**.NET Framework ---**

**.NET Framework is a software framework developed by Microsoft that runs primarily on Microsoft Windows.**

**It includes a large class library named Framework Class Library (FCL) and provides language interoperability (each language can use code written in other languages) across several programming languages.**

**Programs written for .NET Framework execute in a software environment (in contrast to a hardware environment) named Common Language Runtime (CLR).**

**Microsoft also produces an integrated development environment largely for .NET software called Visual Studio.**

**It is released in 13 feb 2002.Latest version-4.7**

**In the .net framework we can do web development, desktop development, mobile apps and games.**

**Components of .Net Framework 3.5**

**-Common Language Runtime or CLR**

**-.Net Framework Class Library**

**-Common Language Specification**

**-Common Type System**

**-Metadata and Assemblies**

**-Windows Forms**

**-ASP.NET and ASP.NET AJAX**

**-ADO.NET**

**-Windows Workflow Foundation (WF)-**

**It helps in building workflow-based applications in Windows. It contains activities, workflow runtime, workflow designer, and a rules engine.**

**-Windows Presentation Foundation-**

**It provides a separation between the user interface and the business logic. It helps in developing visually stunning interfaces using documents, media, two and three dimensional graphics, animations, and more.**

**-Windows Communication Foundation (WCF)-**

**It is the technology used for building and executing connected systems.**

**-Windows CardSpace-**

**It provides safety for accessing resources and sharing personal information on the internet.**

**LINQ**

**-It imparts data querying capabilities to .Net languages using a syntax which is similar to the tradition query language SQL.**

**ASP.NET--**

**ASP.NET is an open-source server-side web application framework designed for web development to produce dynamic web pages.**

**It was developed by Microsoft to allow programmers to build dynamic web sites, web applications and web services.**

**ASP.NET works on top of the HTTP protocol, and uses the HTTP commands and policies to set a browser-to-server bilateral communication and cooperation.**

**ASP.NET is a part of Microsoft .Net platform. ASP.NET applications are compiled codes, written using the extensible and reusable components or objects present in .Net framework.**

**It was first released in January 2002 with version 1.0 of the .NET Framework, and is the successor to Microsoft's Active Server Pages (ASP) technology. ASP.NET**

**ASP.NET framework helps in storing the information regarding the state of the application, which consists of:**

**Page state**

**Session state**

**The page state is the state of the client, i.e., the content of various input fields in the web form.**

**The session state is the collective information obtained from various pages the user visited and worked with, i.e., the overall session state. To clear the concept, let us take an example of a shopping cart.**

**ASP.NET's successor is ASP.NET Core.**

**Right now in asp .net these technologies ae used mvc, webapi, Entity Framework 7, SignalR, OData protocol, Azure, Nuget**

**c# and sql is not enough you have to learn more like MVC, Webservice , WCF service, javascript , ajax, json, Internet Information Server (IIS), and jQuery.**

**latest release was 4.6**

**It consists of a large number of controls such as text boxes, buttons, and labels for assembling, configuring, and manipulating code to create HTML pages.**

**HTTP is a stateless protocol.**

**When a user requests an ASP.NET page, the IIS delegates the processing of the page to the ASP.NET runtime system.**

**Some events cause the form to be posted back to the server immediately, these are called the postback events**

**All web forms are basically instances of the ASP.NET Page class.**

**Features of asp.net-**

**1-Microsoft has made important changes with ASP.NET Core. Now, it is open source and available on GitHub. On GitHub, you can find everything about ASP.NET code and download it if you need**

**2-ASP.NET Core can run on any platform like Windows, Mac, or Linux. It means, if you create the ASP.NET Core application then it will run on any of these platforms without any issue. Earlier ASP.NET applications were not able to run on Linux and Mac operating system. But ASP.NET 5 application supports a cross platform runtime, so you can easily run the application on Mac and Linux.**

**3-ASP.NET Core has two frameworks, the first one is same as earlier .NET Framework and second one is .NET Core. DNX Core is a very light weight and open source framework which also run on Mac and Linux.**

**4-ASP.NET Core has one more command line tool manager and it is called DNVM. DNVM [DotNet Version Manager] allows you to install or upgrade the DNX Version.**

**5-ASP.NET Core is faster in development and deployment because it works on a file system. So, when you make some changes in the code, in the background internally the compilation processes are going on and also update the running application. So, there is no need to make the browser refresh.**

