**State Management**

Therefore it cannot automatically indicate whether the sequential requests are coming from the same or different clients. For example, if you need to develop a Web application that provides a personalized page to users after subsequent logon, you need to keep track of user activities when users access the web page. You need to implement session tracking for each user accessing the Web application because HTTP does not enable you to keep track of user activities in a Web Page. In addition, a Web page is fetched again, each time the page is posted to a Web server. This results in loss of information associated with the page and the controls on the page with each round trip. For example, if user selects a checkbox in a Web page, the selection will be lost in the round trip of the page.

To overcome this limitation, ASP.NET provides different ***state management*** options.

State management options can be divided into two categories:

1. ***Client-Side management***
2. ***Server-Side management***

The following table lists the Client-side and server-side options:

|  |  |
| --- | --- |
| **CLIENT-SIDE OPTIONS** | **SERVER-SIDE OPTIONS** |
| View State | Application state |
| Hidden Fields | Session state |
| Cookies | Database support |
| Query Strings |  |

## Client-Side State Management Options

Client side state management options involve storing information either in a Web page or on a Client computer. Client-side management gives a better performance, as the load on the server is less. Disadvantage of this option is that only limited data can be stored. The client-side options are as follows:

## View State

The ViewState property of an ASP.NET web page enables you to retain page and control-specific values between round trips. Each web page and the controls on the page have the ViewState property that is inherited from the base Control class. ASP.NET framework uses the ViewState property to automatically save the values of the Web page and each control on the Web page prior to rendering the page. During postback of a Web page to itself, one of the tasks performed by ASP.NET is to restore the values in **\_VIEWSTATE.**

The view state is implemented with a hidden form field called **\_VIEWSTATE**, which is automatically created in every Web page. The encoded string is then assigned to the *Value* attribute of the hidden form field \_VIEWSTATE and is sent to the client as a part of the Web page. When page is initialized during postback, ASP.NET Framework parses the ViewState string to restore the property information in the page.

The view state of a Web page or a control consists of the cumulative property values of the page or the control. To preserve these values across stateless HTTP requests, Web pages and controls in ASP.NET use an instance of the **StateBag** class. This class is the primary storage mechanism for HTML and server controls. It stores attribute/value pairs associated with HTML and server controls as strings. Therefore, this class is used to manage the view state of an ASP.NET Web page and the controls added to the page after the OnInit method is executed. When you add values to this class, the values are automatically added to the hidden \_VIEWSTATE form variable. The syntax to add a value to the StateBag class is as follows:

ViewState("any string") = "some value"

By default, the Viewstate property of both Web page and the controls on the Web page is enabled.

Perform the following steps to determine the effect of enabling and disabling the ViewState property in an ASP.NET web application:

* Create a new Web application.
* Add a label and a button control to Webform1.aspx. The label control and the button control should have the default name, Label1 and Button1, repectively. Ensure that the EnableViewState property of the Label and Button control are set to true. In addition, ensure that the EnableViewState property to the Page is set to true.
* Add the following lines of code in the load event of the page and execute the application.

If Not IsPostBack Then

Label1.Text = "Hello"

End If

When the page is first loaded in the memory of the client computer, the message Hello is assigned to the Text property of the label control. However, if you click the button and reload the page, the label continues to display the message. It is because the Text property of the Label control, Label1 is automatically preserved through the view state property of the control..

Now if you set the ViewState property of the label control to false and click the button, the displayed text would not be "hello" but "Label", the default text.

**Advantages:**

* Simple for page level data
* Encrypted
* Can be set at the control level

**Disadvantages:**

* Overhead in encoding View State values
* Makes a page heavy

## Hidden Fields

Hidden fields are used to store data at the page level. As its name says, these fields are not rendered by the browser. It's just like a standard control for which you can set its properties. Whenever a page is submitted to server, hidden fields values are also posted to server along with other controls on the page.

//to assign a value to Hidden field

Hidden1.Value="Create hidden fields";

//to retrieve a value

string str=Hidden1.Value;

**Advantages:**

* Simple to implement for a page specific data
* Can store small amount of data so they take less size.

**Disadvantages:**

* Inappropriate for sensitive data
* Hidden field values can be intercepted(clearly visible) when passed over a network
* Small amount of data store

## Cookies

A cookie is used to store small piece of information on client machine. A cookie contains page-specific information that a Web server sends to a client along with page output. Cookies are used for sending page-specific information because HTTP is a stateless protocol and cannot indicate whether page requests are coming from the same or different clients. You can use cookies to keep track of each individual user who access a Web page across an HTTP connection.

