

# 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 is a subset 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

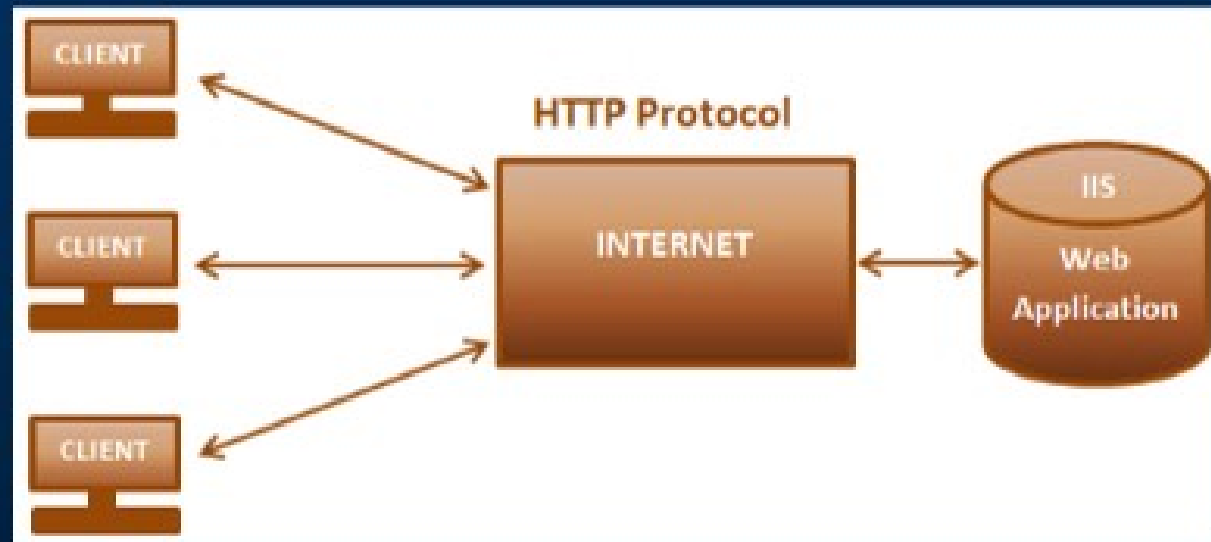
## **What other technologies can be used to build web applications**

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

# Web Applications

## What are the advantages of Web applications?

1. Web Applications just need to be installed only on the web server, where as desktop applications need to be installed on every computer, where you want 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 you need is a **browser**, that can **understand HTML**
3. On the server side, the Web application runs under **Microsoft Internet Information Services (IIS)**

