Cookies is a small piece of information stored on the client machine. This file is located on client machines "C:\Document and Settings\Currently\_Login user\Cookie" path.  Its is used to store user preference information like Username, Password,City and PhoneNo etc on client machines. We need to import namespace called  Systen.Web.HttpCookie before we use cookie.  
 

Type of Cookies

1. Persist Cookie - A cookie has not have expired time Which is called as Persist Cookie
2. Non-Persist Cookie - A cookie has expired time Which is called as Non-Persist Cookie

How to create a cookie?

It is really easy to create a cookie in the Asp.Net with help of Response object or HttpCookie

Example 1

HttpCookie userInfo = **new** HttpCookie("userInfo");

userInfo["UserName"] = "Annathurai";

userInfo["UserColor"] = "Black";

userInfo.Expires.Add(**new** TimeSpan(0, 1, 0));

Response.Cookies.Add(userInfo);

Example 2

1. Response.Cookies["userName"].Value = "Annathurai";
2. Response.Cookies["userColor"].Value = "Black";

How to retrieve from cookie?

It is easy way to retrieve cookie value form cookies with the help of Request object.

Example 1

**string** User\_Name = **string**.Empty;

**string** User\_Color = **string**.Empty;

User\_Name = Request.Cookies["userName"].Value;

User\_Color = Request.Cookies["userColor"].Value;

Example 2

**string** User\_name = **string**.Empty;

**string** User\_color = **string**.Empty;

HttpCookie reqCookies = Request.Cookies["userInfo"];

**if** (reqCookies != **null**)

{

    User\_name = reqCookies["UserName"].ToString();

    User\_color = reqCookies["UserColor"].ToString();

}

When we make a request from the client to web server, the web server process the request and give a lot of information with big pockets which will have Header information, Metadata, cookies etc., Then repose object can do all the things with browser.

Cookie's common property

1. Domain => Which is used to associate cookies to domain.
2. Secure  => We can enable secure cookie to set true(HTTPs).
3. Value    => We can manipulate individual cookie.
4. Values  => We can manipulate cookies with key/value pair.
5. Expires => Which is used to set expire date for the cookies.

Advantages of Cookie

1. Its clear text so user can able to read it.
2. We can store user preference information on the client machine.
3. Its easy way to maintain.
4. Fast accessing.

Disadvantages of Cookie

1. If user clear cookie information we can't get it back.
2. No security.
3. Each request will have cookie information with page.

How to clear the cookie information?

1. We can clear cookie information from client machine on cookie folder
2. To set expires to cookie object
   1. userInfo.Expires = DateTime.Now.AddHours(1);

It will clear the cookie with one hour duration.

# **State Management In ASP.NET**

This article is all about how to maintain, clear or hold the states of your pages in ASP.NET applications. In this article I tried to briefly summarize the concept of State Management but I'll include Client-Side State Management only.

**Stateless meaning:**

It does not keep track of configuration settings, transaction information or any other data for the next session. When a program "does not maintain state" (is stateless) or when the infrastructure of a system prevents a program from maintaining state, it cannot take information about the last session into the next, such as settings the user chose or conditions that arose during processing.

**HTTP:**

The HTTP protocol, which is the communications vehicle for Web transactions, is stateless. After a Web page is delivered to the user, the connection is closed. Counter measures, such as the use of cookies, have been developed to maintain the state of a user moving from page to page on a Web site.

**OR it can be simple web definition:**

A protocol is stateless if there is no relation between subsequent request-response pairs. The server can handle each request uniquely and does not have to keep a session state for the client.

**Agenda**

The agenda of this article will be as follows,

* State Overview
* State Introduction
* State Outline
* State Management Types  
    
  + Client-side Management
  + Server-side Management
* State Management Scenario
* State Management Techniques  
    
  + Client-side Techniques



* + - View
    - Hidden
    - Cookies
    - Control State
    - Query Strings
  + Server-side Techniques



* + - Session State
    - Application State

State Overview

As we all know, browsers are generally stateless.



Now the question arises here, what does stateless actually mean?

Stateless means, whenever we visit a website, our browser communicates with the respective server depending on our requested functionality or the request. The browser communicates with the respective server using the HTTP or HTTPs protocol.

But after that response, what's next or what will happen when we visit that website again after closing our web browser?



n this case HTTP/HTTPs doesn't remember what website or URL we visited, or in other words we can say it doesn't hold the state of a previous website that we visited before closing our browser, that is called stateless.



Now I guess you have at least an idea of what state and stateless actually means.

So our browsers are stateless.

State Introduction

In this article I'll try to give you a feel of state and why we need states and State Management in ASP.NET. I'll take you through several State Management techniques along with their respective general case examples.

State Outline

As I said in the beginning, HTTP is a stateless protocol. It just cleans up or we can say removes all the resources/references that were serving a specific request in the past. These resources can be:



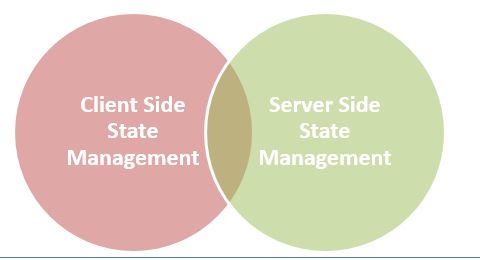
* Objects



* Allocated Memory
* Sessions ID's
* Some URL info  
  and so on.

State Management Types

In ASP.NET there are the following 2 State Management methodologies:



**Client-Side State Management**

Whenever we use Client-Side State Management, the state related information will directly get stored on the client-side. That specific information will travel back and communicate with every request generated by the user then afterwards provides responses after server-side communication.



This architecture is something like the following,



**Server-Side State Management**



Server-Side State Management is different from Client-Side State Management but the operations and working is somewhat the same in functionality. In Server-Side State Management all the information is stored in the user memory. Due to this functionality there is more secure domains at the server side in comparison to Client-Side State Management.



The structure is something like the following,



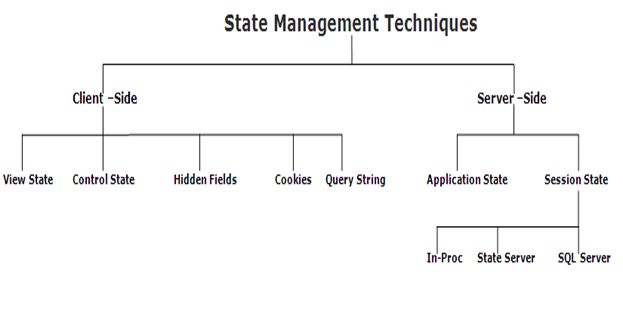
State Management Scenario

It will be a little difficult to directly evaluate what will be better for our application. We cannot directly say that we will use client-side or server-side architecture of State Management.

State Management Techniques

State Management techniques are based on client side and server side. Their functionality differs depending on the change in state, so here is the hierarchy:





**Client-side | Techniques**

Client-Side State Management techniques are,

* View State
* Hidden field
* Cookies
* Control State
* Query Strings

**Server-side | Technique**

Server-Side State Management techniques are,

* Session State
* Application State

Now I am defining each and every technique in detail with their reference example.

View State

In general we can say it is used for storing user data in ASP.NET, sometimes in ASP.NET applications the user wants to maintain or store their data temporarily after a post-back.. In this case VIEW STATE is the most used and preferred way of doing that.

This property is enabled by default but we can make changes depending on our functionality, what we need to do is just change the EnableViewState value to either TRUE for enabling it or FALSE for the opposite operation.

  
Figure: [View State Management]

1. // Page Load Event
2. **protected** **void** Page\_Load(**object** sender, EventArgs e)
3. {
4. **if** (IsPostBack)
5. {
6. **if** (ViewState["count"] != **null**)
7. {
8. **int** ViewstateVal = Convert.ToInt32(ViewState["count"]) + 1;
9. View.Text = ViewstateVal.ToString();
10. ViewState["count"]=ViewstateVal.ToString();
11. }
12. **else**
13. {
14. ViewState["count"] = "1";
15. }
16. }
17. }
19. // Click Event
20. **protected** **void** Submit(**object** sender, EventArgs e)
21. {
22. View.Text=ViewState["count"].ToString();
23. }

**Points to Remember**

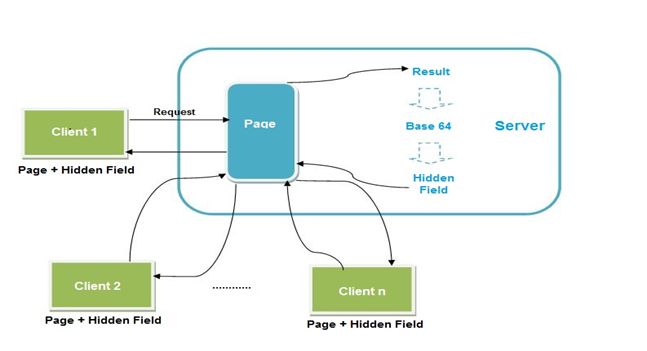
Some of the features of view state are:

* It is page-level State Management
* Used for holding data temporarily
* Can store any type of data
* Property dependent

Hidden Field

A hidden field is used for storing small amounts of data on the client side. In most simple words it's just a container of some objects but their result is not rendered on our web browser. It is invisible in the browser.