Cookies are saved on the Client computer. Cookies can either be ***temporary or persistent.*** *Temporary cookies*, also known as session cookies, exist in the memory space of a browser. When the browser is closed, all session cookies added to the browser are lost. *A* ***persistent cookie*** is saved as a text file in the file system of the client computer.

A Web browser can access a cookie from the ***HttpCookieCollection*** by using the **Request** object. If a cookie is accessed using the Request built-in object, the cookie is a read only file. The cookie is stored on the Web browser and not on the Web server. However, if you want to modify a cookie, you need to use the **Response** built in object.

Perform the following steps to create a persistent cookie on a client computer and read the contents of the cookie:

* Create a new web application and add a button control to the form.
* Add the following lines of code in the click event of the button control.

Dim mycookie As New HttpCookie("Cookiename", "hello")

mycookie.Expires = System.Convert.ToDateTime("25/07/2018")

Response.Cookies.Add(mycookie)

Dim myvar As HttpCookie

myvar = Request.Cookies.Get("Cookiename")

Response.Write("Cookie :" + myvar.Value & "<br>")

Response.Write("Expires : " + myvar.Expires & "<br>")

**Advantages:**

* Simplicity

**Disadvantages:**

* Cookies can be disabled on user browsers
* Cookies are transmitted for each HTTP request/response causing overhead on bandwidth
* Inappropriate for sensitive data

## Query String

If you need to submit information back to a Web page or another page by using a URL, you can use a query string. The query string is part of the request that appears after the Question mark (?) character in the URL. A query string provides a simple way to pass information from one page to another. For example, in a logon screen, the user name can be passed to the next page in the application by using query strings.

**To set value**

Respose.Redirect(“default.aspx?name=abc”)

**To get value**

Dim str as String

Str=Request.QueryString(“name”)

You can also add more then one value inside parameter by

**To set value**

Respose.Redirect(“default.aspx?name=abc&city=vapi”)

**To get value**

Dim strname,strcity as String

strname=Request.QueryString(“name”)

strcity= Request.QueryString(“city”)

**Advantages:**

* Simple to Implement

**Disadvantages:**

* Human Readable
* Client browser limit on URL length
* Cross paging functionality makes it redundant
* Easily modified by end user

## Server-Side State Management Options

There is a limit to client-side options, there are server-side state management options. By using these options, you can manage application and session-related information. Server-side options store information on the Web server. These options are as follows:

### Application State

ASP.NET provides Application state as a means of storing application wide-specific information such as objects and variables. The information in the application state is stored in a key-value pair and is used to maintain data consistency between server round trips and between pages.

Application state is created when each browser request is made for a specific URL. After an application state is created, the application-specific information is stored in it. All information stored in the application state is shared among all the pages of the Web application by using the HttpApplicationState class. Syntax **to create** a variable and store it in the application state:

Application("myvar") = "Hello"

After the application in which you declared myvar is executed, any page contained in the application can retrieve the value of the myvar. To read the value of the myvar, you need to use the following statement:

Response.Write(Application("myvar"))

You can also add complex objects, such as Collection and Dataset, in application state. For example, you can add a dataset to an application state by using the following statements:

Dim ds as new Dataset()

Application("DataSet") = ds;

To remove the application variable myvar from the application state, you need to use the following statement:

Application.Remove(“myvar”)

or to remove all variable for application state wrote the following line of code:

Application.RemoveAll()

*It is important to note that after an object is added to an application state, it remains in the application state until the application is shut down, the* ***Global.asax*** *file is modified, or the item is explicitly removed from the application state.*

Since these variables are global to an application, it is important to consider the following issues while storing any value in an application state variable:

* The memory occupied by variables stored in an application state is not released until the value is either removed or replaced. Therefore, the number of variables and objects in an application state should be minimum. Otherwise, it will be an overhead on the Web server and the server response will be slow.
* Multiple pages within an application can access values stored in an application state simultaneously. Therefore, explicit synchronization methods need to be used to avoid deadlocks and access violations.

The ASP.NET application supports events. Two important events associated with ASP.NET application are discussed in the following table:

|  |  |
| --- | --- |
| **EVENT** | **DESCRIPTION** |
| **Application\_Start** | This event is triggered when an application starts. If you want a code to be executed as the application starts, you should add it to the Application\_Start event. This event is triggered only when the application starts and is not triggered again until the IIS is stopped, the Global.asax file is modified, or the application is unloaded. |
| **Application\_End** | This event is triggered when an application ends |
|  |  |

***Application State and Synchronization***

Multiple pages within an ASP.NET Web application can simultaneously access values, stored in an application state. This can result in conflicts and deadlocks. For example you can add a variable named ***PageCounter*** in the  application state to keep track of the number of times a page has been requested. If two users access a Web page simultaneously, there will be an attempt to update the value of the variable PageCounter simultaneously. This will lead to a problem. To avoid such situations, the HttpApplicationState class provides two methods, LOCK ( ) and UNLOCK ( ). These methods only allow one thread at a time to access applications state variables and objects.