**6- Strongly Typed Data Controls**

**7- Value Providers-**

**ASP.NET4.5 provides many Value Providers that can be used to filter data. These are:**

**Querystring**

**Session**

**Cookie**

**Control Value**

**8- ASP.NET Web API**

**This is included in ASP.NET MVC 4 and ASP.NET Web Forms. This new ASP.NET Web API helps you to build and consume HTTP services easily**

**cross-platform-**

**In computing, cross-platform software (also multi-platform software or platform-independent software) is computer software that is implemented on multiple computing platforms.**

**Cross-platform software may be divided into two types; one requires individual building or compilation for each platform that it supports,**

**and the other one can be directly run on any platform without special preparation.**

**What is a Web Application?**

**A web application is an application that is accessed by users using a web browser. Examples of web browsers include**

**1. Microsoft Internet Explorer**

**2. Google Chrome**

**3. Mozilla FireFox**

**4. Apple Safari**

**5. Netscape Navigator**

**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**

**using System;**

**using System.Collections.Generic;**

**using System.Linq;**

**using System.Web;**

**using System.Web.UI;**

**using System.Web.UI.WebControls;**

**public partial class \_Default : System.Web.UI.Page**

**{**

**protected void Page\_Load(object sender, EventArgs e)**

**{**

**Response.Write("Page\_load <br/>");**

**}**

**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\_\_reRenderComplete <br/>");**

**}**

**protected void Page\_SaveStateComplete(object sender, EventArgs e)**

**{**

**Response.Write("Page\_\_SaveStateComplete <br/>");**

**}**

**protected void Page\_Render(object sender, EventArgs e)**

**{**

**Response.Write("Page\_\_Render <br/>");**

**}**

**protected void Button1\_Click(object sender, EventArgs e)**

**{**

**int a = int.Parse(Txtfn.Text);**

**int b = int.Parse(Txtsn.Text);**

**int sum = a + b;**

**Lvlresult.Text = "sum is " + sum;**

**}**

**protected void Button2\_Click(object sender, EventArgs e)**

**{**

**int a = int.Parse(Txtfn.Text);**

**int b = int.Parse(Txtsn.Text);**

**int diff = a - b;**

**lvl.Text = "diff is " + diff;**

**}**

**protected void Button3\_Click(object sender, EventArgs e)**

**{**

**int a = int.Parse(Txtfn.Text);**

**int b = int.Parse(Txtsn.Text);**

**int Multi = a \* b;**

**Lvlresult.Text = "Multi is " + Multi;**

**}**

**}**

**-----**

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

**<!DOCTYPE html>**

**<html xmlns="http://www.w3.org/1999/xhtml">**

**<head runat="server">**

**<title></title>**

**</head>**

**<body>**

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

**<div>**

**<h1>My first web</h1>**

**<asp:Label ID="Label1" runat="server" Text="FirstNumebr"></asp:Label><br />**

**<asp:TextBox ID="Txtfn" runat="server"></asp:TextBox><br /><br />**

**<asp:Label ID="Label2" runat="server" Text="SecondNumber"></asp:Label><br />**

**<asp:TextBox ID="Txtsn" runat="server"></asp:TextBox><br /><br />**

**<asp:Button ID="Button1" runat="server" Text="+" OnClick="Button1\_Click" />**

**<asp:Button ID="Button2" runat="server" Text="-" OnClick="Button2\_Click" />**

**<asp:Button ID="Button3" runat="server" Text="\*" OnClick="Button3\_Click" /><br />**

**<asp:Label ID="Lvlresult" runat="server" ></asp:Label>**

**<asp:Label ID="lvl" runat="server" ></asp:Label>**

**</div>**

**</form>**

**</body>**

**</html>**

**----------------**

**Add this to web.config file if you get any error-**

**<appSettings>**

**<add key="ValidationSettings:UnobtrusiveValidationMode" value="None" />**

**</appSettings>**

**----------------------------------**

**Below is the sequence of event of page events which contain master page with chronological order.**

**Master page (Page\_Init) event**

**Content page (Page\_Init) event**

**Content page (Page\_Load) event**

**Master page (Page\_Load) event**

**Content page (Page\_Prerender) event**

**Master page (Page\_Prerender) event**

**Content page (Page\_Prerendercomplete) event**

**Master page (Page\_Prerendercomplete) event**

**In case of page load event first content page load event fire then master page event fire**

**HTML, form & body tags are not allowed in content page.**

**Controls events are not sent from the content page to the master page. Similarly, you cannot handle an event from a master page control in a content page**