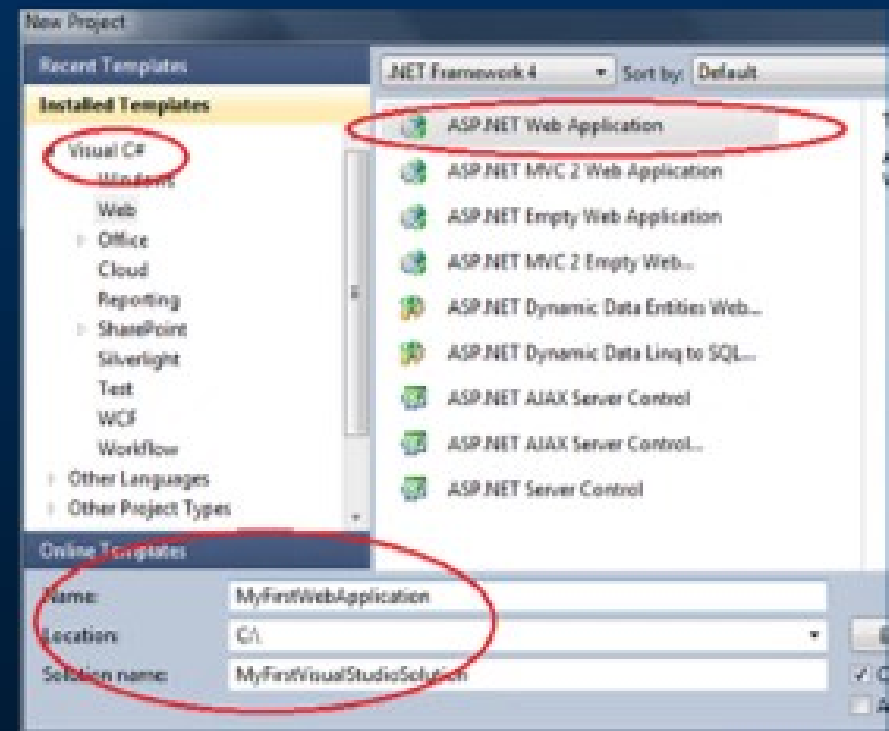
# Start Page in Visual Studio

When you first run visual studio, you will see the start page. The start page contains latest news related to .NET development, learning and community resources. At the bottom of visual studio 2010 start page

1. Close page after project load
2. Show page on startup

If you have closed the start page, and later, if you want to see it again, select **START PAGE** from the **VIEW** menu.

**Creating your first ASP.NET web application**  
Select File -> New Project



# Different windows in visual studio

## Solution Explorer:

To view the solution explorer window, from the **VIEW** menu, select **SOLUTION EXPLORER**. Or you can also use keyboard shortcut, **CTRL + W, S**. On the solution explorer, use the **AUTO-HIDE** push pin button, to either show or hide solution explorer.

**Visual Studio** organizes applications into **projects** and **solutions**. The solution file will have a **.sln** extension and the project file will have **.csproj** or **.vbproj**

**STARTUP PROJECT** in solution explorer is **bolded**. To change your start up project, **RIGHT CLICK** the project, and select "**SET AS STARTUP PROJECT**"

## Tool Box:

To view the **TOOL BOX**, Select **TOOL BOX** from the **VIEW** menu, or use the keyboard shortcut, **CTRL + W, X**

## Properties Window:

Used to change property of a webform or a control on a webform. To view the Properties window, select **PROPERTIES WINDOW** from the **VIEW** menu, or use keyboard shortcut **CTRL + W, P**

# Web forms

**Web Forms:** WebForms has the extension of .aspx.

A web form also has a code behind and designer files. Code behind files has the extension of .aspx.cs or .aspx.vb.

Designer files contains the extension of .aspx.designer.cs or .aspx.designer.vb.

Code behind files contain the code that user writes, where as the designer file contains the auto generated code. You shouldn't change the code in the designer file, because that code might later be modified by Visual Studio and your changes could be overwritten.

A Web form is associated with its code file using the @Page directive

```
<%@ Page Title="Home Page" Language="C#" MasterPageFile="~/Site.master" AutoEventWireup="true"
    CodeBehind="Default.aspx.cs" Inherits="WebApplication2._Default" %>
```

A webform's **HTML** can be edited either in **Source** or **Design** mode. You can also choose **SPLIT** mode, which shows both the DESIGN and the SOURCE at the same time.

# ViewState

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



0 Click Me

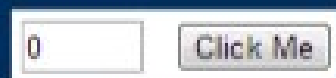
```
public partial class WebForm1 : System.Web.UI.Page
{
    int ClicksCount = 0;
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            TextBox1.Text = "0";
        }
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        ClicksCount = ClicksCount + 1;
        TextBox1.Text = ClicksCount.ToString();
    }
}
```

**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

# ViewState



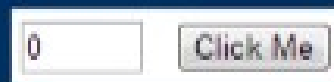
```
public partial class WebForm1 : System.Web.UI.Page
{
    int ClicksCount = 1;
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            TextBox1.Text = "0";
        }
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        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 variable Clicks to preserve the data between requests. The ViewState data, travels with every request and response between the client and the web server.

# ASP.NET Server Controls & ViewState



```
public partial class WebForm1 : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (!IsPostBack)
        {
            TextBox1.Text = "0";
        }
    }