It stores a value for the single variable and it is the preferable way when a variable's value is changed frequently but we don't need to keep track of that every time in our application or web program.

   
Figure: [Hidden Field Management]

1. // Hidden Field

**int** newVal = Convert.ToInt32(HiddenField1.Value) + 1;

HiddenField1.Value = newVal.ToString();

Label2.Text = HiddenField1.Value; 

**Points to Remember**

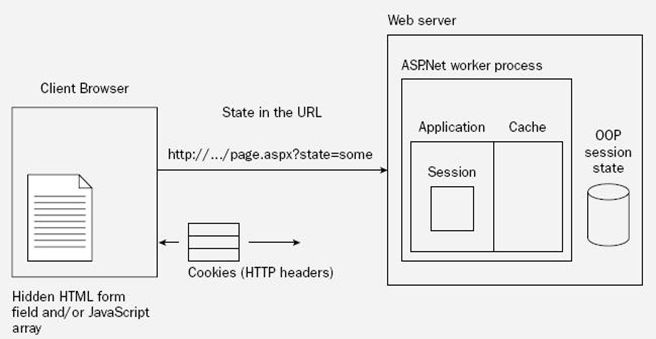
Some features of hidden fields are:

* Contains a small amount of memory
* Direct functionality access

Cookies

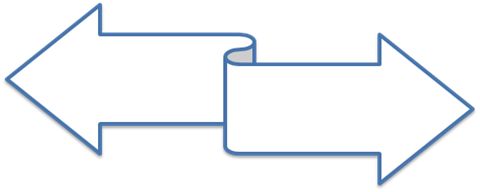
A set of Cookies is a small text file that is stored in the user's hard drive using the client's browser. Cookies are just used for the sake of the user's identity matching as it only stores information such as sessions id's, some frequent navigation or post-back request objects.

Whenever we get connected to the internet for accessing a specific service, the cookie file is accessed from our hard drive via our browser for identifying the user. The cookie access depends upon the life cycle or expiration of that specific cookie file.

  
Figure: [Cookie Management]

1. **int** postbacks = 0;
2. **if** (Request.Cookies["number"] != **null**)
3. {
4. postbacks = Convert.ToInt32(Request.Cookies["number"].Value) + 1;
5. }
6. // Generating Response
7. **else**
8. {
9. postbacks = 1;
10. }
11. Response.Cookies["number"].Value = postbacks.ToString();
13. Result.Text = Response.Cookies["number"].Value;

**Cookie | Types**

****

**Persistent Cookie**

Cookies having an expiration date is called a persistent cookie. This type of cookie reaches their end as their expiration dates comes to an end. In this cookie we set an expiration date.

Response.Cookies["UserName"].Value = "Abhishek";

Response.Cookies["UserName"].Expires = DateTime.Now.AddDays(1);

HttpCookie aCookie = **new** HttpCookie("Session");

aCookie.Value = DateTime.Now.ToString();

aCookie.Expires = DateTime.Now.AddDays(1);

Response.Cookies.Add(aCookie); 

**Non-Persistent Cookie**

Non-persistent types of cookies aren't stored in the client's hard drive permanently. It maintains user information as long as the user access or uses the services. Its simply the opposite procedure of a persistent cookie.

1. HttpCookie aCookie = **new** HttpCookie("Session");
2. aCookie.Value = DateTime.Now.ToString();
3. aCookie.Expires = DateTime.Now.AddDays(1);
4. Response.Cookies.Add(aCookie);

**Points to Remember**

Some features of cookies are:

* Store information temporarily
* It's just a simple small sized text file
* Can be changed depending on requirements
* User Preferred
* Requires only a few bytes or KBs of space for creating cookies

Control State

Control state is based on the custom control option. For expected results from CONTROL STATE we need to enable the property of view state. As I already described you can manually change those settings.

**Points to Remember**

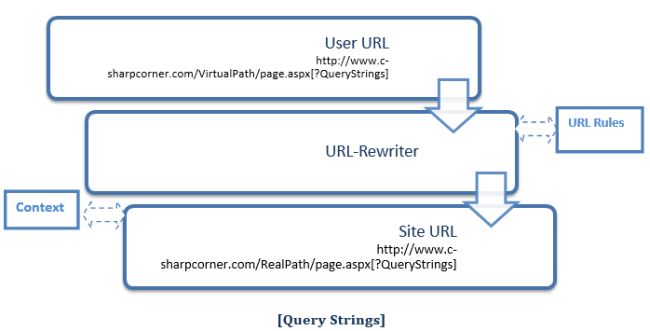
Some features of query strings are:

* Used for enabling the View State Property
* Defines a custom view
* View State property declaration
* Can't be modified
* Accessed directly or disabled

Query Strings

Query strings are used for some specific purpose. These in a general case are used for holding some value from a different page and move these values to the different page. The information stored in it can be easily navigated to one page to another or to the same page as well.



  
Figure: [Query Strings]



1. // Getting data
2. **if** (Request.QueryString["number"] != **null**)
3. {
4. View.Text = Request.QueryString["number"];
5. }
7. // Setting query string
8. **int** postbacks = 0;
10. **if** (Request.QueryString["number"] != **null**)
11. {
12. postbacks = Convert.ToInt32(Request.QueryString["number"]) + 1;
13. }
14. **else**
15. {
16. postbacks = 1;
17. }
19. Response.Redirect("default.aspx?number=" + postbacks);

**Points to Remember**

Some of the features are,

* It is generally used for holding values
* Works temporarily
* Switches info from one to another page
* Increase performance
* Uses real and virtual path values for URL routing

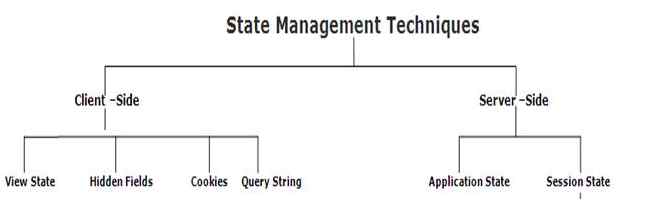
As we know that HTTP is stateless protocol, it means after every request data gets lost. Browser does not know about previous data and we cannot get earlier request data. So, basically the state of the page or control lost after every page request.

State Management is a technique or process through which we can maintain the page information between multiple requests for same or multiple pages. So, using this we can get the earlier page or control data on the same page or on different page at the time of next request.  
  
State Management techniques available in the following types:  
  
**Client Side State Management**  
It is used to store the state related information on the client side. So, every time to get data there is no need to hit the server. The main benefit is that you relieve the server from extra burden of storing the state related information. And you can save the server memory and utilize it in different task.  
There is one drawback to use client side state management; you can store the sensitive data like your password, bank details, etc. on the client side.  
  
The following techniques are used in client side state management:

1. Hidden Field
2. Cookie
3. View State
4. Query Strings

**Server Side State Management**  
It works same as client side state management technique work but it stores the data on server rather than client side. Web server stores the data in web server memory. It is safe and secure for your sensitive data.  
  
The following techniques are used in server side state management:

1. Session State
2. Application State

So, it is time to go in details about every technique.  
  
  
  
**Client Side- Technique**  
**Hidden Field**  
It is used to store the small data on the client side. It is invisible control that contains the small data like a single variable value in it. The data of hidden filed is never shown by the browser. If you want to see the value of hidden field, you can view by page source. There is not any encryption technique applied to encrypt the hidden field data. ViewState use hidden field to store the data internally.

1. //Storing in Hidden Field
2. **string** myname = "Mukesh Kumar";
3. HiddenField1.Value = myname.ToString();
4. lblUserName.Text = HiddenField1.Value;

**Cookie**  
It is nothing but a small text file which stores the small pieces of data using client’s browser. The main use of the cookie is that it is only for identifying the user who is visiting the web application by their specific data that is stored into the cookie.  
  
When cookie is persistent then it stores into the text file and when it is temporally based then it saves into the client browser. On the page request, server connects to cookie and gets the relevant information about visitors. You can take a simple example of cookie i.e login page.  
  
We can easily create, read and delete the cookie using the code.