**---------------------------------------------**

**<link rel="shortcut icon" href="~/favicon.ico" type="image/x-icon"/>**

**----------------------------------------------------------**

**asp.net vs html controls-**

**HTML controls are the native browser elements and they are part of HTML language.**

**These are client side controls which is accessible only in the HTML page, so it will improve the performance of the web page.**

**HTML controls on an ASP.NET Web page are not available to the web server.**

**Web Server Controls are group of controls derived directly from the System.Web.UI.WebControls base class.**

**They are executed on the server side and output HTML sent back to the client browser.**

**These controls are programmable and reusable that can perform function as the ordinary HTML controls.**

**Web Server Controls can detect the target browser's capabilities and render themselves accordingly.**

**-----------------------------------------**

**events in life cycle of web app**

**webform. These events are shown in order of occurrence, except for, Error event, which occurs only if there is an unhandled exception.**

**..**

**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.**

**..**

**-------------------------------------**

**Server control events-**

**In this session, we will discuss about control level events. ASP.NET server controls, such as TextBox, Button, and DropDownList has their own events.**

**For example, Button has a click event. TextBox has TextChanged event, and DropDownList has SelectedIndexChanged event.**

**We have a set of asp.net validation controls, that has validation events. The events that all 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.**

**---------------------------**

**Web server-**

**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.**

**--------------------**

**Authentication is the process of identifying users. Authorization is the process of granting access to those users based on identity.**

**Together, authentication and authorization secures our Web application.**

**Authentication - Who is the User?**

**Authorization - What rights the user has? What resources the user can access?**

**Most of the public web sites, does not ask the user to enter any user name and password.**

**But still, we will be able to access the content of these web sites. ASP.NET Web applications provide anonymous access to resources on the server.**

**Anonymous authentication allows users to access the public areas of the web site, without prompting the users for a user name or password.**

**Windows authentication, identifies and authorizes users based on the server’s user list. Access to resources on the server is then granted or denied based on the user account’s privileges.**

**Windows authentication is best suited for Intranet Web applications.**

**The advantage of Windows authentication is that, the Web application can use the exact same security scheme that applies to your corporate network.**

**User names, passwords, and permissions are the same for network resources and Web applications.**

**-------------------------------**

**Advantages of using HTTPS-**

**HTTP stands for Hyper Text Transfer Protocol. HTTPS, stands for Hyper Text Transfer Protocol Secure.**

**As the name suggests, HTTPS is more secure than HTTP.**

**When the web server and the client communicate, using HTTP, protocol, the messages that are exchanged over the internet are not encrypted.**

**Any one can secretly listen and see the messages that are exchanged between the client and the web server.**

**That's why, any sensitive information like passwords, financial transactions should never be done over HTTP protocol.**

**Most of the banking applications use HTTPS protocol.**

**Messages exchanged between the client and web server, using the HTTPS protocol are encrypted and are very secure. HTTP use port 80 and HTTPS use port 443.**

**--------------------------------**

**What is Secure Socket Layer and how is it different from HTTPS**

**HTTPS is HTTP (HyperText Transfer Protocol) plus SSL (Secure Socket Layer).**

**SSL standing for Secure Sockets Layer (SSL) is a standard security technology for establishing an encrypted link between a web server and a browser,**

**so that the data sent over the Internet can’t be read by others.**

**When a user requests a secure Web page, the server generates an encryption key for the user’s session and then encrypts the page’s data before sending a response.**

**On the client side, the browser uses that same encryption key to decrypt the requested Web page and to encrypt new requests sent from that page.**

**SSL uses server certificates for encryption and decryption.**

**An SSL certificate contains a public key and certificate issuer.**

**Not only can clients use the certificate to communicate with a server, clients can verify that the certificate was cryptographically signed by an official Certificate Authority.**

**For example, if your browser trusts the VeriSign Certificate Authority, and VeriSign signs my SSL certificate, your browser will inherently trust my SSL certificate.**

**-------------------------------------**

**Why use master pages**

**It is common for a web site to have a consistent look and behaviour across all the pages in the web application.**

**So, the best approach to have a consistent look and feel across all pages in a web application is to use master pages.**