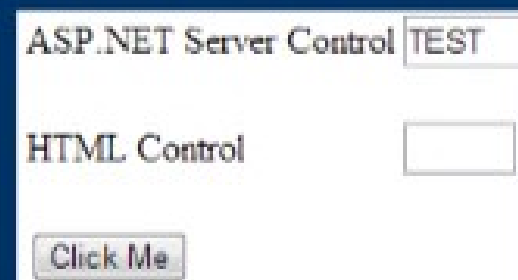
    protected void Button1_Click(object sender, EventArgs e)
    {
        int ClicksCount = Convert.ToInt32(TextBox1.Text) + 1;
        TextBox1.Text = ClicksCount.ToString();
    }
}
```

**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(), Button\_Click(), and many more events, that occur after Page\_Init() event.



# ASP.NET Server Controls & HTML Controls



The screenshot shows a white rectangular box containing two controls. The top control is labeled 'ASP.NET Server Control' and contains a text input field with the value 'TEST'. The bottom control is labeled 'HTML Control' and contains an empty text input field. Below these two controls is a button labeled 'Click Me'.

**ASP.NET server controls** retains 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="__EVENTVALIDATION" id="__EVENTVALIDATION" value="/wEWAwL68rq4DQLs0bLrBgKM54rGBjB21qgMluJwMYsqYtMHUh00dHvWNYBxEN9EeqPrZi8B" />
```

# Events

**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. Query Strings
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.

```
void Application_Start(object sender, EventArgs e)
{
    // Code that runs on application startup
}
void Application_End(object sender, EventArgs e)
{
    // Code that runs on application shutdown
}
void Application_Error(object sender, EventArgs e)
{
    // Code that runs when an unhandled error occurs
}
void Session_Start(object sender, EventArgs e)
{
    // Code that runs when a new session is started
}
void Session_End(object sender, EventArgs e)
{
    // Code that runs when a session ends.
    // Note: The Session_End event is raised only when the sessionstate mode
    // is set to InProc in the Web.config file. If session mode is set to StateServer
    // or SQLServer, the event is not raised.
}
```

# Example

GLOBAL  
·  
ASAX

```
void Application_Start(object sender, EventArgs e)
{
    // Create Application state variables
    Application["TotalApplications"] = 0;
    Application["TotalUserSessions"] = 0;
    // Increment TotalApplications by 1
    Application["TotalApplications"] = (int)Application["TotalApplications"] + 1;
}
void Session_Start(object sender, EventArgs e)
{
    // Increment TotalUserSessions by 1
    Application["TotalUserSessions"] = (int)Application["TotalUserSessions"] + 1;
}
void Session_End(object sender, EventArgs e)
{
    // Decrement TotalUserSessions by 1
    Application["TotalUserSessions"] = (int)Application["TotalUserSessions"] - 1;
}
```

WEB  
FORM  
1  
·  
ASP  
X

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Write("Number of Applications: " + Application["TotalApplications"]);
    Response.Write("<br/>");
    Response.Write("Number of Users Online: " + Application["TotalUserSessions"]);
}
```

# What is a Session

## What is a Session, in a web application?

A session is a **unique instance of the browser**. A single user can have multiple sessions, by visiting your 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>
```

# 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 you navigate away from the page, or if the browser is closed.
3. ViewState is used by all asp.net controls to retain their state across postback

## **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

## **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.

# ASP.NET Page Life Cycle Events

In Part 4, of this video series, we have discussed that, events can occur at 3 levels in an asp.net web application.

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)

**Note:** Control Level events will be discussed in a later video session

**Web applications work on a stateless protocol. Everytime 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.



# ASP.NET Page Life Cycle Events

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
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.
Init	Page Init, event occurs after the Init 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.
InitComplete	As the name says, this event gets raised immediately after page initialization.
PreLoad	Happens just before the Page Load event.
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.
Load Complete	This event is raised after the control events are handled.
PreRender	This event is raised just before the rendering stage of the page.
PreRenderComplete	Raised immediately after the PreRender event.
Unload	Raised for each control and then for the page. At this stage the page is, unloaded from memory.
Error	This event occurs only if there is an unhandled exception.