1. Response.Cookies["UserName"].Value = "Mukesh Kumar";
2. Response.Cookies["UserName"].Expires = DateTime.Now.AddDays(2);
4. HttpCookie myCookie = **new** HttpCookie("Session");
5. myCookie.Value = DateTime.Now.ToString();
6. myCookie.Expires = DateTime.Now.AddDays(1);
7. Response.Cookies.Add(myCookie);

**ViewState**  
In ASP.NET MVC, ViewState is used to store the state of the control or object data or any data source, etc. It contains the stored data after page is posted. So, we can say, it is a technique to retain the value or data between multiple requests for same page. We cannot access the value of ViewState on other page.  
  
When we use the ViewState to store the page information or any control information then ViewSate hash the data into a string and saves it into the hidden field. If your data is too long or heavy then ViewState creates multiple hidden fields and save the data in chunk.



1. <input type="hidden" name="\_\_VIEWSTATE" id="\_\_VIEWSTATE” value=" /wEPDwUKMTIxNDIyOTM0Mg9kFgICdAw9kFgICAQ8PFgIeBFRleHQFEzQvNS8yMDA24IDEd6Mzc6MTEgUE1kZGROWHn/rt75XF/pMGdnqjqHlH66cdw==" />



ViewState is a dictionary object so our data is stored into the ViewState as key-value pair. EnableViewState property is used to enable or disable the ViewState. You can make it either True or False.



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



1. {
2. **if** (IsPostBack)
3. {
4. **if** (ViewState["UserName"] != **null**)
5. {
6. **string** userName = Convert.ToString(ViewState["UserName"]);
7. }
8. **else**
9. {
10. ViewState["UserName"] = "Guest";
11. }
12. }
13. }
14. **protected** **void** Submit(**object** sender, EventArgs e)
15. {
16. lblUserName.Text = ViewState["UserName"].ToString();
17. }

**Query String**  
Sometimes, it is required to send some specific information on the other page, so using the query string we can hold the value and send it to other page easily. It is mainly used to store the variables as parameters such as you can see that most of the websites use query string in search page. In that the terms entered by user is showed in the [browser](http://search.microsoft.com/results.aspx?mkt=en-US&setlang=en-US&q=hello+world).  
  
You can see above url, query string values appear into the url, so never pass some sensitive data into the query string. There is some limitation of query string, you cannot pass too much characters into the query string. There is some limit based on the browsers.  
  
Pass the query string values as in the following way.

1. **protected** **void** btnSend\_Click(**object** sender, EventArgs e)
2. {
3. Response.Redirect("Default2.aspx?UserId=" + txtUserId.Text + "&UserName=" + txtUserName.Text);
4. }

Get the query string values as in the following way:

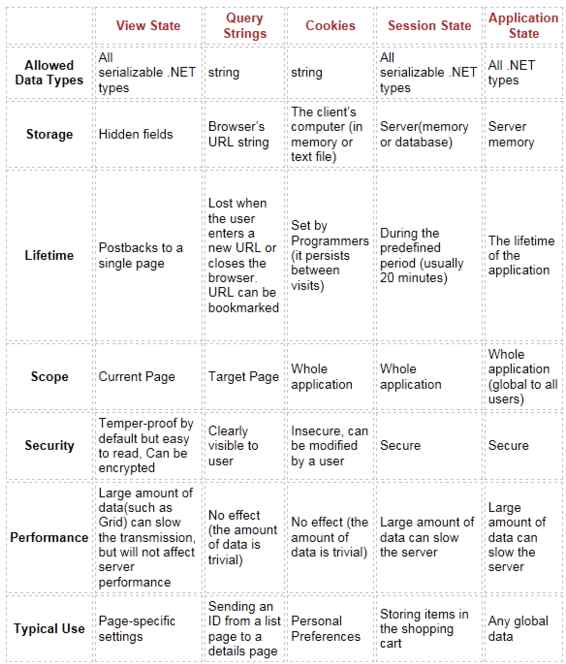
1. **protected** **void** Page\_Load(**object** sender, EventArgs e)
2. {
3. **if** (!IsPostBack)
4. {
5. lblUserId.Text = Request.QueryString["UserId"];
6. lblUserName.Text = Request.QueryString["UserName"];
7. }
8. }
9. **protected** **void** Page\_Load(**object** sender, EventArgs e)
10. {
11. **if** (!IsPostBack)
12. {
13. lblUserId.Text = Request.QueryString[0];
14. lblUserName.Text = Request.QueryString[1];
15. }
16. }

**Server Side- Technique**  
**Session State**  
In ASP.NET you can use the session object to store the value which is accessible throughout the application in single browser. Some time it is required to store the value which might be sensitive and it should be accessible on every page of the application, in that scenario you can go with session object for storing the data.  
  
Mostly you have seen that Session is used to store the user related information which needs to be accessed on every page for processing. The Session cannot be accessible outside of the browser where it has been created. Every browser creates own session.  
  
If you leave the application or your session is going to time out then you will not access the session data. It means, the session data lost after specific time period. So, again browser will create a new session for you. Actually, Session stores the data as key/value pair.

1. //global.asax
2. **void** Session\_Start(**object** sender, EventArgs e)
3. {
4. // Code that runs when a new session is started
5. Session["UserName"] = "Guest";
6. }
8. //store the session value
9. Session["UserName"] = Convert.ToString(txtLoginUser.Text);
11. //get the session value
12. lblUserName.Text = Session["UserName"].ToString();

**Application State**  
It is a global storage to store the value which will be accessible to all the pages. It also store the data into key/value pair. Server auto manages all the activity on application state data and does not expose it to client. The information stored into application state is accessible to all the page.  
  
So, application state is best place for storing such type of data which is useful for entire users who is using the same application. I want to confirm here that the application state data is not permanent. If you restart or shutdown your application then application state will lost.

1. //global.asax
2. **void** Application\_Start(**object** sender, EventArgs e)
3. {
4. Application["OnlineUser"] = 0;
5. }
7. //In web pages
8. Application.Lock();
9. Application["OnlineUser"] = Convert.ToInt32(Application["OnlineUser"]) + 1;
10. Application.UnLock();
12. lblOnlineUser.Text = Application["number"].ToString();

  
  
So, finally we can say client side as well as server side state management technique has their own importance and we can use all these as per requirement in our application.



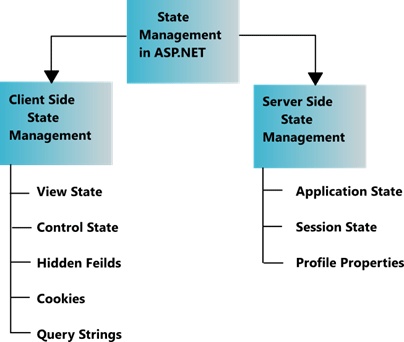
# **What Is View State And How It Works In ASP.NET**

## Background

A web application is stateless. That means that a new instance of a page is created every time when we make a request to the server to get the page and after the round trip our page has been lost immediately. It only happens because of one server, all the controls of the Web Page is created and after the round trip the server destroys all the instances. So to retain the values of the controls we use state management techniques.

## State Management Techniques

They are classified into the following 2 categories,



Now I am explaining what View State is.

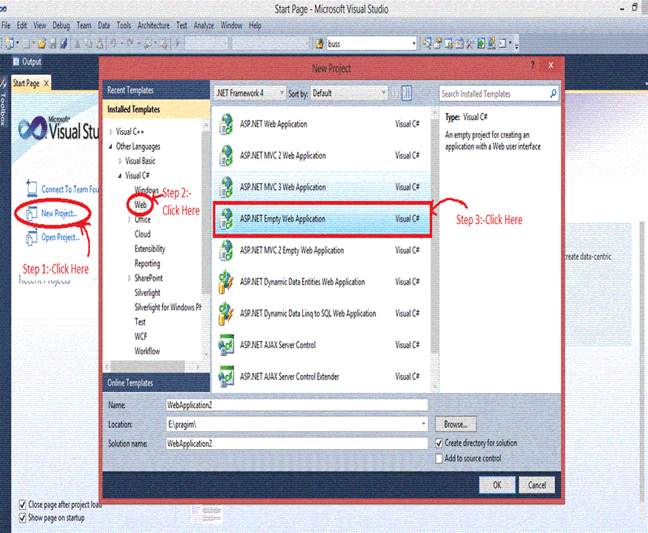
## View State

View State is the method to preserve the Value of the Page and Controls between round trips. It is a Page-Level State Management technique. View State is turned on by default and normally serializes the data in every control on the page regardless of whether it is actually used during a post-back.

Now I am showing you an example of what the problem is when we don't use view state.

**Step 1**

Open Visual Studio 2010.

