeds. The span tag is rendered with style **visibility:hidden**
- **Dynamic**-The error message is displayed next to the control if validation fails. Space is not reserved on the page for the message if the validation succeeds. The span entered with style **display:none**.

CompareValidator

CompareValidator control is used to compare the value of one control with the value of another control or a constant value. The comparison operation can be any of the following

1. Equal
2. GreaterThan
3. GreaterThanEqual
4. LessThan
5. LessThanEqual
6. NotEqual
7. DataTypeCheck

CompareValidator can also be used for DataType checking.

The following are the properties that are specific to the compare validator

1. **ControlToCompare** - The control with which to compare
2. **Type**-The DataType of the value to compare. String, Integer etc.
3. **Operator**-The comparison operator. Equal, NotEqual etc.
4. **ValueToCompare**-The constant value to compare with.

SetFocusOnError property is supported by all validation controls. If this property is set to true, then the control will automatically receive focus, when the validation fails.

RegularExpressionValidator

This is a very powerful validation control. This control is used to check if the value of an associated input control matches the pattern specified by a regular expression. The only property that is **specific to the RegularExpressionValidator** is **ValidationExpression**.

By default, client side validation is turned on. To disable client side validation set **EnableClientScript** to false. This property is supported by all validation controls. To disable validation control set Enabled property to false.

CustomValidator control allows us to write a method with a custom logic to handle the validation of the value entered. If none of the other validation controls, serve our purpose, then the CustomValidator can be used.

Just like any other validation control, CustomValidator can be used for both client and server side validation. Client side and server side validation functions, needs to be written by the developer, and associate them to the custom validator control.

Properties specific to the CustomValidator control

- **OnServerValidate**-Specifies the name of the server side validation method.
- **ClientValidationFunction**- Specifies the name of the client side validation method.
- **ValidateEmptyText**-Specifies whether the validator validates the control, when the text of the control is empty. By default, this property is false, and both the client side and server side validation functions will not be invoked, if the associated input control is empty.

```

<asp:CustomValidator ID="CustomValidator1" runat="server"
    ErrorMessage="Please enter a positive even number."
    ForeColor="Red" Font-Bold="true"
    ControlToValidate="TextBox1"
    OnServerValidate="CustomValidator1_ServerValidate" ValidateEmptyText="False"
    ClientValidationFunction="isEven"></asp:CustomValidator>

```

```

protected void CustomValidator1_ServerValidate(object source,
ServerValidateEventArgs args)
{
    bool isNumber = int.TryParse(args.Value, out int number);
    string input = args.Value?.Trim();

    if (!isNumber || number <= 0 || number % 2 != 0)
    {
        args.IsValid = false;
    }
    else
    {
        args.IsValid = true;
    }
}

```

```

protected void Button1_Click(object sender, EventArgs e)
{
    if (Page.IsValid)
    {
        lblStatus.Text = "Saved";
        lblStatus.ForeColor = System.Drawing.Color.Green;
    }
    else
    {
        lblStatus.Text = "Validation Error";
        lblStatus.ForeColor = System.Drawing.Color.Red;
    }
}

```

```

<script src="https://code.jquery.com/jquery-3.6.0.min.js"></script>
<script type="text/javascript" text="javascript">
    function isEven(source, args)
    {
        if (args.Value == "") {
            args.IsValid = false;
        }
        else
        {
            if (args.Value % 2 == 0) {
                args.IsValid = true;
            }
            else {
                args.IsValid = false;
            }
        }
    }
}
</script>

```

ValidationSummary

ValidationSummary control is used to display a summary of all validation errors occurred in a Web page, at one place.

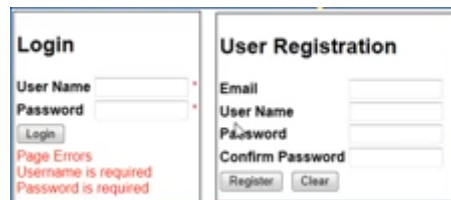
In general, in real time applications, it is common to **display a red star** symbol next to the input control where the error occurred, and then the detailed error message in the validation summary control.

Properties specific to the validation summary control:

- **HeaderText** - The header text for the validation summary control
- **ShowSummary** - Whether to display the summary text of all the validation errors
- **ShowMessageBox** - Whether to display a message box with all the validation errors
- **DisplayMode** - Display format for the summary.

Note: DisplayMode can be List, BulletList, SingleParagraph.