# ASP.NET Page Life Cycle Events

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

protected void Page_Init(object sender, EventArgs e)
{ Response.Write("Page_Init" + "<br/>"); }

protected void Page_InitComplete(object sender, EventArgs e)
{ Response.Write("Page_InitComplete" + "<br/>"); }

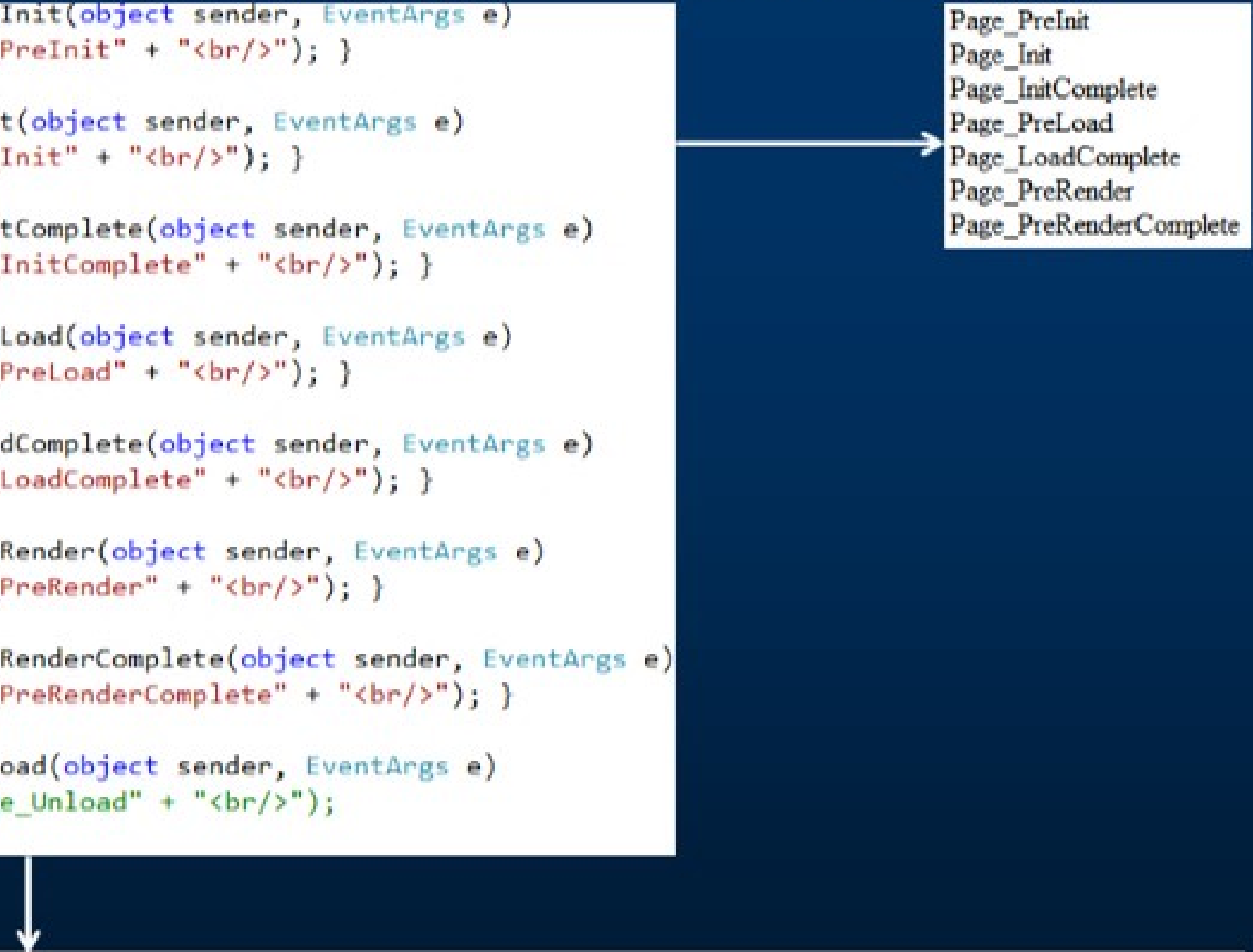
protected void Page_PreLoad(object sender, EventArgs e)
{ Response.Write("Page_PreLoad" + "<br/>"); }

