

# Unit - V

## State Management



# State Management

- The HTTP protocol, is the fundamental protocol of the World Wide Web, is a stateless protocol.
- That is from a web server's perspective, every request is from a new user.
- The HTTP protocol does not provide you with any method of determining whether any two requests are made by the same person.
- But maintaining state is very important, in any web application.
- The paradigmatic example is a shopping cart.
- To overcome this problem ASP.NET Provides some features like View State, Cookies, Session, Application objects etc. to manage the state of page.

# Different types of State Management

There are two different types of state management:

## 1. Client Side State Management

- View State

- Hidden Field

- Cookies

- Control State

## 2. Server Side State Management

- Session

- Application Object

- Caching

- Database

3. Client Side state management does not use any server resource , it store information using client side option. Server Side state management use server side resource for storing data.

# View State.....

1. View State is one of the most important and useful client side state management mechanism.
2. It can store the page value at the time of post back (Sending and Receiving information from Server) of your page.
3. ASP.NET pages provide the ViewState property as a built-in structure for automatically storing values between multiple requests **for the same page.**
4. Advantages of view state
  - Easy to implement
  - No server resources are required
  - Enhanced security features ,like it can be encoded and compressed.

# View State.....

## **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



# ViewState.....

## Disadvantages of view state?

- It can be performance overhead if we are going to store larger amount of data , because it is associated with page only.
- It does not have any support on mobile devices.



# Enable/ Disable Viewstate

1. You can enable and disable View state for a single control as well as at page level also.
2. To turnoff view state for a single control , set EnableViewState Property of that control to false.
3. TextBox1.EnableViewState =false;
4. To turnoff the view state of entire page, we need to set EnableViewState to false of Page Directive as shown bellow.

## How to make view state secure?

```
<%@ Page Language="C#" EnableViewState="false"
<%@ Page Language="C#" EnableViewState="true" EnableViewStateMac="true"
<%@ Page Language="C#" EnableViewState="true" ViewStateEncryptionMode="Always"
```

- MAC Stands for "Message Authentication Code"
  - ViewStateEncryptionMode has three different options to set:
    - Always
    - Auto
    - Never
1. Always, mean encrypt the view state always,
  2. Never means, Never encrypt the view state data and
  3. Auto Says , encrypt if any control request specially for encryption.

# ViewState Example

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ViewStateDemo.aspx.cs"
Inherits="ViewStateDemo" %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

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

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

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

```
</head>
```

```
<body>
```

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

```
<div>
```

```
<asp:Label ID="lblName" runat="server" Text="Enter Your Name"></asp:Label>
```

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

```
<asp:Button ID="btnSubmit" runat="server" Text="Submit"
```

```
onclick="btnSubmit_Click" /><br />
```

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

```
</div>
```

```
</form>
```

```
</body>
```

```
</html>
```