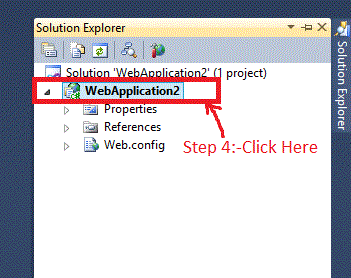


**Step 2**

Then click on "New Project" > "Web" >"ASP.NET Empty Web Application".

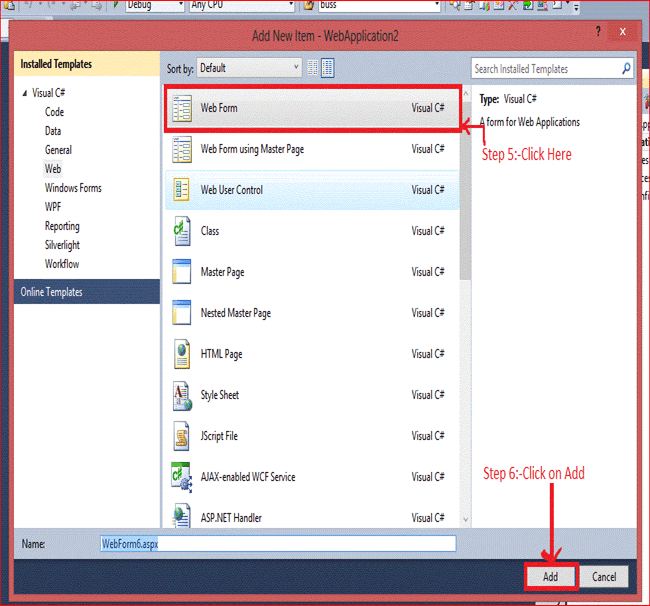
**Step 3**

Now click on Solution Explorer.



**Step 4**

Now right-click on the "ADD" > "New Item" > "Web Form" and add the name of the Web Form just like I did in WebForm6.aspx.



**Step 5**

After adding the WebForm6.aspx you will see the following code:

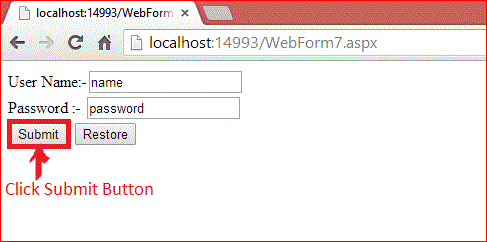
1. <%@ page language="C#" autoeventwireup="true" codebehind="WebForm6.aspx.cs" inherits="view\_state.WebForm6" %>
3. <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4. <html xmlns="http://www.w3.org/1999/xhtml">
5. <head runat="server">
6. <title></title>
7. </head>
8. <body>
9. <form id="form1" runat="server">
10. <div>
11. User Name:-<asp:textbox id="TextBox1" runat="server"></asp:textbox>
12. <br />
13. Password  :-<asp:textbox id="TextBox2" runat="server"></asp:textbox>
14. <br />
15. <asp:button id="Button1" runat="server" onclick="Button1\_Click" text="Submit" />
16. <asp:button id="Button3" runat="server" onclick="Button3\_Click" text="Restore" />
17. </div>
18. </form>
19. </body>
20. </html>

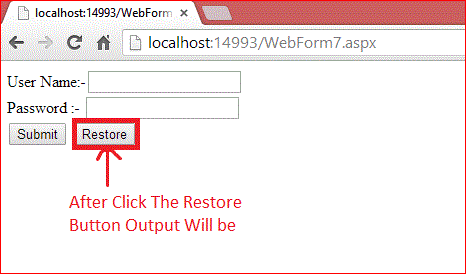
Now write the code as in the following,

1. //Declaration of a and b
2. **public** string a, b;
3. **protected** **void** Button1\_Click(object sender, EventArgs e)
4. {
5. //TextBox1 and TextBox2 Value is Assigning on the variable a and b
6. a = TextBox1.Text;
7. b = TextBox2.Text;
8. //after clicking on Button TextBox value Will be Cleared
9. TextBox1.Text = TextBox2.Text = string.Empty;
10. }
12. **protected** **void** Button3\_Click(object sender, EventArgs e)
13. {
14. //value of variable a and b is assingning on TextBox1 and Textbox2
15. TextBox1.Text = a;
16. TextBox2.Text = b;
17. }

**Outputft**

Now the output is,





It only happens because all the controls are classes and on the server all the Control Objects are created and then after the round trip the Page is returned to the client's browser in HTML format and the objects are destroyed at the server.

After the Submit button is clicked the value of user name and password is submitted to the server. We cannot restore the value again because after the postback the instance of the control is destroyed and on clicking of the Restore Button the server takes a new request and the server cannot restore the value of the TextBox.

## Features Of View State

These are the main features of view state,

1. Retains the value of the Control after post-back without using a session.
2. Stores the value of Pages and Control Properties defined in the page.
3. Creates a custom View State Provider that lets you store View State Information in a SQL Server Database or in another data store.

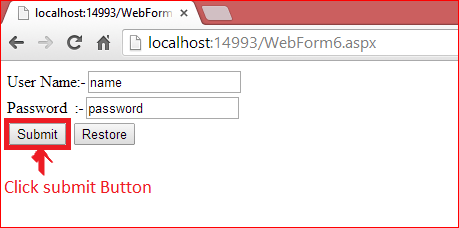
And now I am explaining the stored value in the View State and the remaining steps are the same as the previous.

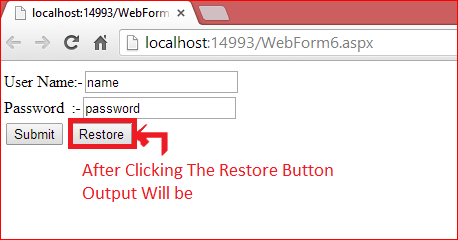
Now write this code,

1. **protected** **void** Button1\_Click(object sender, EventArgs e)
2. {
3. //Value of Textbox1 and TectBox2 is assigin on the ViewState
4. ViewState["name"] = TextBox1.Text;
5. ViewState["password"] = TextBox2.Text;
6. //after clicking on Button TextBox value Will be Cleared
7. TextBox1.Text = TextBox2.Text = string.Empty;
8. }
9. **protected** **void** Button3\_Click(object sender, EventArgs e)
10. {
11. //If ViewState Value is not Null then Value of View State is Assign to TextBox
12. **if** (ViewState["name"] != **null**)
13. {
14. TextBox1.Text = ViewState["name"].ToString();
15. }
16. **if** (ViewState["password"] != **null**)
17. {
18. TextBox2.Text = ViewState["password"].ToString();
19. }
20. }

**Output**

Now the output is,





After clicking on the Submit Button the value of user name and password is submitted in View State and the View State stores the value of user name and password during post-back.

After click on the Restore Button we can get the value again. The Value must be retained during post-back and the values are stored into a base 64 encoded string and this information is then put into the View State Hidden Field.

## Data Objects That Can be Stored in View state

1. String
2. Boolean Value
3. Array Object
4. Array List Object
5. Hash Table
6. Custom type Converters

## Advantages of View State

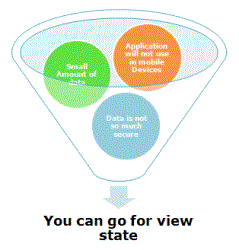
1. Easy to Implement.
2. No server resources are required: The View State is contained in a structure within the page load.
3. Enhanced security features: It can be encoded and compressed or Unicode implementation.

## Disadvantages of View State

1. Security Risk: The Information of View State can be seen in the page output source directly. You can manually encrypt and decrypt the contents of a Hidden Field, but It requires extra coding. If security is a concern then consider using a Server-Based state Mechanism so that no sensitive information is sent to the client.
2. Performance: Performance is not good if we use a large amount of data because View State is stored in the page itself and storing a large value can cause the page to be slow.
3. Device limitation: Mobile Devices might not have the memory capacity to store a large amount of View State data.
4. It can store values for the same page only.

## When We Should Use View State

1. When the data to be stored is small.
2. Try to avoid secure data.



## How to Enable and Disable View State

You can enable and disable View State for a single control as well as at the page level also. To turn off View State for a single control, set the EnableViewState property of that control to false.

1. TextBox1.EnableViewState=**false**;

To turn off the View State for an entire page, we need to set EnableViewState to false of the page directive as shown below:

1. <%Page Language="C#" EnableViewState="false";

For enabling the same, you need to use the same property just set it to "True".