**----------------------------------------------------**

**About asp.net-**

**ASP.NET pages execute on the server and generate markup such as HTML, WML, or XML that is sent to a desktop or mobile browser.**

**ASP.NET offers three frameworks for creating web applications: ASP.NET Web Forms, ASP.NET MVC, and ASP.NET Web Pages.**

**All three frameworks are stable and mature, and you can create great web applications with any of them.**

**Each framework targets a different audience or type of application**

**-----------------**

**iis-**

**IIS (Internet Information Server) is one of the most powerful web servers from Microsoft that is used to host your ASP.NET Web application.**

**IIS has it's own ASP.NET Process Engine to handle the ASP.NET request**

**------------------------**

**Difference Between Dynamic & Static Web Pages**

**Static Web pages display the exact same information whenever anyone visits it. Static Web pages do not have to be simple plain text.**

**They can feature detailed multimedia design and even videos.**

**However, every visitor to that page will be greeted by the exact same text, multimedia design or video every time he visits the page until you alter that page's source code.**

**Dynamic Web pages are capable of producing different content for different visitors from the same source code file.**

**The website can display different content based on what operating system or browser the visitor is using, whether she is using a PC or a mobile device,**

**or even the source that referred the visitor. A dynamic Web page is not necessarily better than a static Web page**

**eg-online shopping website**

**----------------------------------**

**Client-side and Server-side Scripting**

**A script is a set of instructions. For Web pages they are instructions either to the Web browser (client-side scripting) or to the server (server-side scripting).**

**The client-side environment used to run scripts is usually a browser. The processing takes place on the end users computer.**

**The source code is transferred from the web server to the users computer over the internet and run directly in the browser.**

**The server-side environment that runs a scripting language is a web server.**

**A user's request is fulfilled by running a script directly on the web server to generate dynamic HTML pages.**

**This HTML is then sent to the client browser. It is usually used to provide interactive web sites that interface to databases or other data stores on the server.**

**--------------------------------------**

**Directories used in ASP.NET Web Applications?**

**The "Account" folder contains logon and security files**

**The "App\_Data" folder contains databases and data files**

**App\_Code-Contains source code for shared classes and business objects**

**The "Images" folder contains images**

**The "Scripts" folder contains browser scripts**

**The "Shared" folder contains common files (like layout and style files)**

**--------------------**

**Files used in ASP.NET Application**

**AssemblyInfo.cs:**

**This file contains a set of attributes that contain general information about the assembly such as name,description and version of the assembly.**

**Global.asax:**

**It contains scripts that define the start and end events of the application and its session.**

**Web.config:**

**It is an XML file , that contains configuration data on each unique URL resource used in the project.**

**WebApplication.vsdisco:**

**This is an XML file containing URLs that links to resource with information related to dynamic discovery of web services.**

**Webform1.aspx:**

**Web Form created by IDE ,when you start a new application.**

**----------------------------------**

**AutoEventWireup attribute in ASP.NET**

**The ASP.NET page framework supports an automatic way to associate page events and methods.**

**If the AutoEventWireup attribute of the Page directive is set to true, the page framework calls page events automatically, specifically the Page\_Init and Page\_Load methods.**

**----------------------------------**

**events in asp**

**An event is an action or occurrence such as a mouse click, a key press, mouse movements, or any system-generated notification. A process communicates through events.**

**ASP.NET event handlers generally take two parameters and return void. The first parameter represents the object raising the event and the second parameter is event argument.**

**The default event for the Page object is Load event. Similarly, every control has a default event. For example, default event for the button control is the Click event.**

**------------------------**

**user controls-**

**User controls are containers into which you can put markup and Web server controls. You can then treat the user control as a unit and define properties and methods for it.**

**The file name extension for the user control is .ascx.**

**------------------------------------------------------------------------**

**Caching-**

**Caching is a state management technique that can store a copy of the data in memory.**

**To increase the performance of the application and improve the access time, caching is used.**

**It exists in temporary storage, in other words when the data is no longer used then it expires.**

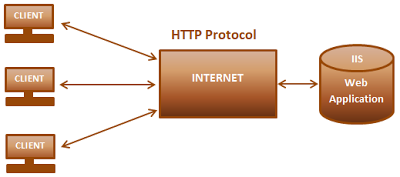
**Using a cache we can retrieve the data from a database directly.**

**3 types-**

**Output Caching**

**Data Caching**

**Fragment Caching**

**How Web applications work?**  
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)   
  
   
  
When the client enters the URL of the web application in the browser, and submits the request. The web server which hosts the web application, receives the request. The request is then processed by the application. The application generates, the HTML and hands it over to the IIS (web server). Finally, IIS sends the generated HTML to the client, who made the initial request. The client browser will the interpret the HTML and displays the user interface. All this communication, happens over the internet using HTTP protocol. HTTP stands for Hyper Text Transfer Protocol. A protocol is a set of rules that govern how two or more items communicate.