protected void Page_LoadComplete(object sender, EventArgs e)
{ Response.Write("Page_LoadComplete" + "<br/>"); }

protected void Page_PreRender(object sender, EventArgs e)
{ Response.Write("Page_PreRender" + "<br/>"); }

protected void Page_PreRenderComplete(object sender, EventArgs e)
{ Response.Write("Page_PreRenderComplete" + "<br/>"); }

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



- Page\_PreInit
- Page\_Init
- Page\_InitComplete
- Page\_PreLoad
- Page\_LoadComplete
- Page\_PreRender
- Page\_PreRenderComplete

**Note:** Uncommenting `Response.Write("Page_Unload" + "<br/>")` in Page\_Unload Event, causes runtime exception  
*Response is not available in this Context*

# ASP.NET Server Control Events

In Part 4, of this video series, we have discussed that, events can occur at 3 levels in an asp.net web application.

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.

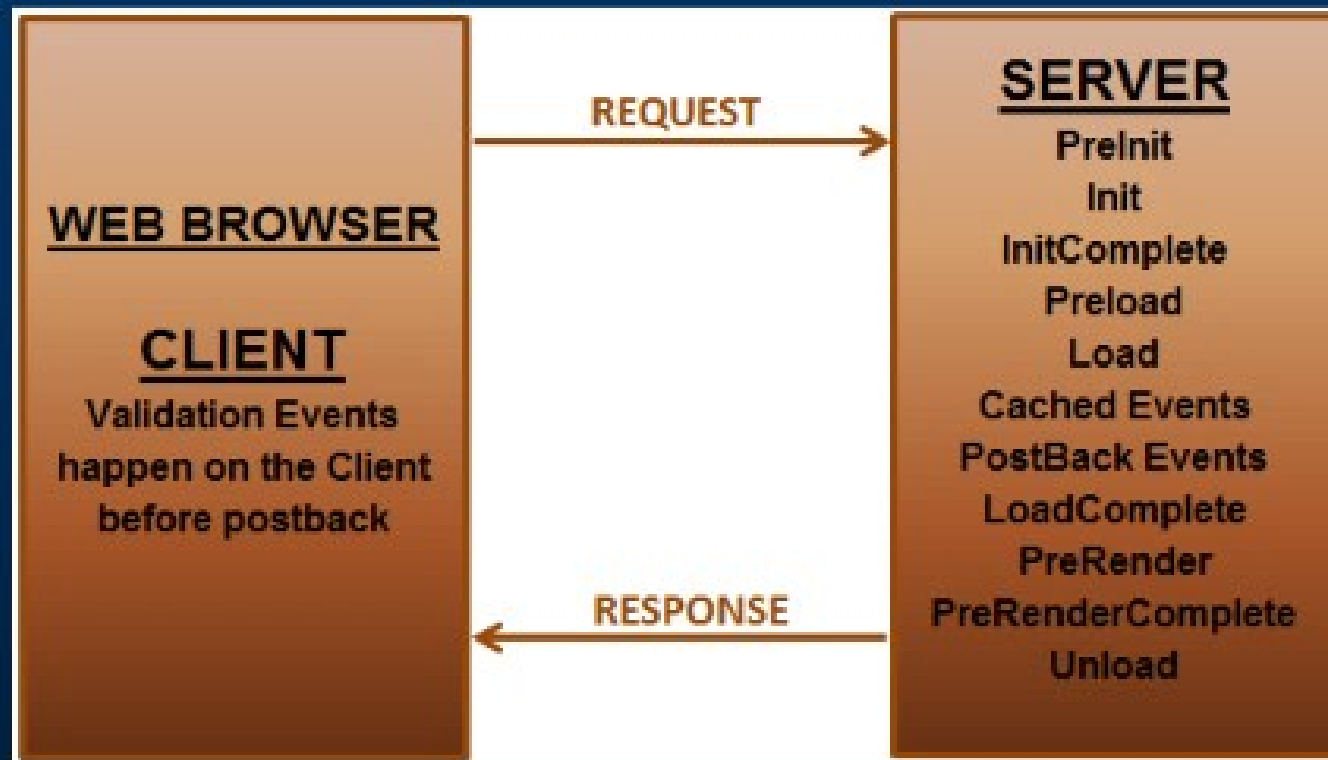
**Postback events** - These events submit the Web page, immediately to the server for processing. Click event of a button control is an example forPostBack event.