## View State Security

View State Data is stored in the form of Base 64 encoding but it is not more secure, anyone can easily break it. So there are the following 2 options,

1. Using the MAC for Computing the View State Hash Value  
   Generally, the larger MAC key is used to generate a Hash Key. When the key is auto-generated then ASP.NET uses SHA-1 encoding to create a larger key. Those keys must be the same for all the server. If the key is not the same and the page is posted back to a different server than the one that created the page then the ASP.NET Page Framework raises an exception. We can enable it by using,
   1. <%Page Language="C#" EnableViewState="true"  EnableViewStateMac="true";
2. Encryption   
   By using MAC Encoding we cannot prevent the viewing of the data so to prevent the viewing, transmit the page over SSL and encrypt the View State Data. To encrypt the data we have the ViewStateEncryptionMode Property and it has the following 3 options:  
   1. Always: Encrypt the data Always.
   2. Never: Encrypt the data Never.
   3. Auto: Encrypt if  any Control request specially for Encryption

We can enable it by using,

1. <%Page Language="C#" EnableViewState="true ViewStateEncryptionMode="Always"

**Cookies  
  
Cookies are one of the client-side state management players, which store the data at the client side.**

Mostly, 4096 bytes per cookie can be stored. Storing the data client side means it stores them in the client's web browser or hard disk.  
  
Cookies start to work when a user requests a page from your site or enters the URL in the browser. The browser looks for cookies in the local hard disk, associated with the URL. If the cookie exists, the browser sends the cookie to your site along with the page request.  
  
You will get [more](http://browsercookielimits.squawky.net/) interesting facts about cookie limitations.  
  
Defining/Creating/Setting a cookie with Response.Cookies[“COOKIES\_NAME HERE”].Value  
  
**Example**

1. Response.Cookies["UserID"].Value = txtUserID.Text;

Read/Get a cookie value with Request.Cookies[“COOKIES\_NAME HERE”].Value.  
  
**Example**

1. lblUserLoginID.Text = Request.Cookies["UserID"].Value;

**Uses of cookies**We use cookies mostly for the following reasons.

1. Log-In Information.
2. To check the previously logged in or not.
3. Cart details in cookies before checkout.
4. Conducting a poll.

**Types of cookies**There are two types of cookies.

* Persistent Cookies
* Non-Persistent Cookies

**Persistent Cookies**Cookies are stored on your computer hard disk. The data stays in your hard disk and can be accessed by web servers unless the cookies are deleted or have expired.  
  
**Non-persistent Cookies**If you don’t set expiry property of cookies, those cookies are called Non-persistent Cookies. Cookies are saved in web browser and remain till the browser is running.  
  
**Advantages and disadvantage of cookies**  
  
**Advantage of Cookies**

1. Stored in client-side, require less server-side resources.
2. Easy to create/write/set.
3. Auto deleted as we close the browser or we can set the expiry date.
4. Restrict the cookies scope on folder level or domain level.

**Disadvantage of Cookies**

1. Cookies can be deleted by user at any time, so we can not completely rely on cookies.
2. Cookies store in user system (computer) as normal text file.
3. There is limitation of 4096 bytes (4 KB) for storing the data.

**Create/Set/Write the cookies**There are two ways to create cookies,

1. //First way to create cookies
2. //Assigning the TextBox called txtUserID value into cookies called UserID.
3. Response.Cookies["UserID"].Value = txtUserID.Text;
5. //Assigning the expiry of cookies called UserID.
6. Response.Cookies["UserID"].Expires = DateTime.Now.AddDays(1);


10. //Second way to create cookies
11. HttpCookie aCookie = **new** HttpCookie("LoginDateTime");
12. aCookie.Value = DateTime.Now.ToString();
14. //Added Expiry of cookies
15. aCookie.Expires = DateTime.Now.AddDays(1);
17. //Added cookies
18. Response.Cookies.Add(aCookie);

**Read the cookies**While reading the cookies, you should always check the null or not null, otherwise an exception can be thrown on the page.  
  
**Example**

1. **if** (Request.Cookies["UserID"] != **null**)
2. {
3. lblUserLoginID.Text = Request.Cookies["UserID"].Value;
4. }
6. **if** (Request.Cookies["LoginDateTime"] != **null**)
7. {
8. HttpCookie aCookie = Request.Cookies["userName"];
9. lblLoginDateTime.Text = aCookie.Value;
10. }

**Restrict the scope of cookies**Basically, restriction is required on cookies because all the cookies for a site are sent to the Server.  
  
We scope the cookies in the  following ways,

1. Path or Folder base.
2. Domain base.

Path or Folder based restricted cookies example:

1. HttpCookie PurchaseAppCookie = **new** HttpCookie("PurchaseAppCookie");
2. PurchaseAppCookie.Value = "Entered in Purchase " + DateTime.Now.ToString();
3. PurchaseAppCookie.Expires = DateTime.Now.AddDays(1);
4. PurchaseAppCookie.Path = "/Purchase";
5. Response.Cookies.Add(PurchaseAppCookie);

In the above code, cookies are available only to those pages which have the  path of “PURCHASE” folder.  
  
Domain based restricted cookies.

1. Response.Cookies["currentDomain"].Value = DateTime.Now.ToString();
2. Response.Cookies["currentDomain"].Expires = DateTime.Now.AddDays(1);
3. Response.Cookies["currentDomain"].Domain = "www.c-sharpcorner.com";

In the above code, cookies are available only to this domain called “www.c-sharpcorner.com”.   
  
**Sample code of using cookies**  
  
Create a new empty web site project named “CookiesExample”.  
  
Graphical user interface, application

Description automatically generated  
  
Right click on the project and select Add-->Add New Item and select WebForm.  
  
Graphical user interface, application

Description automatically generated  
  
Add a new Web Form named “Default.aspx”. In this page, we will be accepting user id and password as well as creating cookies.  
  
Graphical user interface, application

Description automatically generated  
  
**Default.aspx code**

1. <%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="\_Default" %>
3. <!DOCTYPE html>
5. <html xmlns="http://www.w3.org/1999/xhtml">
6. <head runat="server">
7. <title></title>
8. <style type="text/css">
9. .auto-style1 {
10. width: 140px;
11. }
12. </style>
13. </head>
14. <body>
15. <form id="form1" runat="server">
16. <div>
17. <h2>Sign-In </h2>
18. <table style="width: 100%;">
19. <tr>
20. <td **class**="auto-style1">User ID.</td>
21. <td>
22. <asp:TextBox ID="txtUserID" runat="server"></asp:TextBox>
23. </td>
24. <td> </td>
25. </tr>
26. <tr>
27. <td **class**="auto-style1">Password</td>
28. <td>
29. <asp:TextBox ID="txtPassword" runat="server" TextMode="Password"></asp:TextBox>
30. </td>
31. <td> </td>
32. </tr>
33. <tr>
34. <td colspan="2">
35. <asp:Button ID="btnSignIn" runat="server" OnClick="btnSignIn\_Click" Text="Sign-In" />
36. </td>
37. <td> </td>
38. </tr>
39. </table>


43. <p>Login page of Portal and User ID and Login DateTime is stored **in** cookies.</p>
44. </div>
45. </form>
46. </body>
47. </html>

**Default.aspx.cs code**

1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Web;
5. using System.Web.UI;
6. using System.Web.UI.WebControls;
8. **public** partial **class** \_Default : System.Web.UI.Page
9. {
10. **protected** **void** Page\_Load(object sender, EventArgs e)
11. {
12. }
13. **protected** **void** btnSignIn\_Click(object sender, EventArgs e)
14. {
15. //First way to create cookies
16. //Assigning the TextBox called txtUserID value into cookies called UserID.
17. Response.Cookies["UserID"].Value = txtUserID.Text;
19. //Assigning the expiry of cookies called UserID.
20. Response.Cookies["UserID"].Expires = DateTime.Now.AddDays(1);


24. //Second way to create cookies
25. HttpCookie aCookie = **new** HttpCookie("LoginDateTime");
26. aCookie.Value = DateTime.Now.ToString();
28. //Added Expiry of cookies
29. aCookie.Expires = DateTime.Now.AddDays(1);
31. //Added cookies
32. Response.Cookies.Add(aCookie);
34. Response.Redirect("ReadingCookies.aspx");
36. }
37. }

**Reading the Cookies**  
  
Right click on the project and select Add-->Add New Item and select WebForm.  
  
