
Chapter -5 ASP.Net State Management

5.1 State Management

- In this chapter we will learn about various options for state management for web applications developed using ASP.NET. Generally, web applications are based on stateless HTTP protocol which does not remember any information about user requests. In typical client and server communication using HTTP protocol, page is created each time the page is requested.
- Developers have to implement various state management techniques when developing applications which provide customized content and which "remembers" the user.

What is State Management?

- State management is the process by which you maintain state and page information over multiple requests for the same or different pages.
- State: The current value of all the controls and variables for the current user in the current session is called the State.
- We can classify state management techniques in following ways:

There are two different types of state management:

Client-Based State Management

Server-Based State Management

➤ Client Side State Management

Client based state management techniques stores data on the client in various ways. Client based state management techniques are:

View state

Control state

Hidden fields

Cookies

Query strings

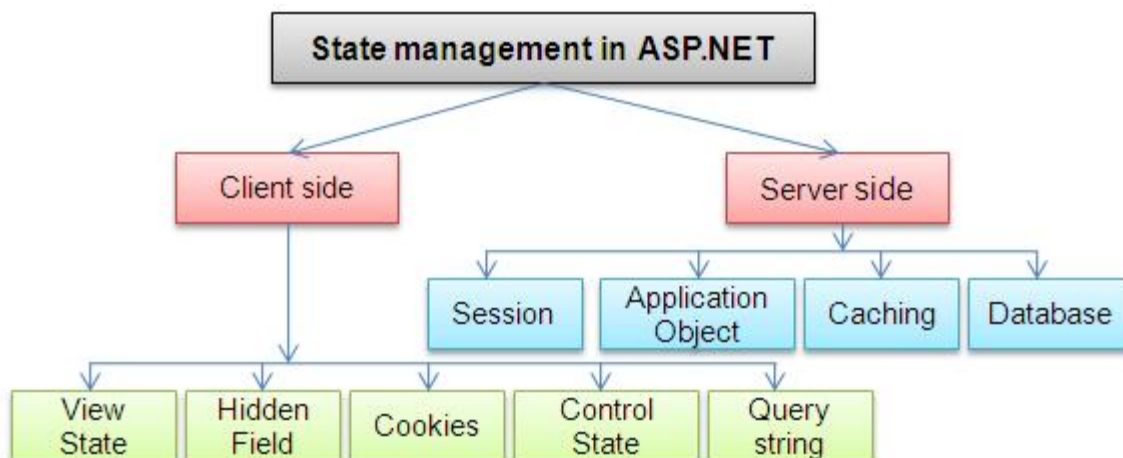
➤ **Server Side State Management**

Server based state management techniques store data in memory on the server. Server based state management techniques are:

Application state

Session state

Profile Properties



5.1.1 View State

- ViewState is used to maintain the state of controls during page postback
- The ViewState indicates the status of the page when submitted to the server.
- View State can be used to store state information for a single user.
- View State is a built in feature in web controls to persist data between page post backs.
- You can set ViewState on/off for each control using **EnableViewState** property.
- By default, **EnableViewState** property will be set to true.
- View state mechanism poses performance overhead.

- View state information of all the controls on the page will be submitted to server on each post back.
- To reduce performance penalty, disable View State for all the controls for which you don't need state. (Data grid usually doesn't need to maintain state).
- You can also disable View State for the entire page by adding **EnableViewState=false** to @page directive.
- Care must be taken to ensure view state for a page is smaller in size.

Storing & Retrieve value Objects in View State

We can store the value in viewstate using following syntax:

Syntax : `ViewState("myviewstate") = myValue`

Example: `ViewState("FavoriteColor") = TextBox1.Text`

We can retrieve the value from viewstate using following syntax:

Syntax : `myvalue = ViewState("myviewstate")`

Example:

```
Dim color As String  
Color=ViewState("FavoriteColor")
```

Advantages of using a View State

- Easy to implement
- No server resources are required
- Enhanced security features. The values in view state are hashed, compressed, and encoded for Unicode implementations.
- Simple for page level data
- Encrypted
- Can be set at the control level

Disadvantages of using a View State

- Scope is limited to only single page.
- Performance. The view state is stored in the page itself, so increase the page size.
- Security. The view state is stored in a hidden field on the page. Although view state stores data in a hashed format, it can be tampered with.
- Overhead in encoding View State values
- Makes a page heavy

