**Microsoft ASP .Net Web Forms**

**Day -1**

### What is ASP.NET?

ASP.NET is a Web application framework developed by Microsoft to build dynamic data driven Web applications (application with accessible via web browser) and Web services.

1. ASP.NET is a subset of .NET framework. In simple terms a framework is a collection of classes.
2. ASP.NET is the successor to classic ASP (Active Server Pages).

* **What other technologies can be used to build web applications --**  
  1. PHP  
  2. Java  
  3. CGI  
  4. Ruby on Rails  
  5. Perl
* **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, whereas 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
* **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** (if c# is the programming language) or **.aspx.vb** (if vb is the programming language). Designer files contains the extension of **.aspx.designer.cs** (if c# is the programming language) or **.aspx.designer.vb** (if visual basic is the programming language). Code behind files contain the code that user writes, whereas 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 found in the Web form’s HTML.  
  **<%@ 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.

**Day -2**

**In this video session, we will learn about**  
**1.** Stateless nature of HTTP protocol  
**2.** How a webform is processed  
**3.** What is ViewState  
**4.** The role of ViewState in ASP.NET  
**5.** Primary difference between ASP.NET Server controls and HTML controls  
  
**Web Applications work on HTTP protocol**. HTTP protocol is a **stateless protocol**, meaning it **does not retain state between user requests**. Let's understand the stateless nature of the HTTP protocol with an example.

**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;**  
**}**  
**}**

**Events in the life cycle of a web application** - Part 4

**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.