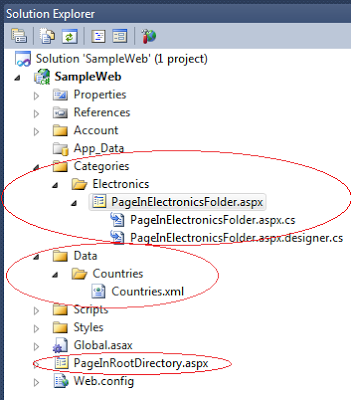
-------------------------------------------------------------------------------------------------------------  
  
Web Applications work on HTTP protocol. HTTP protocol is a stateless protocol, meaning it does not retain state between user requests..Drag and drop a TextBox and a Button control onto the webform. Change the Text property of the Button control to Click Me.  
  
**At this point, double click the button control**, which should generate the event handler in the code behind file. Modify the code behind file, so the code in WebForm1 class looks as shown below.   
**1.** In the scope of **WebForm1** class, we are creating an integer variable **ClicksCount** which is initialized to **ZERO**.  
2. On the **Page\_Load()** event handler, we are setting the **Text** property of **TextBox1** to ZERO. We do this initialization, only, when the request is an initial **GET** request.  
3. In the **Button1\_Click()** event, we are incrementing the value of the **ClicksCount** by 1, and then assigning the value to the **Text** property of **TextBox1**.  
**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();**  
**}**  
**}**  
  
**With this code in place, run the application, and click the Button.** We expect the **count**to be increased every time we click the button. When you click it the first time, it gets incremented to 1. After that, no matter how many times you click it, the value stays at 1. This is because of the **stateless nature of the web applications** that work on HTTP protocol.   
  
**So what actually happens when you make a GET request for this WebForm1?**  
**When we compile this project** an assembly is generated. Since the name of the project is **ViewStateDemo**, the name of the assembly will be **ViewStateDemo.dll**. So when a request is made for **WebForm1**, The **application's assembly**(ViewStateDemo.dll) creates an instance (object), of WebForm1, initializes **ClicksCount** to **ZERO**, and set's the **TextBox1.Text** to ZERO. As this is the initial **GET** request, the **Button1\_Click()** event will not be executed. At this point the web server, **generates the HTML** to respond to the request, and posts that response back to the browser.**It then immediately destroys the instance of the WebForm1.**  
  
**The browser receives the HTML, and we should now see textbox set to ZERO.**  
  
**What happens when we click the Button on WebForm1?**  
When we click the Button, the WebForm1 gets posted to the server. This is a **PostBack**request, **NOT A GET REQUEST**. So, when the webform is posted back, a new instance of this webform is created again, initializing the ClicksCount variable to ZERO. This time, the code that is wrapped between **IF(!ISPOSTBACK)** block is not executed. **Button1\_Click()** event gets executed as this is a **PostBack** event. ClicksCount is incremented from 0 to 1. The value is then assigned to the Text Property of TextBox1. Generates the HTML, sends it to client and destroys the webform.  
  
**At this Point, we should see the value increased to 1.**  
  
**What happens when we click the Button on WebForm1 again?**  
When you click the button for the second time, the webform gets posted back again. A new instance of WebForm1 is created. ClicksCount initialized to ZERO. In the Button1\_Click() event, the value gets incremented to 1 and assigned to TextBox1. HTML gets generated and sends it to client and destroys the webform.  
  
**So, no matter how many times you click the Button**, the value of the TextBox, will not move beyond 1.  
  
**Now, let's see, how to preserve the state between requests using ViewState variables.** Re-write the code in WebForm1, as shown below.  
**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**.  
  
**Now, let's try to achieve the same behaviour, without explicitly storing data in a ViewState** variable. Modify the WebForm1 code as shown below.  
**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. We will discuss about, all the events in the life cycle of a webform and the order in which they occur in a later video session.  
  