Example:**Design Code :**

```
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        asp.net ViewState example: data read
        <asp:Label ID="Label1" runat="server" </asp:Label>
        <br />
        <asp:Label ID="Label2" runat="server" Text="Color
        Name"></asp:Label>
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
        <br />
        <asp:Button ID="Button1" runat="server" Text="Save Data In View State"
        OnClick="Button1_Click"/>
        &nbsp;
        <asp:Button ID="Button2" runat="server" Text="Read Data In View
        State" OnClick="Button2_Click"/>
    </form>
</body>
</html>
```

VB Code :

```
<%@ Page Language="VB" %>
<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As
System.EventArgs)
        ViewState("FavoriteColor") = TextBox1.Text
        Label1.Text = "Your data saved in ViewState."
    End Sub

    Protected Sub Button2_Click(ByVal sender As Object, ByVal e As
System.EventArgs)
        Dim color As String
        color = ViewState("FavoriteColor")
        Label1.Text = "Hi your favorite color is: " & color & " | " & city
    End Sub
End Sub
```

Store Value**Read Value**

5.1.2 The Query String

- Query strings are data that is appended to the end of a page URL. They are commonly used to hold data like page numbers or search terms or other data that isn't confidential.
- An example of a query string can look like **http://www.srl.co.il?a=1& b=2**. Query strings are included in bookmarks and in URLs that you pass in an e-mail. They are the only way to save a page state when copying and pasting a URL.
- The QueryString collection is used to retrieve the variable values in the HTTP query string.
- The HTTP query string is specified by the values following the question mark (?), like this:
- Several times in ASP.NET applications we need to transfer data or information provided by user from one aspx page to another.
- Query strings provide a simple but limited way of maintaining some state information. You can easily pass information from one page to another, But most

browsers and client devices impose a 255-character limit on the length of the URL.
http://yourdomainname.com/default.aspx?variable1=value1&variable2=value2

Suppose we have a textbox txtData and we want its value on other page than in code behind we would write in click event of btnGo

```
private void btnGO_Click(object sender, System.EventArgs e)  
{  
    Response.Redirect("Default2.aspx?Value=" +  
    txtData.Text);  
}
```

Or

```
private void btnGO_Click(object sender, System.EventArgs e)  
{  
    Response.Redirect("Default2.aspx?city=" +  
    txtData.Text + "&country=" + txtcountry.Text);  
}
```

Now to retrieve these values on other page we need to use **Request.QueryString()**, we can either retrieve them by variable name or by index

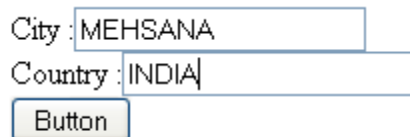
```
private void Page_Load(object sender, System.EventArgs e)  
{  
    txtCity.Text = Request.QueryString["city"];  
    txtCountry.Text = Request.QueryString["country"];  
}
```

Example: Sender**Design Code : QueryStringState_Sender.aspx**

```
<form id="form1" runat="server">  
    City :<asp:TextBox ID="txtcity" runat="server"></asp:TextBox><br />  
    Country :<asp:TextBox ID="txtcountry" runat="server">  
    </asp:TextBox><br />  
    <asp:Button ID="btnGO" runat="server" Text="Button" />  
</form>
```

VB Code : QueryStringState_Sender.aspx.VB

```
Partial Class QueryStringState_Sender Inherits System.Web.UI.Page  
    Protected Sub btnGO_Click(ByVal sender As Object, ByVal e As  
        System.EventArgs) Handles btnGO.Click  
        Response.Redirect("QueryStringState_Recive.aspx?city=" & txtcity.Text &  
            "&country=" & txtcountry.Text)  
    End Sub  
End Class
```

Output View :

City : MEHSANA
Country : INDIA
Button

Example: Receive**Design Code : QueryStringState_Recive.aspx**

```
<form id="form1" runat="server">
  <div>
    CITY :<asp:TextBox ID="txtcity" runat="server"></asp:TextBox><br />
    COUNTRY :<asp:TextBox ID="txtcountry" runat="server"></asp:TextBox>
  </div>
</form>
```