Graphical user interface, application

Description automatically generated  
  
Graphical user interface, application

Description automatically generated  
  
**ReadingCookies.aspx code**

1. <%@ Page Language="C#" AutoEventWireup="true" CodeFile="ReadingCookies.aspx.cs" Inherits="ReadingCookies" %>
3. <!DOCTYPE html>
5. <html xmlns="http://www.w3.org/1999/xhtml">
6. <head runat="server">
7. <title></title>
8. </head>
9. <body>
10. <form id="form1" runat="server">
11. <h2>Values are coming from cookies.</h2>
12. <br/>
13. <br/>
14. Your Login ID:
15. <asp:Label ID="lblUserLoginID" runat="server" Text="Label" Font-Bold="true"></asp:Label>
16. <br />
17. <br />
18. Your login date time:
19. <asp:Label ID="lblLoginDateTime" runat="server" Text="Label" Font-Bold="true"></asp:Label>
20. <div>
22. </div>
23. </form>
24. </body>
25. </html>

**ReadingCookies.aspx.cs code**

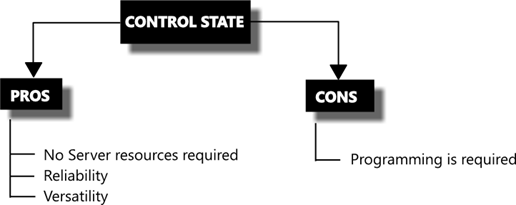
1. using System;
2. using System.Collections.Generic;
3. using System.Linq;
4. using System.Web;
5. using System.Web.UI;
6. using System.Web.UI.WebControls;
8. **public** partial **class** ReadingCookies : System.Web.UI.Page
9. {
10. **protected** **void** Page\_Load(object sender, EventArgs e)
11. {
12. **if** (Request.Cookies["UserID"] != **null**)
13. {
14. lblUserLoginID.Text = Request.Cookies["UserID"].Value;
15. }
17. **if** (Request.Cookies["LoginDateTime"] != **null**)
18. {
19. HttpCookie aCookie = Request.Cookies["LoginDateTime"];
20. lblLoginDateTime.Text = aCookie.Value;
21. }
22. }
23. }

Control State

The purpose of the control state repository is to cache data necessary for a control to properly function. ControlState is essentially a private ViewState for your control only, and it is not affected when ViewState is turned off. ControlState is used to store small amounts of critical information. Heavy usage of ControlState can impact the performance of application because it involves serialization and deserialization for its functioning.

There are two methods you have to implement in your custom control.

* Load Control State
* Save Control State



**Code Example**

1. **public** **class** ControlStateWebControl : Control
2. {
3. #region Members
4. **private** **string** \_strStateToSave;
5. #endregion
6. #region Methods
7. **protected** **override** **void** OnInit(EventArgs e)
8. {
9. Page.RegisterRequiresControlState(**this**);
10. **base**.OnInit(e);
11. }
12. **protected** **override** **object** SaveControlState()
13. {
14. **return** \_strStateToSave;
15. }
16. **protected** **override** **void** LoadControlState(**object** state)
17. {
18. **if** (state != **null**)
19. {
20. \_strStateToSave = state.ToString();
21. }
22. }
23. #endregion
24. }

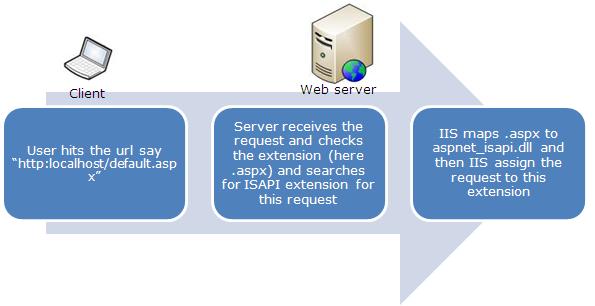
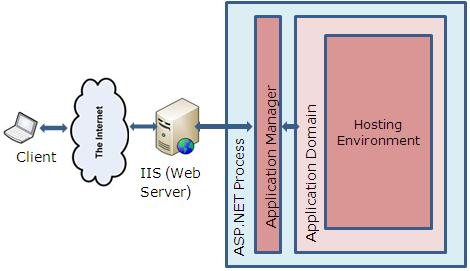
# **Application State in ASP.Net**

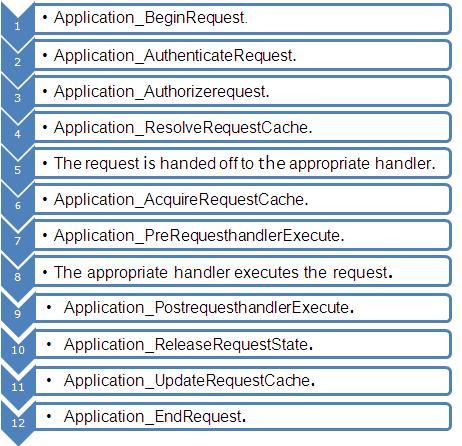
**Background**  
  
We all know that the web uses the HTTP protocol and that the HTTP protocol is a stateless protocol, in other words when a client sends a request to the server, an instance of the page is created and the page is converted to HTML format and then the server returns the response and then the instance of the page and the value of the control is destroyed. So if we have a requirement to store the value of controls then a State Management technique is used.

## Introduction

Application State is a state management technique. Application State is stored in the memory of the the server and is faster than storing and retrieving information in a database. Session sate is specific for a single user session, but Application State is for all users and sessions. Application State does not have a default expiration period. When we close the worker process the application object will be lost. Technically the data is shared amongst users by a HTTPApplcationState class and the data can be stored here in a key/value pair. It can also be accessed using the application property of the HTTPContext class.

## Application State Life Cycle

**Step 1:**When the Browser sends a request to the web server and the server receives the request it first checks the extension to determine whether or not it is ISAPI because this request can only be handled by the ISAPI extension; if the extension is different then the request is handled by the server itself.  
  
  
  
**Step 2:** After receiving the request the Application Manager creates an application domain. In the application domain an instance of the class HostingEnvironment is created that provides access to information about all application resources.  
  
  
  
**Step 3:** After creating the application domain, ASP.NET initializes the basic objects as HTTPContext, HTTPRequest and HTTPResponse. HTTPContext holds objects to the specific application request as HTTPRequest and HTTPResponse.HTTPRequest contains all the information regarding the current request like cookies, browser information and so on and the HTTPResponse contains the response that is sent to the client.  
  
**Step 4:**Here all the basic objects are being initialized and the application is being started with the creation of the HTTPApplication class.  
  
Diagram

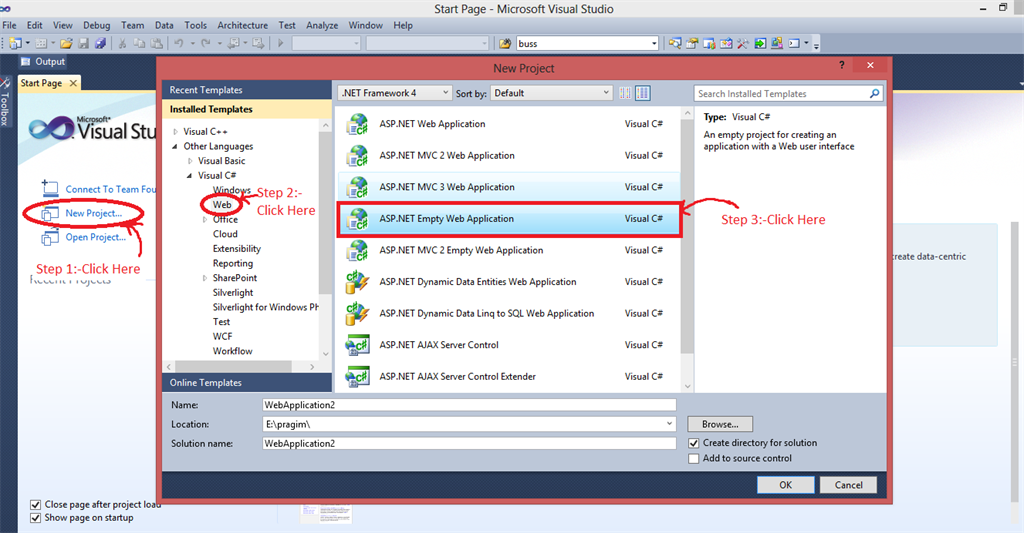
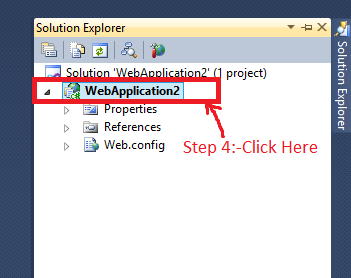
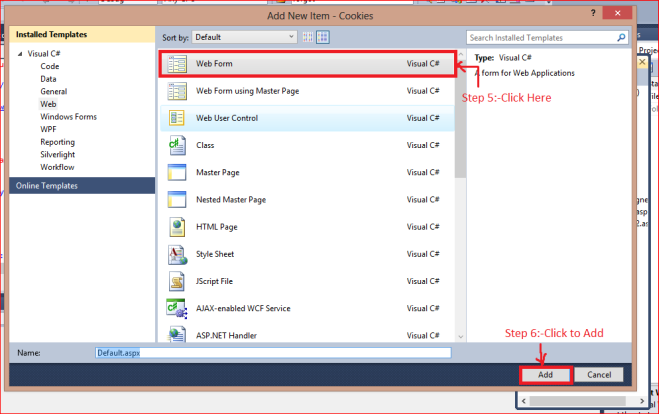
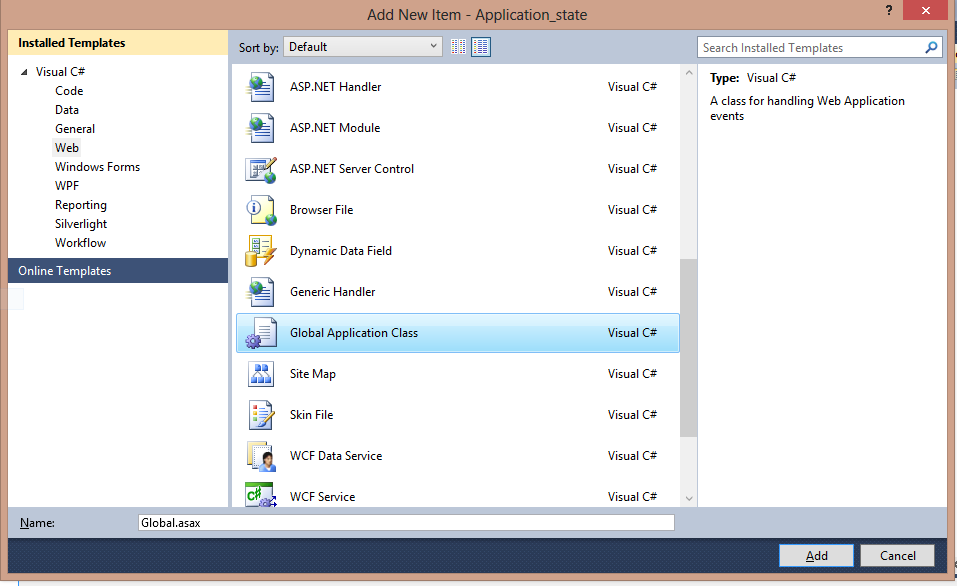
Description automatically generated  
  