**Cached events** - These events are saved in the page's viewstate 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.

# ASP.NET Server Control Events

In Part 6, of this video series, we have understood that control events are processed after the PageLoad event. The picture below depicts the same. Among the control events, Cached events happen beforePostBack events.



**Note:** We will discuss about individual control events and validation controls in a later video session.

# ASP.NET Page Level Events

Event Name	Description
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.
Init	Page Init, event occurs after the Init 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.
InitComplete	As the name says, this event gets raised immediately after page initialization.
PreLoad	Happens just before the Page Load event.
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.
Load Complete	This event is raised after the control events are handled.
PreRender	This event is raised just before the rendering stage of the page.
PreRenderComplete	Raised immediately after the PreRender event.
Unload	Raised for each control and then for the page. At this stage the page is, unloaded from memory.
Error	This event occurs only if there is an unhandled exception.

# IsPostBack in ASP.NET

**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.

```
protected void Page_Load(object sender, EventArgs e)
{
    LoadCityDropDownList();
}
Public void LoadCityDropDownList()
{
    ListItem li1 = new ListItem("London");
    ddlCity.Items.Add(li1);

    ListItem li2 = new ListItem("Sydney");
    ddlCity.Items.Add(li2);

    ListItem li3 = new ListItem("Mumbai");
    ddlCity.Items.Add(li3);
}
protected void Button1_Click(object sender, EventArgs e)
{
}
```

Employee Details Form

First Name:

Last Name:

City:



Employee Details Form

First Name:

Last Name:

City:

**London**  
Sydney  
Mumbai  
London  
Sydney  
Mumbai

**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 **uplicated**. So, every time you click the button, the city names are again added to the DropDownList.

# 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 load, 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.

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        LoadCityDropDownList();
    }
}
```

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 your selection across postback.
2. `DropDownList` events may not work correctly as expected.

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