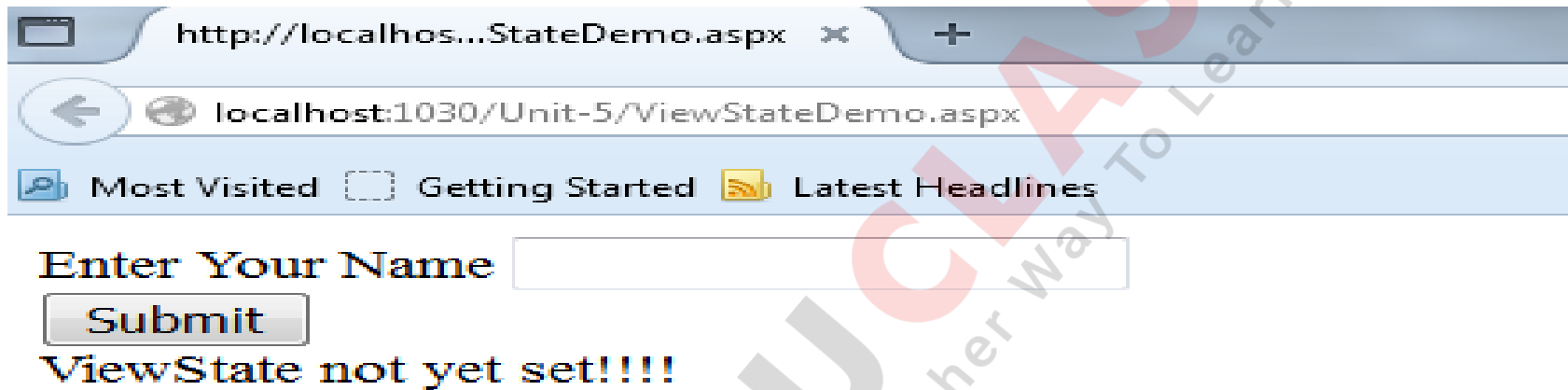
# ViewState Example

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

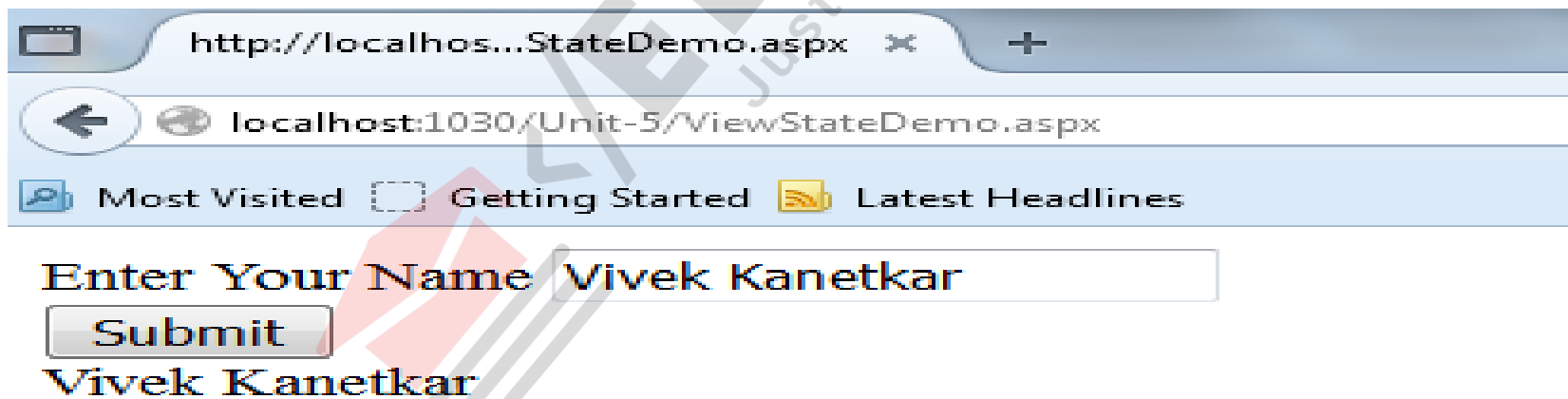
public partial class ViewStateDemo : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        if (ViewState["Name"] == null)
            lblViewStateInfo.Text = "ViewState not yet set!!!!";
        else
            lblViewStateInfo.Text = ViewState["Name"].ToString();
    }
    protected void btnSubmit_Click(object sender, EventArgs e)
    {
        ViewState["Name"] = txtName.Text;
    }
}
```

# ViewState

For the first time when page is loaded:



When Submit button is clicked: (ie for POSTBACK)



# Using Browser Cookies

- Cookies were introduced into the world with the first version of the Netscape browser. (They wanted to create a shopping cart. But as HTTP protocol is a stateless protocol it cant save the state. Hence they used cookies)....



# How the cookies work?

- When a web server creates a cookie, an additional HTTP header is sent to the browser when a page is served to the browser. The HTTP header looks like this:

Set-Cookie: message=Hello

This Set-Cookie header causes the browser to create a cookie named message that has the value Hello.

- After a cookie has been created on a browser, whenever the browser requests a page from the same application in the future, the browser sends a header that looks like this:

Cookie: message=Hello

- The Cookie header contains all the cookies that have been set by the web server. The cookies are sent back to the web server each time a request is made from the browser.
- A cookie is nothing more than a little bit of text. ***You can store only string values when using a cookie.***

# Cookies

- There are two types of cookies: ***session cookies*** and ***persistent cookies***.
- A session cookie exists only in memory. If a user closes the web browser, the session cookie disappears forever.
- A persistent cookie, on the other hand, can last for months or even years. (Persistent cookie have an expiry period, which is set at the time of cookie creation.)
- When you create a persistent cookie, the cookie is stored by the user's browser on the user's computer.



# Cookies

- Cookies are ***Browser specific***.
- Different browsers store cookies in different locations.
- Internet Explorer, for example, stores cookies in a set of text files contained in the following folder:

*\Documents and Settings\[user]\Cookies*

- The Mozilla Firefox browser, on the other hand, stores cookies in the following file:

*\Documents and Settings\[user]\ApplicationData\Mozilla\Firefox\Profiles\ ➡ [random folder name]\Cookies.txt*

- Both Internet Explorer and Firefox store cookies in clear text. Hence you should never store sensitive information—such as Social Security numbers or credit card numbers in a cookie.

# Security Restrictions on cookies

- Cookies raise security concerns. When you create a persistent cookie, you are modifying a file on a visitor's computer. A hacker may hack this file.
- To prevent cookies from doing horrible things to people's computers, browsers enforce a number of security restrictions on cookies.
  1. *First, **all cookies are domain-relative**. If the Amazon website sets a cookie, the Yahoo website cannot read that cookie.*
  2. *When a browser creates a cookie, the browser records the domain associated with the cookie and doesn't send the cookie to another domain.*

# Creating a cookie

The other important restriction that browsers place on cookies is a ***restriction on size.***

- *A single domain cannot store more than 4,096 bytes. This size restriction encompasses the size of both the cookie names and the cookie values.*
- Finally, most browsers restrict ***the number of cookies that can be set by a single domain*** to no more than 20 cookies (but not Internet Explorer).
- If you attempt to set more than 20 cookies, the oldest cookies are automatically deleted.



# Creating Cookies

- You create a new cookie by adding a cookie to the ***Response.Cookies collection***, which contains all the cookies sent from the web server to the web browser.
- For example, the following example enables you to create a new cookie named “message”.
- The page contains a form that enables you to enter the value of the message cookie



# Creating Cookies

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="CookieDemo.aspx.cs" Inherits="CookieDemo" %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

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

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

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

```
</head>
```

```
<body>
```

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

```
<div>
```

```
<asp:Label ID="lblCookie" runat="server" Text="Enter cookie value"></asp:Label>
```

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

```
<asp:Button ID="btnSetCookie" runat="server" Text="Set Cookie" onclick="btnSetCookie_Click" />
```