**Step 5:** Then events are executed by the HTTPApplication class for any specific requirement. Here is a list of events:  
  
  
  
**Global.asax file:**the Global.asax file is used for handling application events or methods. It always exists in the root level. Events are one of the following of the 2 types in the Global application:

1. Events that will be raised on a certain condition.
2. Events that will be raised on every request.

The application will be started only once; if 10 users send a request then 10 user sessions are created. The events of the Global.asax file are:

1. **Application\_Start() :**This method is invoked initially when first application domain is created.
2. **Session\_Start() :**This method is called every time a session is start.
3. **Application\_BeginRequest() :** After an application has started the first method Application\_BeginRequest() is executed for every user.
4. **Application\_AuthenticateRequest() :** It checks to determine whether or not the user is valid.
5. **Application\_Error() :**Whenever an unhandled exception occurs then this event will be called.
6. **Session\_End() :** When a user session is ended and all the data related to a specific user is cleared then the Session\_End() event is called.
7. **Application\_End() :** This method is called before the application ends. This can take place if IIS is restarted or the application domain is changing.
8. **Application\_Disposed() :**This event is called after the application will be shut down and the .NET GC is about to reclaim the memory it occupies. Although this is very late to perform any clean-up but we can use it for safety purposes.

## ASP.NET Application State real-life example

Now I am explaining the real-life example. If you want to see the number of users online then we need to use Application State.  
  
**Step 1:**Open Visual Studio 2010.  
  
**Step 2:** Then click on "New Project" > "Web" > "ASP.NET Empty Web Application" .  
  
  
  
**Step 3:**Now click on Solution Explorer.  
  
  
  
**Step 4:**Now right-click on "Add" > "New Item" > "Web Form" and add the name of the web form.  
  
  
  
**Step 5:** Now add the Global.asax file. Again go to Solution Explorer and "Add" > "New Item" > "Global Application Class".  
  
  
  
**Step 6:** Now to configure the session we need to use the web.config file as in the following:

1. <sessionState mode="InProc" timeout="20" cookieless="true"></sessionState>

**Step 7:**Now to count the number of users online we need to use the global.asax file as in the following:

1. **protected** **void** Application\_Start(**object** sender, EventArgs e)
2. {
3. //this event is execute only once when application start and it stores the server memory until the worker process is restart
4. Application["user"] = 0;
5. }
6. **protected** **void** Session\_Start(**object** sender, EventArgs e)
7. {

   //when session in start application variable is increased by 1

   Application.Lock();

   Application["user"] = (**int**) Application["user"]+1;

   Application.UnLock();

1. }
2. **protected** **void** Session\_End(**object** sender, EventArgs e)
3. {

   //when session in end application variable is decrease by 1

   Application.Lock();

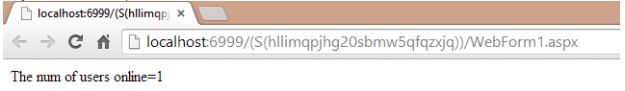
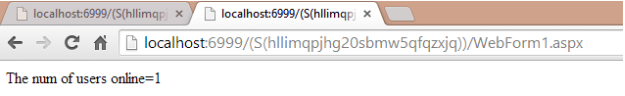
   Application["user"] = (**int**)Application["user"] - 1;

   Application.UnLock();

1. }

**Step 8:** Now to show the online users we need to use a web form as in the following:

1. **protected** **void** Page\_Load(**object** sender, EventArgs e)
2. {
3. Response.Write("The num of users online=" + Application["user"].ToString());
4. }

**Output**  
  
  
  
  
When the same request is sent to the server with a different browser then the number of online clients is also not increased because the browser binds with the session id so both of the tabs have the same session id so the server knows that the request comes from the same user. If we change the session id from the URL and again refresh then the number of online clients is increased by one because the server thinks that the request comes from a different browser.

## Important points of Application State variables

1. Application State variables are available across all pages and all sessions. Application State variables are like multi-user Global data.
2. Application variables are stored on a web server.
3. Application State variables are cleared, only when the process hosting the application is restarted, that is when the application is ended.
4. Application State variables do not support web farms and web gardens: Application State variables are not supported be web farms.  
     
   Diagram

   Description automatically generated  
     
   A client sends a request and the request goes to the load balancer and the load balancer sends a request to web server1 and the Application State variables are stored in a web server1. If the subsequent request is sent by the client again and the load balancer sends a request to web server2 and the Application State variables are not stored in web server2 then something. Web servers do not share application state variables.
5. Application State variables have a concurrency problem so we need to synchronize the method by using the lock and unlock methods. So multiple thread problems are resolved since only one thread can do the work.
6. An application variable is used only when the variable needs to have global access and when you need them for the entire time, during the lifetime of an application.

Cookies is a small piece of information stored on the client machine. This file is located on client machines "C:\Document and Settings\Currently\_Login user\Cookie" path.  Its is used to store user preference information like Username, Password,City and PhoneNo etc on client machines. We need to import namespace called  Systen.Web.HttpCookie before we use cookie.  
 

Type of Cookies

1. Persist Cookie - A cookie has not have expired time Which is called as Persist Cookie
2. Non-Persist Cookie - A cookie has expired time Which is called as Non-Persist Cookie

How to create a cookie?

It is really easy to create a cookie in the Asp.Net with help of Response object or HttpCookie

Example 1

HttpCookie userInfo = **new** HttpCookie("userInfo");

userInfo["UserName"] = "Annathurai";

userInfo["UserColor"] = "Black";

userInfo.Expires.Add(**new** TimeSpan(0, 1, 0));

Response.Cookies.Add(userInfo);

Example 2

1. Response.Cookies["userName"].Value = "Annathurai";
2. Response.Cookies["userColor"].Value = "Black";

How to retrieve from cookie?

It is easy way to retrieve cookie value form cookies with the help of Request object.

Example 1

**string** User\_Name = **string**.Empty;

**string** User\_Color = **string**.Empty;

User\_Name = Request.Cookies["userName"].Value;

User\_Color = Request.Cookies["userColor"].Value;

Example 2

1. **string** User\_name = **string**.Empty;
2. **string** User\_color = **string**.Empty;
3. HttpCookie reqCookies = Request.Cookies["userInfo"];
4. **if** (reqCookies != **null**)
5. {
6. User\_name = reqCookies["UserName"].ToString();
7. User\_color = reqCookies["UserColor"].ToString();
8. }

When we make a request from the client to web server, the web server process the request and give a lot of information with big pockets which will have Header information, Metadata, cookies etc., Then repose object can do all the things with browser.

Cookie's common property

1. Domain => Which is used to associate cookies to domain.
2. Secure  => We can enable secure cookie to set true(HTTPs).
3. Value    => We can manipulate individual cookie.
4. Values  => We can manipulate cookies with key/value pair.
5. Expires => Which is used to set expire date for the cookies.