```
protected void Page_Load(object sender, EventArgs e)
{
    ddlCity.Items.Clear();
    LoadCityDropDownList();
}
```



# IIS & ASP.NET

## 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.

## Do you need IIS to develop and test asp.net web applications?

No, Visual Studio ships with a built-in web server. If you want to just build and test web applications on your machine, you don't need an 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 you get IIS manager window, it is installed, otherwise not installed.

## How to install IIS?

1. Click on the start button and select ControlPanel and the click on Programs
3. Click on Turn windows features on or Off, under Programs and features option
4. In the windows features, Select Internet Information Services and related services



# IIS & ASP.NET

**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 you will use in the URL, when connecting to your web application.
5. Click the button next to the textbox under physical path. Select the physical web application folder.

**You can also create the virtual directory from visual studio, on the project properties window.**

1. Select Use Local IIS Web Server
2. Project URL will be populated automatically. You can change the name of the virtual directory if you wish to do so.
3. Click Create Virtual Directory button.
4. After a few seconds the virtual directry was successfully created message will appear.
5. Click OK

# TextBox Control in ASP.NET

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 you 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 you 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.

# TextBox Control in ASP.NET

## 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, you can also find the events supported by the control.

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

# Radio Button Control in ASP.NET

**Radio Button Control** is used, when you want the user to select **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 you want the user to select his or her favourite colour.

**In short, if you want to provide the user with mutually exclusive options, then choose a Radio Button Control.**

**Gender**

☐ Male ☐ Female ☐ UnKnown

Button

**Favourite Colour**

☐ Red  
☐ Green  
☐ Blue  
☐ Yellow

Button

# Radio Button Control in ASP.NET

## Important Properties of the Radio Button Control

**Checked** - This is a boolean property, that is used to check if the button is checked or not.

**Text** - This is a string property used to get or set the text associated with the radio button control

**TextAlign** - right or left. On which side of the radio button the text should appear

**AutoPostBack** - Set this property to true, if you want the webform to be posted immediately when the checked status of the radio button changes.

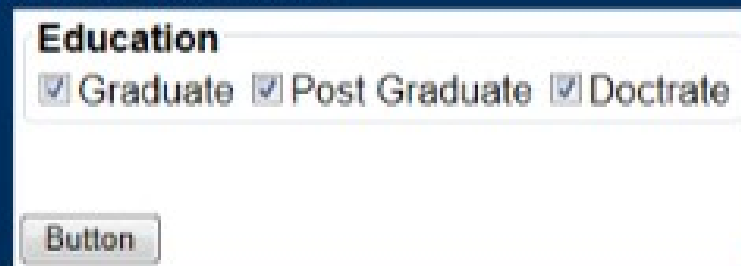