```
</div>
```

```
</form>
```

```
</body>
```

```
</html>
```

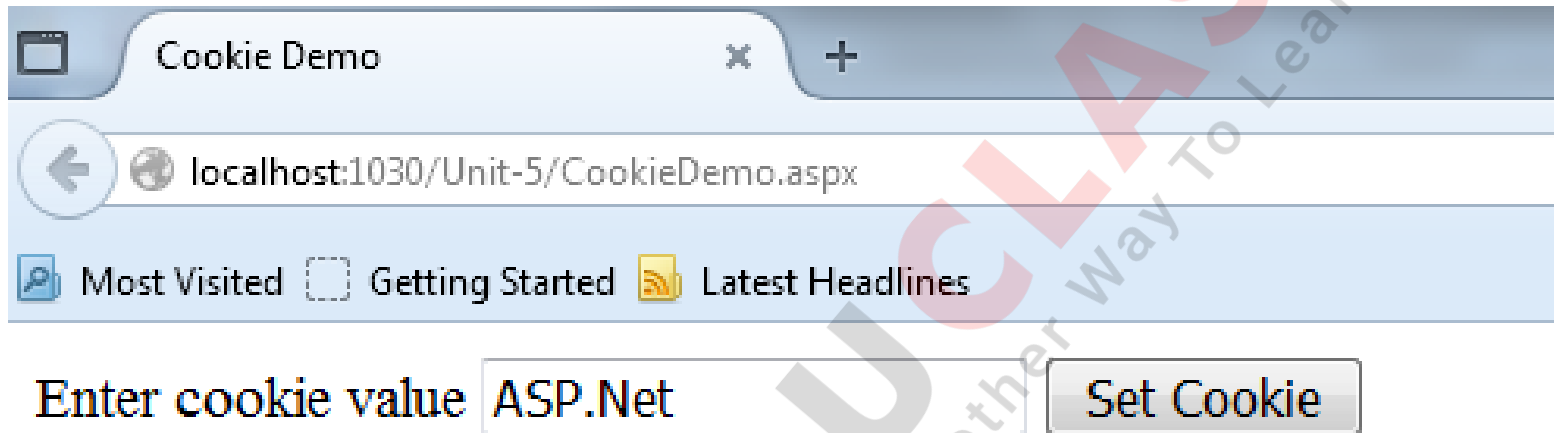
# Cookies

- CookieDemo.aspx.cs

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

```
public partial class CookieDemo : System.Web.UI.Page  
{  
    protected void Page_Load(object sender, EventArgs e)  
    {  
  
    }  
    protected void btnSetCookie_Click(object sender, EventArgs e)  
    {  
        Response.Cookies["message"].Value = txtCookieValue.Text;  
    }  
}
```

# Cookies



Cookie Demo

localhost:1030/Unit-5/CookieDemo.aspx

Most Visited Getting Started Latest Headlines

Enter cookie value

- Cookie names are case-sensitive. Setting a cookie named message is different from setting a cookie named Message.
- This example creates a Session Cookie. This cookie will expire immediately when we end the session.

# Creating Persistent Cookies

- If you want to create a ***persistent cookie***, you need to specify an expiration date for the cookie.
- The following example sets a Persistent Cookie:
- The below example tracks the number of times that you request the page. A persistent cookie named “counter” tracks page requests. The counter cookie’s Expires property is set to 2 years in the future.

```
<%@ Page Language="C#" %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```



# Persistent Cookies

```
<script runat="server">
void Page_Load()
{
    // Get current value of cookie
    int counter = 0;
    if (Request.Cookies["counter"] != null)
        counter = Int32.Parse(Request.Cookies["counter"].Value);

    // Increment counter
    counter++;

    // Add persistent cookie to browser
    Response.Cookies["counter"].Value = counter.ToString();
    Response.Cookies["counter"].Expires = DateTime.Now.AddYears(2);

    // Display value of counter cookie
    lblCounter.Text = counter.ToString();
}
</script>
```

# Persistent Cookies

//create a cookie

```
HttpCookie myCookie = new HttpCookie("myCookie");
```

//Add key-values in the cookie

```
myCookie.Values.Add("userid", objUser.id.ToString());
```

//set cookie expiry date-time. Made it to last for next 12 hours.

```
myCookie.Expires = DateTime.Now.AddHours(12);
```

//Most important, write the cookie to client.

```
Response.Cookies.Add(myCookie);
```

# Persistent Cookie

```
<html xmlns="http://www.w3.org/1999/xhtml" >
  <head id="Head1" runat="server">
    <title>Set Persistent Cookie</title>
  </head>

  <body>
    <form id="form1" runat="server">
      <div>
        You have visited this page
        <asp:Label id="lblCounter" Runat="server" /> times!
      </div>
    </form>
  </body>
</html>
```



# Reading Cookies

- The ***Response.Cookies collection*** is used to create and modify cookies. You use the ***Request.Cookies collection*** to retrieve a cookie's value.

```
void Page_Load()  
{  
    if (Request.Cookies["message"] != null)  
        lblCookieValue.Text = Request.Cookies["message"].Value;  
}
```



# Deleting Cookies

- To delete an existing cookie, you must set its expiration date to a date in the past.