*Note: Each browser request for a Web page initiates a new thread on the Web server.*

Calling the Lock ( ) method on an Application object causes ASP.NET to block attempts by the code running on other worker threads to access anything in an application state. These threads are unblocked only when the thread that called the Lock ( ) method calls the corresponding Unlock ( ) method on the Application object.

The following example illustrates the use of the Lock ( ) method and the Unlock ( ) method:

Create a new ASP.NET Web application.

Add the following lines of code in the Page\_Load event of the page:

Application.Lock()

If (Application("PageCounter") Is Nothing) Then

Application("PageCounter") = 0

else

Application("PageCounter") = Convert.ToInt32(Application("PageCounter")) + 1

Response.Write(Application("PageCounter"))

End If

Application.UnLock()

In the preceding example, the Lock() method is first called to ensure that the variable PageCounter cannot be simultaneously modified by another thread. Next, the counter is increased by 1 and then the value is displayed in the browser window. At last, the UnLock() methods is called to release the imposed lock on the application state variable PageCounter. You can notice the effect of Lock() and UnLock() methods by clicking the Refresh button on the browser window. Each time you click the Refresh button, the value of the PageCounter is incremented.  You can open the same form in another window by copying and pasting the URL, you will note that the value displayed will be not 1 but 1 incremented to the value you left on the previous window.

*Note: If you do not explicitly call the Unlock( ) method, the .NET Framework automatically removes the lock when the request completes or times out, or when an unhandled error occurs during request execution and causes the request to fail. This automatic unlocking prevents the application from deadlocking.*

You cannot selectively lock items in an application state, the application state object as a whole is blocked.

### Session State

In ASP.NET, session state is used to store session-specific information for a Website. Unlike application state, the scope of session state is limited to the current browser session. If different users are accessing a Web application, each will have a different session state. In addition, if a user exits and returns later, the user will have a different session state.

The session state has a built in support in ASP.NET. The built-in session state feature automatically performs the following actions:

* Identify and classify requests coming from a browser into a logical application session on the server.
* Store session-specific data on the server for use across multiple browser requests.
* Raise session lifetime-related events, such as ***Session\_OnStart*** and Session\_OnEnd, which can be handled using application code.

A unique 120-bit SessionID string containing ASCII characters identifies and tracks each active ASP.NET session. The following code can be used to add the variable name myvar in the session state:

Session("myvar") = "HELLO"

To display the value of my var, you can use the following statement:

Response.Write(Session("myvar"))

There are three issues that you need to consider while adding variables and objects to a session state:

* Any variable or object that you add to a session state is available only until the user closes the browser window. The variables and the objects are explicitly removed form session state if the user does not request a page for more than 20 minutes.
* Any variable or object added to a session state is related to a particular user. For example, you can store different values for myvar for two different users accessing the Web page and each user can access only the value that is assigned to him.
* Any object that supports serialization can be added to a session state. Since objects stored in a session state are stored on the server, session objects are not subject to the same size limitations as cookies.

Similar to the application state you can remove a variable added to the session state by using the Remove ( ) or RemoveAll ( ) method.

Session.Remove(“myvar”)

And Remove All

Session.RemoveAll()

***Starting and Ending a User Session***

A user session starts when a user requests the first page from a Web Site. when the first page is requested,  the Web server adds the ASP.NET SessionID cookie to the client computer. The following line enables you to view the value of session ID:

Response.Write(Session.SessionID)

You can use the Abandon() method of the Session object to explicitly stop a user session, by using the following line of code:

Session.Abandon()

You can also modify the Timeout property of the Session object to change the default value of timeout i.e. 20 minutes.

To do this open the **Web.Config** file and scroll to the <sessionState> section and modify the value of the Timeout property

***Handling Session events***

Session states has the following events that you can capture to manipulate an ASP.NET web application.:

* ***Session\_Start***
* ***Session\_End***

You can capture both these events by adding subroutines in the **Global.asax** file. A practical demonstration in the end of this article would make the usage of these events more clear.

You can also use the Session\_Start subroutine to automatically redirect users to a particular page when a user session begins.

***Advantages***

* 1. It is Secure.
  2. It can store complex data.

**DisAdvantages**

1. If simultensiolly millions of users are connected it can slowdown server