**On the other hand the HTML controls**, **do not retain state across post backs**. Only ASP.NET server controls retains state. To prove this  
**1.** Add a new webform to the web application project  
**2.** Drag and Drop Input(Text) control from the HTML tab, in the ToolBox  
**3.** Drag and Drop TextBox control from the Standard tab, in the ToolBox  
**4.** Finally drag and drop a button  
**5.** Set the newly added webform as the start page by right clicking on it, in the solution explorer  
**6.** Run the project, by pressing CTRL + F5  
**7.** Type "TEST" into both the controls (ASP.NET TextBox and the HTML TextBox), and press the button  
**8.** You should see that, the value in the ASP.NET TextBox is preserved across postback, but not the value in the standard HTML textbox  
  
**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" />**  
  
**Now, if you type TEST and click the button, both controls now retain state across postback.**  
  
**ViewState** data is serialized into **base64-encoded** strings, and is stored in Hidden input field **\_\_ViewState**. To view this hidden input field, right click on the browser and select "View Page Source" for google chrome. In internet explorer, right click and select "View Source"

……………

**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)  
  
**In this video, we will learn about Application Level events**. Before understanding Application level events, lets talk about Session State and Application State variables. [In Part 3](http://csharp-video-tutorials.blogspot.com/2012/10/what-is-viewstate-in-aspnet-part-3.html) of this video series we have learnt about ViewState. ViewState variables are used to preserve data across page post back. By default, ViewState of one webform is not available in another webform.   
  
For example, if you define ViewState["MyData"] = "View State Example" in WebForm1. ViewState["MyData"] is only available in WebForm1. ViewState["MyData"] will be null on any other web form in the application.  
  
**If you want to make your data available on multiple web forms**, there are several techniques in ASP.NET, as listed below.  
**1.** Query Strings  
**2.** Cookies  
**3.** Session State   
**4.** Application State  
  
**We will discuss about Query Strings and Cookies in a later video.**  
  
**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.  
  
**In an ASP.NET web application, Global.asax file conatins the application level events.**  
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.  
}   
  
  
  
**In general, Application events** are used to initialize data that needs to be available to all the current sessions of the application. Where as **Session events** are used to initialize data that needs to be available only for a given individual session, but not between multiple sessions.  
  
Now, let's write a simple application, using session and application level events. Create a new asp.net web application, and copy paste the following code in Global.asax file.  
**1.** **Application\_Start**() event gets fired, when a first request is made, and if the application is not already running.   
**2.** **Session\_Start**() event is fired every time a new browser instance, with a different session-id, visits the application.  
**3.** **Session\_End**() event is fired when the user session times out. The default is 20 minutes. This can be configured in the web.config file.  
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;  
}  
  
**Copy and paste the following code in WebForm1.aspx.**  
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"]);  
}   
  
  
  
**Now, when you run the application, you get the following output:**  
Number of Applications: 1  
Number of Users Online: 1  
  
Copy the URL and open a new instance of the browser. Paste the URL and press enter. In the new instance of the browser, we still see the same output.   
  
We expected the Number of Users Online to be 2. The new instance of the browser, is treated as part of the same session, because, by default the browser uses cookies to store session id. The session id is read from the same cookie when you opened the new browser window. Hence, Number of Users Online is not incremented.  
  
**How to get a new session-id and force the Session\_Start() event to execute?**  
**1. Close the browser:**Close the existing browser window, which automatically deletes the session cookie. Now, open a new brwoser instance. Since, the existing session cookie associated with the previous browser instance is deleted. The new instance of the browser, will get a new session-id and a session cookie.Now, if you navigate to WebForm1.aspx, Session\_Start() event gets fired and Number of Users Online is incremented to 2.  
  
**2. Open a new instance of a different browser:** For example, if you first visited the application with Google Chrome, now try accessing the same page with internet explorer, Session\_Start() event gets fired and Number of Users Online is incremented to 2.  
  