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

```
{
```

```
    Response.Cookies["counter"].Expires = DateTime.Now.AddDays(-1);
```

```
}
```

Deleting all the cookies for a particular Domain: (DeleteAllCookies.aspx)

```
void Page_Load()
```

```
{
```

```
    string[] cookies = Request.Cookies.AllKeys;
```

```
    foreach (string cookie in cookies)
```

```
    {
```

```
        Response.Cookies[cookie].Expires = DateTime.Now.AddDays(-1);
```

```
    }
```

```
}
```

# Multivalued Cookies

- According to the cookie specifications, browsers should not store more than 20 cookies from a single domain.
  - To overcome this limitation we can use **Multivalued Cookies**
  - ***A multivalued cookie is a single cookie that contains subkeys. You can create as many sub keys as you need.***
  - ***Following example creates a multivalued cookie named preferences.***
- The preferences cookie stores a first name, last name, and favourite colour.

# Creating Multivalued Cookies

```
<%@ Page Language="C#" %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<script runat="server">
```

```
    void btnSubmit_Click(Object s, EventArgs e)
```

```
{
```

```
    Response.Cookies["preferences"]["firstName"] = txtFirstName.Text;
```

```
    Response.Cookies["preferences"]["lastName"] = txtLastName.Text;
```

```
    Response.Cookies["preferences"]["favoriteColor"] = txtFavoriteColor.Text;
```

```
    Response.Cookies["preferences"].Expires = DateTime.MaxValue;
```

```
}
```

```
</script>
```

```

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Set Cookie Values</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label id="lblFirstName" Text="First Name:" Runat="server" />    <br />
            <asp:TextBox id="txtFirstName" Runat="server" />    <br /><br />

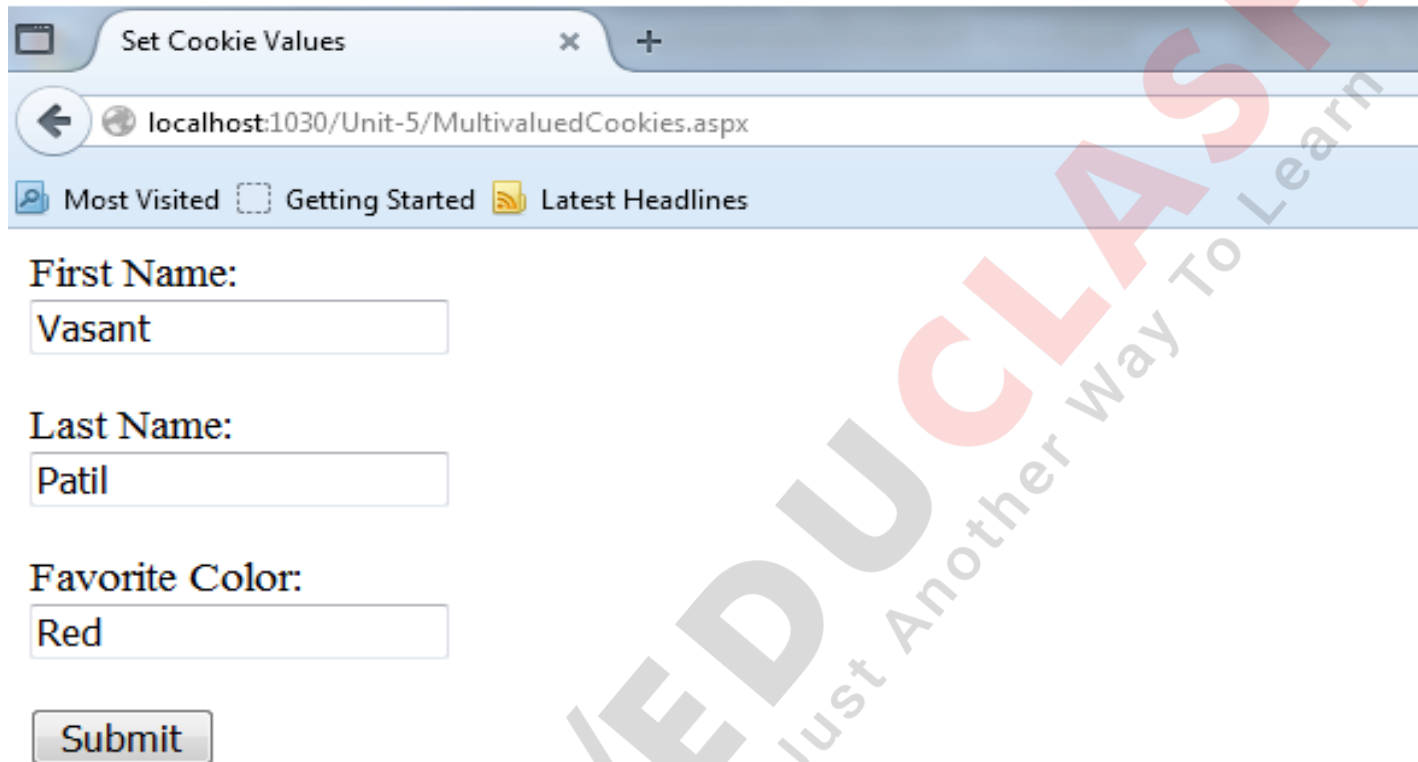
            <asp:Label id="lblLastName" Text="Last Name:" Runat="server" />    <br />
            <asp:TextBox id="txtLastName" Runat="server" />    <br /><br />

            <asp:Label id="lblFavoriteColor" Text="Favorite Color:" Runat="server" />    <br />
            <asp:TextBox id="txtFavoriteColor" Runat="server" />    <br /><br />

            <asp:Button id="btnSubmit" Text="Submit" OnClick="btnSubmit_Click" Runat="server"
        />
        </div>
    </form>

```

# Multivalued Cookies



The screenshot shows a web browser window with a single tab titled 'Set Cookie Values'. The address bar displays 'localhost:1030/Unit-5/MultivaluedCookies.aspx'. Below the address bar is a navigation bar with links: 'Most Visited', 'Getting Started', and 'Latest Headlines'. The main content area contains a form with three text input fields and a submit button. The first field is labeled 'First Name:' and contains the text 'Vasant'. The second field is labeled 'Last Name:' and contains the text 'Patil'. The third field is labeled 'Favorite Color:' and contains the text 'Red'. A 'Submit' button is located below the third field. A large, diagonal watermark reading 'EDUCLASH Just Another Way To Learn' is overlaid on the form.