**Group Name** - By default, the individual radio button selections, are not mutually exclusive. If you have a group of radio buttons, and if you want the selections among the group to be mutually exclusive, then use the same group name for all the radio button controls.

## Events:

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

# CheckBox Control in ASP.NET

**CheckBox Control** is used, when you 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 doctrate. In this case the user selects all the 3 checkboxes. Where as a person, may just have a graduate degree, in which case he only selects, the graduate checkbox.

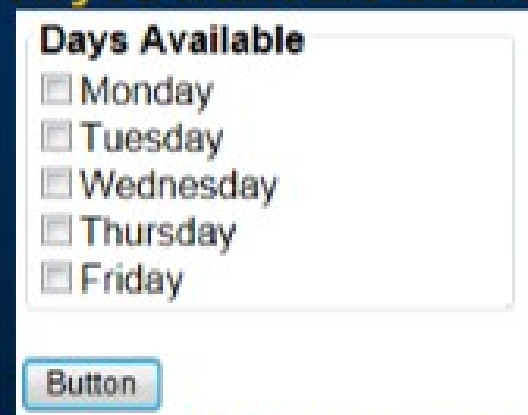


**Education**

☒ Graduate ☒ Post Graduate ☒ Doctrate

Button

**Another example, would be when you want the user to select the days of his availability.**



**Days Available**

☐ Monday  
☐ Tuesday  
☐ Wednesday  
☐ Thursday  
☐ Friday

Button

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

**Note:** You can also use CheckBoxList control, which we will discuss in a later session.

# CheckBox Control in ASP.NET

## Properties:

**Checked** - This is a boolean property, that is used to check if the check box is checked or not.

**Text** - This is a string property used to get or set the text associated with the check box control

**TextAlign** - right or left. On which side of the check box the text should appear

**AutoPostBack** - Set this property to true, if you 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 you 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.



# HyperLink Control in ASP.NET

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

**NavigateURL** - 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 you specify both the Text and ImageUrl, the image will be displayed instead of the text. If for some reason, the image is not available, 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 you 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

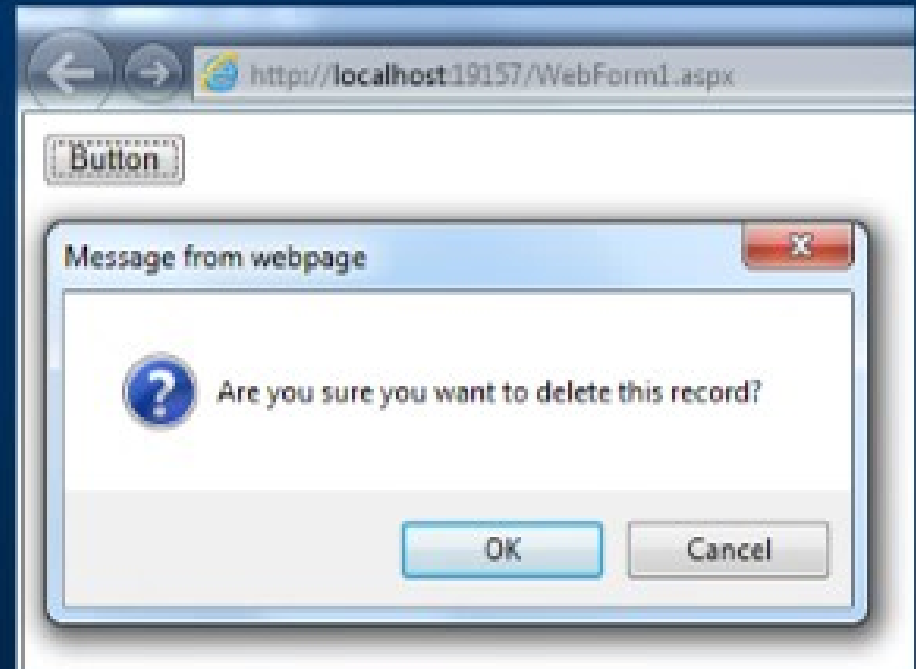
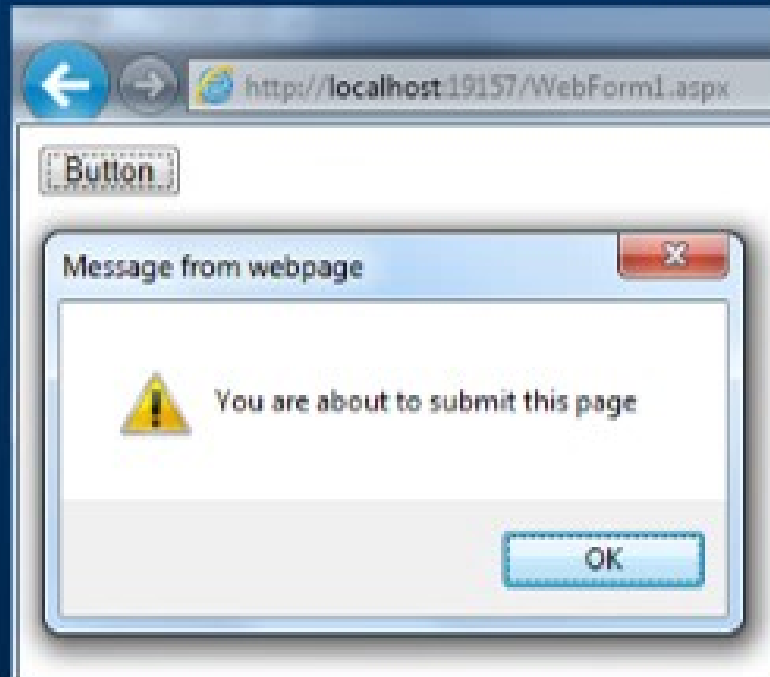
The Button, LinkButton and ImageButton controls in ASP.NET are used to post a page to the server.

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 LinkText.
3. **ImageButton** - ImageButton provides the flexibility of associating an Image with the button, using the ImageURL property.

## Properties that we will discuss later

1. CommandName
2. CommandArgument
3. CausesValidation
4. ValidationGroup
- 5.PostBackURL

# Button, LinkButton & ImageButton



# Command Event

**ASP.NET button control exposes 2 events - Click and Command events.** In Part 14, we have discussed about the **click event**. In this session we will discuss about the **Command event**. When the Button is clicked, both the events are raised. 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
2. Programmatically using delegates

**If you have multiple button controls on a webform, and if you want to programmatically determine which Button control is clicked,** we 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.

**Later:** We will discuss about Repeater, GridView, and DataList.