**3. Use Cookie-less Sessions:** To use cookie-less sessions set the cookieless attribute to true in web.config as shown below.  
**<sessionState mode="InProc" cookieless="false"></sessionState>**  
  
**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.

Server.MapPath() method

This method returns the physical path for a given virtual path

Create an asp.net web application in **C:\** and name it **SampleWeb**.  
**2.** Right click on the **SampleWeb** project in solution explorer and add a new webform and name it **PageInRootDirectory.aspx**  
**3.** Add a new folder to the project and name it **Categories**.  
**4.** Right click on the **Categories** folder, and add another folder. name it **Electronics**  
**5.** Add a webform to the Electronics folder and name it **PageInElectronicsFolder.aspx**  
**6.** At this point, right click on the web application project and add a new folder. Name it **Data**.  
**7.** Add a sub folder to Data, and name it **Countries**  
**8.** Right click on the Countries folder and add an XML file. Name it **Countries.xml**.  
**9.** Copy and paste the following in **Countries.xml**file.  
<?xml version="1.0" encoding="utf-8" ?>  
<Countries>  
  <Country>  
    <CountryId>101</CountryId>  
    <CountryName>India</CountryName>  
  </Country>  
  <Country>  
    <CountryId>102</CountryId>  
    <CountryName>US</CountryName>  
  </Country>  
  <Country>  
    <CountryId>103</CountryId>  
    <CountryName>Australia</CountryName>  
  </Country>  
  <Country>  
    <CountryId>104</CountryId>  
    <CountryName>UK</CountryName>  
  </Country>  
</Countries>   
  
  
  
**If you are following along with me, at this stage, the solution explorer should look as shown below.**  
   
  
**Copy and paste the following code in PageInElectronicsFolder.aspx.cs**  
Response.Write(". returns " + Server.MapPath(".") + "<br/>");  
Response.Write(".. returns " + Server.MapPath("..") + "<br/>");  
Response.Write("~ returns " + Server.MapPath("~") + "<br/>");  
  
**Running this page would produce the following output.**  
. returns C:\SampleWeb\SampleWeb\Categories\Electronics  
.. returns C:\SampleWeb\SampleWeb\Categories  
~ returns C:\SampleWeb\SampleWeb  
  
**From the output, it should be clear that**  
Server.MapPath(".") returns the current physical directory of the page that you are running  
Server.MapPath("..") returns the parent pysical directory of the page that you are running  
Server.MapPath("~") returns the physical path of the root directory of the application

3 tier architecture------------

1-visual form

2-BL-Data/table object

3-DL-Data access

4-BL-Data access method call

5-call all code in .aspx.cs file

.aspx

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

<div>

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

</div>

</form>

......

BL

public class Dataentity

{

public Dataentity()

{

//

// TODO: Add constructor logic here

//

}

public int NewsTypeId { get; set; }

public int NewsCategoryId { get; set; }

public string Heading { get; set; }

public string SubHeading { get; set; }

public string Description { get; set; }

public Guid ReporterId { get; set; }

public string Place { get; set; }

public DateTime NewsDate { get; set; }

public DateTime DateOfAdd { get; set; }

public string Images { get; set; }

public Guid EditorId { get; set; }

public int Priority { get; set; }

public bool Approved { get; set; }

public bool Read { get; set; }

public int Visit { get; set; }

public bool Active { get; set; }

public List<Dataentity> NewsCollection { get; set; }

public void GetAll()

{

NewsCollection = DataAccess.GetAllNews();

}

}

.....

DA

public class DataAccess

{

public DataAccess()

{

//

// TODO: Add constructor logic here

//

}

public static List<Dataentity> GetAllNews()

{

List<Dataentity> ln = new List<Dataentity>();

using (SqlConnection con = new SqlConnection())

{

con.ConnectionString = "Data Source=SUNNY; Initial catalog=newsreporter; Integrated Security=true";

using (SqlCommand cmd = new SqlCommand())

{

cmd.CommandType = System.Data.CommandType.Text;

cmd.CommandText = "Select \* from News";

cmd.Connection = con;

con.Open();

SqlDataReader dr = cmd.ExecuteReader();

while (dr.Read())

{

Dataentity ne = new Dataentity();

ne.NewsTypeId = Convert.ToInt32(dr[1]);

ne.NewsCategoryId = Convert.ToInt32(dr[2]);

ne.Heading = dr[3].ToString();

ne.SubHeading = dr[4].ToString();

ln.Add(ne);

}

}

}

return ln;

}

}

...........

protected void Page\_Load(object sender, EventArgs e)

{

Dataentity nType = new Dataentity();

nType.GetAll();

GridView1.DataSource = nType.NewsCollection;

GridView1.DataBind();

}