Set Cookie Values

localhost:1030/Unit-5/MultivaluedCookies.aspx

Most Visited Getting Started Latest Headlines

First Name:  
Vasant

Last Name:  
Patil

Favorite Color:  
Red

Submit

When you submit the page in Listing 28.7, the following HTTP header is sent to the browser:

***Set-Cookie: preferences=firstName=Vasant&lastName=Patil&favoriteColor=Red; expires=Fri, 31-Dec-9999 23:59:59 GMT; path=/***

# Reading Multivalued Cookies

- To read the values from the Multivalued Cookies:
- Following program illustrates how we can read values from multivalued cookies:

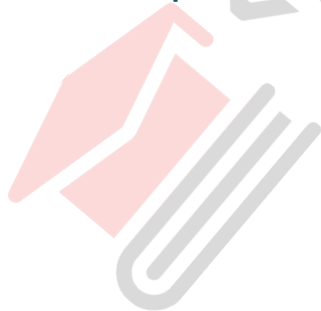
```
<%@ Page Language="C#" %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"  
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<script runat="server">  
    void Page_Load()  
    {  
        if (Request.Cookies["preferences"] != null)  
        {  
            lblFirstName.Text = Request.Cookies["preferences"]["firstName"];  
            lblLastName.Text = Request.Cookies["preferences"]["lastName"];  
            lblFavoriteColor.Text = Request.Cookies["preferences"]["favoriteColor"];  
        }  
    }  
</script>
```

# Multivalued Cookies

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
  <title>Get Cookie Values</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      First Name:
      <asp:Label id="lblFirstName" Runat="server" />
      <br />
      Last Name:  <asp:Label id="lblLastName" Runat="server" />
      <br />
      Favorite Color: <asp:Label id="lblFavoriteColor" Runat="server" />
    </div>
  </form>
</body>
</html>
```





# Multivalued Cookies

You can use the `HttpCookie.HasKeys` property to detect whether a cookie is a normal cookie or a multivalued cookie.



# Session State

- Like cookies, items stored in Session state are scoped to a particular user. (i.e. Sessions are User Specific)
- Unlike cookies, Session state has no size limitations.
- Unlike cookies, Session state can represent more complex objects than simple strings of text.
- You add items to Session state by using the ***Session*** object.
- For example to add a new item named message to Session state that has the value Hello World!,
  - ***Session["message"] = "Hello World!";***
- To retrieve the value of a session we can write:
  - ***lblMessage.Text = Session["message"].ToString();***

# Session State

- When you use Session state, a session cookie named **ASP.NET\_SessionId** is added to your browser automatically. This cookie contains a unique identifier. It is used to track you as you move from page to page.
- When you add items to the Session object, the items are stored on the web server and not the web browser.
- The ASP.NET\_SessionId cookie associates the correct data with the correct user.
- By default, if cookies are disabled, Session state does not work. You don't receive an error, but items that you add to Session state aren't available when you attempt to retrieve them in later page requests.
- (Hence to overcome this problem we can use ***Cookieless Session state*** )
- By default, ASP.NET Framework assumes that a user has left the website when the user has not requested a page for more than 20 minutes. At that point, any data stored in Session state for the user is discarded.

# HttpSessionState

- **HttpSessionState** class provides access to session-state values as well as session-level settings.

**This HttpSessionState class supports the various properties:**

- **CookieMode**—Enables you to specify whether cookieless sessions are enabled. Possible values are AutoDetect, UseCookies, UseDeviceProfile, and UseUri.
- **Count**—Enables you to retrieve the number of items in Session state.
- **IsCookieless**—Enables you to determine whether cookieless sessions are enabled.
- **IsNewSession**—Enables you to determine whether a new user session was created with the current request.

# Cookieless SessionState

- You enable cookieless sessions by modifying the sessionState element in the web configuration file.
- ***The sessionState element in web.config file includes a cookieless attribute that accepts the following values:***
  - **AutoDetect**—The Session ID is stored in a cookie when a browser has cookies enabled. Otherwise, the cookie is added to the URL.
  - **UseCookies**—The Session ID is always stored in a cookie (the default value).
  - **UseDeviceProfile**—The Session ID is stored in a cookie when a browser supports cookies. Otherwise, the cookie is added to the URL.
  - **UseUri**—The Session ID is always added to the URL.

# HttpSessionState

The HttpSessionState object also supports the following methods:

- **Abandon**—Enables you to end a user session.
- **Clear**—Enables you to clear all items from Session state.
- **Remove**—Enables you to remove a particular item from Session state.
- The Abandon() method enables you to end a user session programmatically.
- For example, you might want to end a user session automatically when a user logs out from your application to clear away all of a user's session state information.



# Handling Session Events

There are ***two events related to Session state*** that you can ***handle in the Global.asax*** file:

***Session Start*** and ***Session End***.

- The Session Start event is raised whenever a new user session begins.
- The Session End event is raised when a session ends.
- A session comes to an end when it times out because of user inactivity or when it is explicitly ended with the method `Session.Abandon()`.
- You can handle the Session End event, for example, when you want to automatically save the user's shopping cart to a database table.

**Note:** The `Session_End` event is raised only when the sessionstate mode is set to `InProc` in the `Web.config` file.

# Handling Session Events

- Program to demonstrate Session events: Finding number of active sessions at a time. (That is number of users accessing the website at a particular time).

**Global.asax**