Advantages of Cookie

1. Its clear text so user can able to read it.
2. We can store user preference information on the client machine.
3. Its easy way to maintain.
4. Fast accessing.

Disadvantages of Cookie

1. If user clear cookie information we can't get it back.
2. No security.
3. Each request will have cookie information with page.

How to clear the cookie information?

1. We can clear cookie information from client machine on cookie folder
2. To set expires to cookie object
   1. userInfo.Expires = DateTime.Now.AddHours(1);

It will clear the cookie with one hour duration.

# **Application State in ASP.Net**

* [A picture containing colorful

  Description automatically generated](https://www.c-sharpcorner.com/members/divya-sharma13)

* [Divya Sharma](https://www.c-sharpcorner.com/members/divya-sharma13)

* Updated date Apr 08, 2021

* 252.8k

* [3](javascript:void(0);)

* [8](javascript:void(0);)
* [facebook](https://www.facebook.com/sharer.php?u=https://www.c-sharpcorner.com/UploadFile/225740/application-state-in-Asp-Net/Default.aspx)
* [twitter](https://twitter.com/intent/tweet?&via=CsharpCorner&related=CsharpCorner&text=Application+State+in+ASP.Net%20https://www.c-sharpcorner.com/UploadFile/225740/application-state-in-Asp-Net/Default.aspx)
* [linkedIn](https://www.linkedin.com/shareArticle?mini=true&url=https://www.c-sharpcorner.com/UploadFile/225740/application-state-in-Asp-Net/Default.aspx)
* [Reddit](https://www.reddit.com/submit?title=Application+State+in+ASP.Net&url=https%3a%2f%2fwww.c-sharpcorner.com%2fUploadFile%2f225740%2fapplication-state-in-Asp-Net%2fdefault.aspx)
* [Expand](javascript:void(0);)

[Download Free .NET & JAVA Office Files API](https://www.c-sharpcorner.com/AdRedirector.aspx?AdId=133&target=https://www.e-iceblue.com/download/free-products.html?aff_id=100)

[Try Free File Format APIs for Word/Excel/PDF](https://www.c-sharpcorner.com/AdRedirector.aspx?AdId=133&target=https://products.aspose.com/total/net?utm_source=csharpcorner&utm_medium=text-link&utm_campaign=csharpcorner-2020)

[**Application\_state.rar**](javascript:void(0);)

**Background**  
  
We all know that the web uses the HTTP protocol and that the HTTP protocol is a stateless protocol, in other words when a client sends a request to the server, an instance of the page is created and the page is converted to HTML format and then the server returns the response and then the instance of the page and the value of the control is destroyed. So if we have a requirement to store the value of controls then a State Management technique is used.

## Introduction

Application State is a state management technique. Application State is stored in the memory of the the server and is faster than storing and retrieving information in a database. Session sate is specific for a single user session, but Application State is for all users and sessions. Application State does not have a default expiration period. When we close the worker process the application object will be lost. Technically the data is shared amongst users by a HTTPApplcationState class and the data can be stored here in a key/value pair. It can also be accessed using the application property of the HTTPContext class.

## Application State Life Cycle

**Step 1:**When the Browser sends a request to the web server and the server receives the request it first checks the extension to determine whether or not it is ISAPI because this request can only be handled by the ISAPI extension; if the extension is different then the request is handled by the server itself.

ISAPI extensions are implemented as **DLLs** that are loaded into a process that is controlled by IIS. Like ASP and HTML pages, IIS uses the virtual location of the DLL file in the file system to map the ISAPI extension into the URL namespace that is served by IIS.  
  
Text

Description automatically generated  
  
**Step 2:** After receiving the request the Application Manager creates an application domain. In the application domain an instance of the class HostingEnvironment is created that provides access to information about all application resources.  
  
Diagram, schematic

Description automatically generated  
  
**Step 3:** After creating the application domain, ASP.NET initializes the basic objects as HTTPContext, HTTPRequest and HTTPResponse. HTTPContext holds objects to the specific application request as HTTPRequest and HTTPResponse.HTTPRequest contains all the information regarding the current request like cookies, browser information and so on and the HTTPResponse contains the response that is sent to the client.  
  
**Step 4:**Here all the basic objects are being initialized and the application is being started with the creation of the HTTPApplication class.  
  
Diagram

Description automatically generated  
  
**Step 5:** Then events are executed by the HTTPApplication class for any specific requirement. Here is a list of events:  
  
Table

Description automatically generated  
  
**Global.asax file:**the Global.asax file is used for handling application events or methods. It always exists in the root level. Events are one of the following of the 2 types in the Global application:

1. Events that will be raised on a certain condition.
2. Events that will be raised on every request.

The application will be started only once; if 10 users send a request then 10 user sessions are created. The events of the Global.asax file are:

1. **Application\_Start() :**This method is invoked initially when first application domain is created.
2. **Session\_Start() :**This method is called every time a session is start.
3. **Application\_BeginRequest() :** After an application has started the first method Application\_BeginRequest() is executed for every user.
4. **Application\_AuthenticateRequest() :** It checks to determine whether or not the user is valid.
5. **Application\_Error() :**Whenever an unhandled exception occurs then this event will be called.
6. **Session\_End() :** When a user session is ended and all the data related to a specific user is cleared then the Session\_End() event is called.
7. **Application\_End() :** This method is called before the application ends. This can take place if IIS is restarted or the application domain is changing.
8. **Application\_Disposed() :**This event is called after the application will be shut down and the .NET GC is about to reclaim the memory it occupies. Although this is very late to perform any clean-up but we can use it for safety purposes.

## ASP.NET Application State real-life example

Now I am explaining the real-life example. If you want to see the number of users online then we need to use Application State.  
  
**Step 1:**Open Visual Studio 2010.  
  
**Step 2:** Then click on "New Project" > "Web" > "ASP.NET Empty Web Application" .  
  
Graphical user interface, text, application, email

Description automatically generated  
  
**Step 3:**Now click on Solution Explorer.  
  
Graphical user interface, text, application

Description automatically generated  
  
**Step 4:**Now right-click on "Add" > "New Item" > "Web Form" and add the name of the web form.  
  
Graphical user interface, application

Description automatically generated  
  
**Step 5:** Now add the Global.asax file. Again go to Solution Explorer and "Add" > "New Item" > "Global Application Class".  
  
Graphical user interface, application

Description automatically generated  
  
**Step 6:** Now to configure the session we need to use the web.config file as in the following:

1. <sessionState mode="InProc" timeout="20" cookieless="true"></sessionState>

**Step 7:**Now to count the number of users online we need to use the global.asax file as in the following:

1. **protected** **void** Application\_Start(**object** sender, EventArgs e)
2. {
3. //this event is execute only once when application start and it stores the server memory until the worker process is restart
4. Application["user"] = 0;
5. }
6. **protected** **void** Session\_Start(**object** sender, EventArgs e)
7. {
8. //when session in start application variable is increased by 1
9. Application.Lock();
10. Application["user"] = (**int**) Application["user"]+1;
11. Application.UnLock();
12. }
13. **protected** **void** Session\_End(**object** sender, EventArgs e)
14. {
15. //when session in end application variable is decrease by 1

   Application.Lock();

   Application["user"] = (**int**)Application["user"] - 1;

   Application.UnLock();

1. }

**Step 8:** Now to show the online users we need to use a web form as in the following:

1. **protected** **void** Page\_Load(**object** sender, EventArgs e)
2. {
3. Response.Write("The num of users online=" + Application["user"].ToString());
4. }

**Output**  
  
  
  
  
When the same request is sent to the server with a different browser then the number of online clients is also not increased because the browser binds with the session id so both of the tabs have the same session id so the server knows that the request comes from the same user. If we change the session id from the URL and again refresh then the number of online clients is increased by one because the server thinks that the request comes from a different browser.

## Important points of Application State variables

1. Application State variables are available across all pages and all sessions. Application State variables are like multi-user Global data.
2. Application variables are stored on a web server.
3. Application State variables are cleared, only when the process hosting the application is restarted, that is when the application is ended.
4. Application State variables do not support web farms and web gardens: Application State variables are not supported be web farms.  
     
   Diagram

   Description automatically generated  
     
   A client sends a request and the request goes to the load balancer and the load balancer sends a request to web server1 and the Application State variables are stored in a web server1. If the subsequent request is sent by the client again and the load balancer sends a request to web server2 and the Application State variables are not stored in web server2 then something. Web servers do not share application state variables.
5. Application State variables have a concurrency problem so we need to synchronize the method by using the lock and unlock methods. So multiple thread problems are resolved since only one thread can do the work.
6. An application variable is used only when the variable needs to have global access and when you need them for the entire time, during the lifetime of an application.