ValidationGroups



- **First Problem:** when clicking the Clear button, Form validation still happens. Clicking the clear button, just want to clear the textboxes in the Registration section. Validations doesn't make any sense here. So, to prevent validation from happening - Answer: **CausesValidation="False"**
- **Second problem:** when clicking the Login button, only fields in the Login section (UserName & Password) needs to be validated. Along the same lines when clicking the "Register" button, only fields in the Registration section (Email, UserName, Password and Confirm Password) needs to be validated. If don't use validation groups, then by default, whenever, clicking any button, all the validation controls on the page get validated. So, when clicking the login button, and if only, the fields in the Login section (UserName and Password) to be validated, then set, the ValidationGroup property of the validation controls and the login button control to the same group name. Use a different group name for the validation controls and register button, in the registration section

Page Navigation Techniques

- What are the different page navigation techniques in asp.net?
- How do you link pages in an application?
- How do you move from one webform to another webform in asp.net?

This is a very common interview question in asp.net. There are several techniques to navigate between webforms in asp.net as listed below.

Hyperlink control

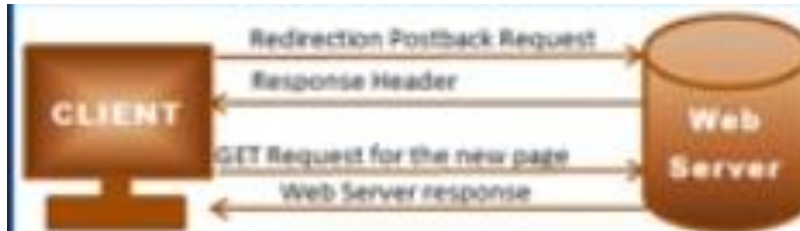
This is used to navigate to another page. The page wants to navigate to is specified by the **NavigateURL** property. Using hyperlink, can navigate to another page with in the same

application or to an external web site. The hyperlink control is rendered as an HTML anchor <a> tag. <HyperLink>

```
<asp:HyperLink runat="server" ID="internal" Text="Dynamic Controls"
NavigateUrl="~/22. DynamicControl.aspx"></asp:HyperLink>
```

```
<asp:HyperLink runat="server" ID="External" Text="Google"
NavigateUrl="https://www.google.co.in"></asp:HyperLink>
```

Response.Redirect



Response.Redirect is similar to clicking on a hyperlink. The Hyperlink control does not expose any server side events. So, when the user clicks on a hyperlink, there is no server side event to intercept the click.

So, if want to intercept a click event in code, use the Button, LinkButton or the ImageButton server control. In the button click event, call Response.Redirect() method. When the user clicks the button, the web server receives, a request for redirection. The server then sends a response header to the client. The client then automatically issues a new GET request to the web server. The web server will then serve the new page. So, in short, Response.Redirect causes 2 request/response cycles.

```
<asp:Button runat="server" ID="responseRedirect" Text="Response.Redirect Internal"
CommandName="responseRedirect" CommandArgument="internal"
OnCommand="responseRedirect_command" />
```

```
<asp:Button runat="server" ID="responseRedirectEx" Text="Response.Redirect External"
CommandName="responseRedirect" CommandArgument="External"
OnCommand="responseRedirect_command" />
```

```
e) protected void responseRedirect_command(object sender, CommandEventArgs
{
    switch (e.CommandName)
    {
        case "responseRedirect":
            if (e.CommandArgument == "internal")
            {
                Response.Redirect("~/22. DynamicControl.aspx");
            }
            else
            {
                Response.Redirect("https://www.google.co.in");
            }
            Break;
    }
}
```

Also, note that when `Response.Redirect` is used the URL in the address bar changes and the browser history is maintained. `Response.Redirect()` can be used to navigate pages/websites on the same web server or on a different web server.

Server.Transfer



Differences between Server.Transfer and Response.Redirect

1. Just like hyperlink and `Response.Redirect`, `Server.Transfer` is used to navigate to other pages/sites running on the same web server.
2. `Server.Transfer` cannot be used to navigate to sites/pages on a different web server.
3. `Server.Transfer` does not change the URL in the address bar
4. `Server.Transfer` is faster than `Response.Redirect` as the redirection happens on the server in one Request/Response cycle. `Response.Redirect()` involves 2 Request/Response cycles.
5. With `Server.Transfer` the Form Variables from the original request are preserved.

```

e) protected void responseRedirect_command(Object sender, CommandEventArgs
    {
        switch (e.CommandName)
        {
            case "serverTransfer":
                if (e.CommandArgument == "internal")
                {
                    Server.Transfer("~/3. ViewStateTextBox.aspx", false); //by default
// the second parameter is null, i.e. true. If false then the value of the current
// page will not transfer to new page
                    lblServerExecute.Text = "Completed Processing current page"; //
// won't come the control here
                }
                else
                {
                    Server.Transfer("https://www.google.co.in"); // This is
// error or exception. External webserver or link not supported for Server.Transfer
                }
                break;
        }
    }
}

```

In the next page, to get the value from this page .