```
<%@ Application Language="C#" %>
```

```
<script runat="server">
```

```
    void Application_Start(object sender, EventArgs e)
```

```
    {
```

```
        Application["SessionCount"] = 0;
```

```
    }
```

```
    void Session_Start(object sender, EventArgs e)
```

```
    {
```

```
        Application.Lock();
```

```
        int count = (int)Application["SessionCount"];
```

```
        Application["SessionCount"] = count + 1;
```

```
        Application.Unlock();
```

```
    }
```

```
    void Session_End(object sender, EventArgs e)
```

```
    {
```

```
        Application.Lock();
```

```
        int count = (int)Application["SessionCount"];
```

```
        Application["SessionCount"] = count - 1;
```

```
        Application.Unlock();
```

```
    }
```

```
</script>
```

FB/TW: @educrashco

[Vipin Dubey]



# Handling Session Events

The Global.asax file tracks the number of active sessions.

- Whenever a new session begins, the Session Start event is raised and the SessionCount variable is incremented by one.
- When a session ends, the Session End event is raised and the SessionCount variable is decremented by one.

*Here the SessionCount variable is stored in **Application state**, which contains items shared among all users of a web application.*

*The Application object is locked before it is modified.*

*You must lock and unlock the Application object because multiple users could potentially access the same item in Application state at the same time.*

# Global.asax File

- The ***Global.asax file, also known as the ASP.NET application file***, is an optional file that contains code for responding to application-level events.
- ***The Global.asax file resides in the root directory of an ASP.NET-based application.***
- At run time, Global.asax is parsed and compiled into a dynamically generated .NET Framework class derived from the [HttpApplication](#) base class.
- The Global.asax file itself is configured so that any direct URL request for it is automatically rejected; external users cannot download or view the code written within it.

# Global.asax file

- ***The Global.asax file is optional.*** If you do not define the file, the ASP.NET page framework assumes that you have not defined any application or session event handlers.
- ***If the Global.asax file or Web.config file for an ASP.NET application is modified, the application will be restarted.*** If the current session-state mode is [InProc](#), any values stored in application state or session state will be lost.



# Web.Config file

- ASP.NET configuration data is stored in XML text files, each named Web.config. Web.config files can appear in multiple directories in ASP.NET applications.
- ASP.NET *Web.config* allows you to define or revise the configuration settings at the time of developing the application or at the time of deployment or even after deployment.

## ***Some Important points about web.config:***

- *Web.config* files are stored in XML format.
- You can have any number of *Web.config* files for an application. Each *Web.config* applies settings to its own directory and all the child directories below it.

# Web.Config file

- All the *Web.config* files inherit the root *Web.config* file available at the following location  
*systemroot\Microsoft.NET\Framework\versionNumber\CONFIG\Web.config* location.
- IIS is configured in such a way that it prevents the *Web.config* files access from the browser.



# Machine.Config file

## Machine.config Vs Web.config

- The machine.config file is the master configuration file on your system with a lot of default settings. Web.config is the file for the local settings to be applied for a website which store configuration data in XML format. The settings of Machine.config file are applied to the whole asp.net applications on your server whereas the settings made in the Web.config file are applied to that particular web application only.
- Each .NET Framework version has only one machine.config file. Whereas, each web application has its own web.config file. Directories inside a web application can also have web.config files too.
- The machine.config file is at the highest level in the configuration hierarchy while Web.config file is to override the settings from the machine.config file.

# Machine.Config file

- The machine.config would be to share values between many applications on the server. While Web.config files contain application specific items such as database connection strings.
- The machine.config file will automatically installed when you install Visual Studio.Net and it exist exists in the c:\windows\microsoft.net\framework\version\config folder whereas web.config will automatically created when you create an ASP.Net web application project.
- Machine.config is configuration file for all the application in the IIS, but Web.config is a configuration file for a particular application or folder.

# Controlling when a Session Times out

- By default, ASP.NET Framework assumes that a user has left an application after 20 minutes have passed without the user requesting a page. Then the Session state is cleared.
- But there may be a situation in which we may want to change the Session Timeout value.
- You can specify the Session timeout in the web configuration file or you can set the Session timeout programmatically.

## Web.Config

```
<?xml version="1.0"?>  
  <configuration>  
    <system.web>  
      <sessionState timeout="60" />  
    </system.web>  
  </configuration>
```



# Using Cookieless Session State

- By default, Session state depends on cookies.
- The ASP.NET Framework uses the **ASP.NET\_SessionId** cookie to identity a user across page requests so that the correct data can be associated with the correct user.
- If a user disables cookies in the browser, Session state doesn't work.
- *If you want Session state to work even when cookies are disabled, you can take advantage of cookieless sessions.*
- *When cookieless sessions are enabled, a user's session ID is added to the page URL.*



# Cookieless SessionState

The following web configuration file enables cookieless sessions by assigning the value AutoDetect to the cookieless attribute.

Web.Config

```
<?xml version="1.0"?>  
<configuration>  
  <system.web>  
    <sessionState cookieless="AutoDetect"  
      regenerateExpiredSessionId="true" />  
  </system.web>  
</configuration>
```

Note: When you enable cookieless session state, you should also enable 'regenerateExpiredSession' attribute because it can help prevent users from inadvertently sharing session state.

Use relative URLs when linking between pages in your application, while using Cookieless SessionState.

# Setting Up an Out-of-Process Session State

## **Why to *set SessionState in out-of-process?***

By default, Session state is stored in memory in the same process as the ASP.NET process. There are two significant disadvantages to storing Session state in the ASP.NET process.

- First, in-process Session state is fragile. ***If your application restarts, all Session state is lost.***

***A number of different events can cause an application restart.*** For example, modifying the web configuration file or errors in your application both can cause an application restart.

- ***Second, in-process Session state is not scalable.***

When Session state is stored in-process, it is stored on a particular web server. In other words, you can't use in-process Session state with a web farm.

# Out-of-Process SessionState

You can store Session state in an alternative location by modifying the Session state mode.

***Different Session State modes are:***

- ***Off***— Disables Session state.
- ***InProc***— Stores Session state in the same process as the ASP.NET process.
- ***StateServer***— Stores Session state in a Windows NT process, which is different from the ASP.NET process.
- ***SQLServer***— Stores Session state in a SQL Server database.
- ***Custom***— Stores Session state in a custom location.



# Configuring SessionState Store

- By **default**, the **Session state mode** is set to the value **InProc**. This is done for performance reasons. In-process Session state results in the best performance. However, it sacrifices robustness and scalability.
- When you set the Session state mode to either **StateServer** or **SQLServer**, you get robustness and scalability at the price of performance. Storing Session state out-of-process results in worse performance because Session state information must be passed back and forth over your network.
- Finally, you can **create a custom Session state store provider** by inheriting a new class from the SessionStateStoreProviderBase class. In that case, you can store Session state any place that you want.



# Configuring 'StateServer' SessionState

- When you enable State Server Session state, Session state information is stored in a separate Windows NT Service. The Windows NT Service can be located on the same server as your web server, or it can be located on another server in your network.
- You must complete the following two steps to use State Server Session state:

## **1. Start the ASP.NET State Service.**

You can start the ASP.NET State Service by opening the Services applet located at Start, Administrative Tools.

## **2. Configure your application to use the ASP.NET State Service.**

The below mentioned web configuration file enables State Server Session State.

# Configuring 'StateServer' SessionState

```
<?xml version="1.0"?>  
<configuration>  
  <system.web>  
    <sessionState mode="StateServer"  
      stateConnectionString="tcpip=localhost:42424"  
      stateNetworkTimeout="10" />  
  </system.web>  
</configuration>
```



# 'StateServer' SessionState

The previous web configuration file modifies three attributes of the sessionState element.

- First, the ***mode attribute is set to the value StateServer.***
- Second, the ***stateConnectionString attribute is used to specify the location of the ASP.NET State Server.*** Here a connection is created to the local server on port 42424.
- Finally, the ***stateNetworkTimeout attribute specifies a connection timeout in seconds.***

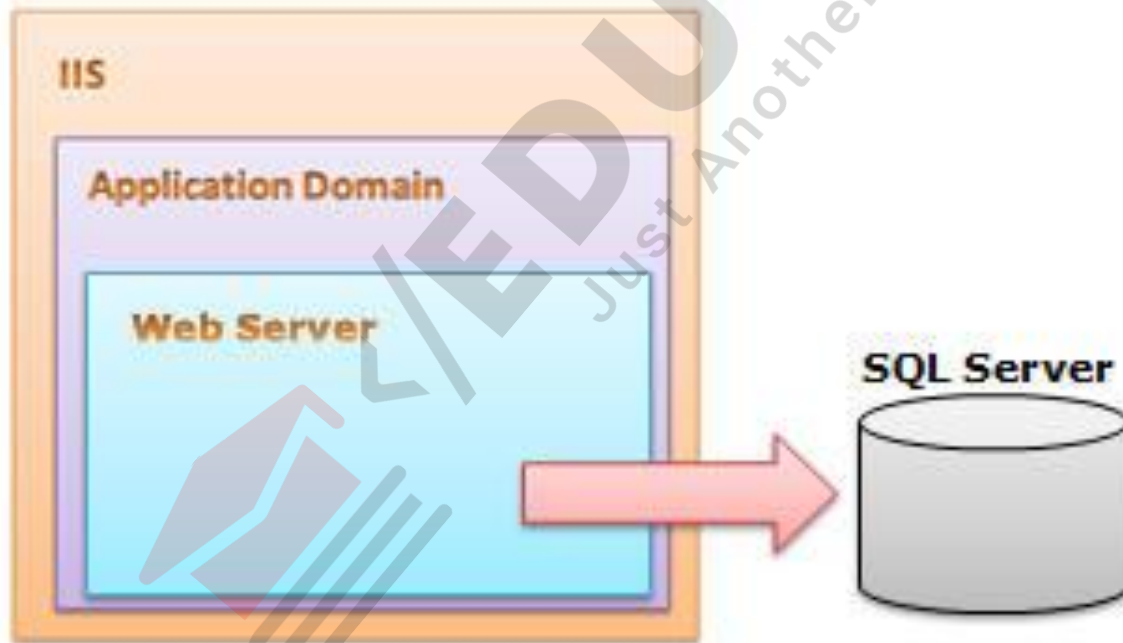
After you complete these configuration steps, Session state information is stored in the ASP.NET State Server automatically. You don't need to modify any of your application code when you switch to out-of-process Session state.



# 'SQLServer' Session Mode

This session mode provide us more secure and reliable session management in ASP.NET.

In this **session** mode, **session** data is serialized and stored in A SQL Server database.



# When to use 'SQLServer' Session Mode

- SQL Server **session** mode is a more reliable and secure **session** state management.
- It keeps data in a centralized location (database).
- We should use the SQLServer **session** mode when we need to implement **session** with more security.
- This is the perfect mode for web farm and web garden scenarios.
- We can use SQLServer **session** mode when we need to share **session** between two different applications.



## Configuration for SQLServer Session Mode

- In SQLServer **session** mode, we store **session** data in SQL Server, so we need to first provide the database connection string in *web.config*. The `sqlConnectionString` attribute is used for this.
- After we setup the connection string, we need to configure the SQL Server

Now configure your SQL Server using the *aspnet\_regsql* command.

**Step 1:** From command prompt, go to your Framework version directory. E.g.: `c:\windows\microsoft.net\framework\<version>`.

**Step 2:** Run the *aspnet\_regsql* command with the following parameters:

```
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727>aspnet_regsql -S localhost -U sa -P india@123 -ssadd -sstype p
```

Parameters	Description
-ssadd	Add support for SQLServer mode <b>session</b> state.
-sstype p	P stands for Persisted. It persists the <b>session</b> data on the server.
-S	Server name.
-U	User name.
-P	Password.

After you run the command, you will get the following message:

```
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727>aspnet_regsql -S localhost -U sa -
P india@123 -ssadd -sstype p

Start adding session state.

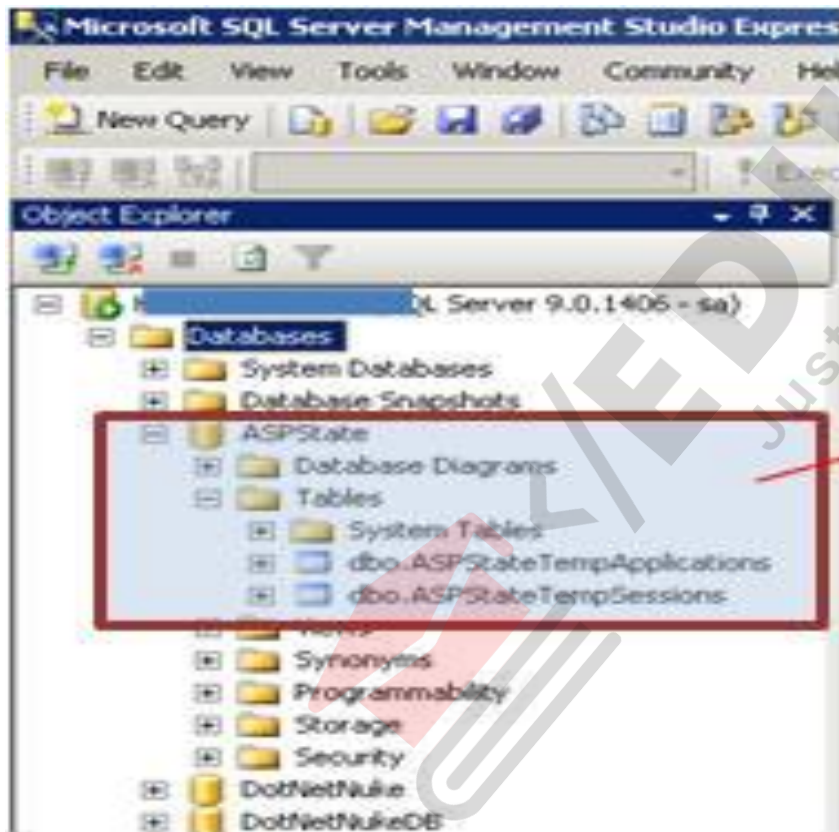
.....

Finished.
```

**Step 3:** Open SQL Server Management Studio, check if a new database ASPState has been created, and there should be two tables:

*ASPStateTempApplications*

*ASPStateTempSessions*



Database  
and Table  
for storing  
Session  
data

Open the *ASPStateTempSessions* table from SQL Server Management Studio.. here is your **session** data:

	SessionId	Created	Expires	LockDate	LockDateLocal	LockCookie	Timeout
▶	wh2w/vn635b5d4c	1/13/2009 2:13:...	1/13/2009 2:33:...	1/13/2009 2:13:...	1/13/2009 7:43:...	1	20
✱	NULL	NULL	NULL	NULL	NULL	NULL	NULL

# Contd...

## Advantages:

- **Session** data not affected if we restart IIS.
- The most reliable and secure **session** management.
- It keeps data located centrally, is easily accessible from other applications.
- Very useful in web farms and web garden scenarios.

## Disadvantages:

- Processing is very slow in nature.
- Object serialization and de-serialization creates overhead for the application.
- As the **session** data is handled in a different server, we have to take care of SQL Server. It should be always up and running.

# Thank You !!!!