Request.Form

```
System.Collections.Specialized.NameValueCollection valueFromNavigation =  
Request.Form;  
    TextBox1.Text = valueFromNavigation["TextBox1"];
```

Here ["TextBox1"] is the control name of the previous page.

Page.PreviousPage

```
Page prevPage = Page.PreviousPage;  
//In this case, even we pass false, in Server.Transfer, the value will still  
traverse from one page to another page. Page.PreviousPage property initializes  
only for Server.Transfer and cross page PostBacks. In case of Response.Redirect  
Page.PreviousPage is null. We will unable to fetch the value.
```

```
if (prevPage != null)  
{  
    TextBox1.Text = ((TextBox)prevPage.FindControl("TextBox1")).Text;  
}
```

Server.Execute

Server.Transfer and Server.Execute are similar in many ways.

1. The URL in the browser remains the first page URL.
2. Server.Transfer and Server.Execute can only be used to navigate to sites/pages on the same web server. Trying to navigate to sites/pages on a different web server, causes runtime exception.
3. Server.Transfer and Server.Execute preserves the Form Variables from the original request.

Major difference between Server.Transfer and Server.Execute –

Server.Transfer terminates the execution of the current page and starts the execution of the new page, whereas **Server.Execute** process the second Web form without leaving the first Web form. After completing the execution of the first webform, the control returns to the second webform.

```
e) protected void responseRedirect_command(Object sender, CommandEventArgs  
{  
    switch (e.CommandName)  
    {  
        case "serverExecute":  
            if (e.CommandArgument == "internal")  
            {  
                Server.Execute("~/3. ViewStateTextBox.aspx", false); //by  
default the second parameter is null, i.e. true. If false then the value of the  
current page will not transfer to new page  
                lblServerExecute.Text = "Completed Processing current  
page"; // Control will com here, in the current page  
            }  
            else
```

```

        {
            Server.Execute("https://www.google.co.in");// This is
error or exception. External webserver or link not supported for Server.Execute
        }
        break;
    default:
        break;
    }
}
}

```

Request.Form

```

        System.Collections.Specialized.NameValueCollection valueFromNavigation =
Request.Form;
        TextBox1.Text = valueFromNavigation["TextBox1"];

```

Here ["TextBox1"] is the control name of the previous page.

Page.PreviousPage

```

        Page prevPage = Page.PreviousPage;
//In this case, even we pass false, in Server.Transfer, the value will still
traverse from one page to another page. Page.PreviousPage property initializes
only for Server.Transfer and cross page PostBacks. In case of Response.Redirect
Page.PreviousPage is null. We will unable to fetch the value.

        if (prevPage != null)
        {
            TextBox1.Text = ((TextBox)prevPage.FindControl("TextBox1")).Text;
        }

```

Cross-PagePostBack

Cross page posting allows to post one page to another page. By default, when click a button, the webform posts to itself. If want to post to another webform on a button click, set the **PostBackUrl** of the button, to the page that want to post to.

Page.IsCrossPagePostBack Property is used to indicate whether the page is involved in a cross-page PostBack.

```

<asp:Button runat="server" ID="CrossPage_PostBack" Text="CrossPage_PostBack
Internal" PostBackUrl="~/3. ViewStateTextBox.aspx" />

```

Request.Form

```

        System.Collections.Specialized.NameValueCollection valueFromNavigation =
Request.Form;
        TextBox1.Text = valueFromNavigation["TextBox1"];

```


Here ["TextBox1"] is the control name of the previous page.

Page.PreviousPage

```
Page prevPage = Page.PreviousPage;  
//In this case, even we pass false, in Server.Transfer, the value will still  
traverse from one page to another page. Page.PreviousPage property initializes  
only for Server.Transfer and cross page PostBacks. In case of Response.Redirect  
Page.PreviousPage is null. We will unable to fetch the value.
```

```
if (prevPage != null && PreviousPage.IsCrossPagePostBack)  
    //this page can be reached by varieties of different ways. Like, we may  
    cross-page post back or type the URL directly and can reached here. Or we can use  
    Response.Redirect, Server.Transfer or Server.Execute etc. If we land in this page  
    by typing the URL directly, in that case, the PreviousPage will be null. In that  
    case null.FindControl will throw a NullReferenceException. To avoid run time null  
    exceptions
```

```
{  
    TextBox1.Text = ((TextBox)prevPage.FindControl("TextBox1")).Text;  
}
```

The problem with **FindControl()** method is that, if the ControlID mis-spelled the, we could get a runtime NullReferenceException.

Window.Open