VB Code : QueryStringState_Recive.aspx.vb

Partial Class QueryStringState_Recive Inherits System.Web.UI.Page

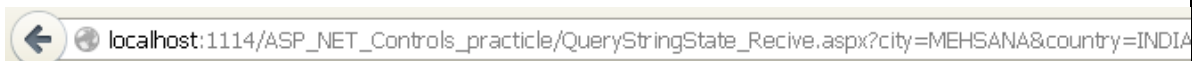
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load

txtcity.Text = Request.QueryString("city")

txtcountry.Text = Request.QueryString("country")

End Sub

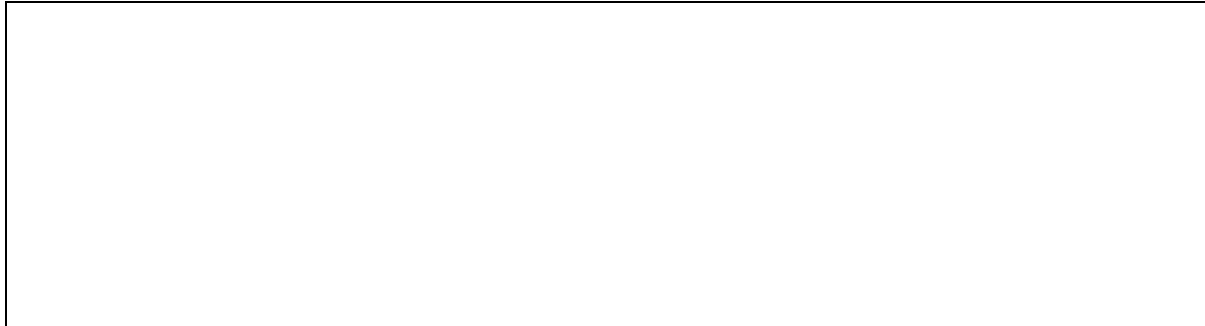
End Class

Output View :**URL :**

Most Visited ☐ Getting Started ☐ Customize Links ☐ Windows Marketplace

CITY : MEHSANA

COUNTRY : INDIA

**Example: 2****Design Code :****Default.aspx**

```
<html>
<head runat="server">
  <title>first Page</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      QueryString Example
      <asp:HyperLink ID="HyperLink1" runat="server"
NavigateUrl="~/Image.aspx?ImageID=1 " Text="Test QueryString">
</asp:HyperLink>
    </div>
  </form>
</body>
</html>
```

Image.aspx

```
<html>
<head runat="server">
  <title>Second Page</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
```

```
QueryString Example: Image View
    <asp:Label ID="Label1" runat="server"> </asp:Label>
    <br />
  </div>
</form>
</body>
</html>
```

VB Code : Image.aspx

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
    Dim ID As String = Request.QueryString("ImageID")
    Label1.Text = "ImageID: " & ID
End Sub
```

➤ Advantages:

- Server resources not required.
- Simple implementation
- Widespread support.

➤ Disadvantages:

- Size limitation: Some browser limits 2083 chars on the length of URLs.
- Security risk: Information is visible to the user.

5.1.3 Cross-Page Posting and Validation

- Cross-page posting means that you are posting data of one page to another page instead of posting data back to the same page
- To use cross-posting, you simply set PostBackUrl to the name of another web form where you would post data.
- When the user clicks the button, the page will be posted to that new URL with the values from all the input controls on the current page.
- Here's an example that defines a form with one text boxes and a button that posts to a page named CrossPage2.aspx

Example: Receive**Design Code : Crosspage1.aspx**

```
<html>
<head runat="server">
  <title>Cross Page Posting Example</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
<asp:Label ID="Label1" runat="server" Text="Enter The Value"></asp:Label>
      <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
      <br />
<asp:Button ID="Button1" runat="server" Text="Post_Back"
PostBackUrl="~/Crosspage2.aspx" />
    </div>
  </form>
</body>
</html>
```

Crosspage2.aspx

```
<html>
<head runat="server">
  <title>Untitled Page</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
<br />
<asp:Label ID="Label2" runat="server" Text="Label"></asp:Label>
      </div>
  </form>
```

```
</body>  
</html>
```

VB Code : Crosspage2.aspx.vb

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)  
Handles Me.Load  
    Label1.Text = PreviousPage.Title.ToString  
    Dim ename As TextBox =  
    DirectCast(PreviousPage.FindControl("TextBox1"), TextBox)  
    Label2.Text = ename.Text  
End Sub
```

- Note that this page checks for a null reference (Nothing in Visual Basic) before attempting to access the PreviousPage object.
- If no PreviousPage object exists, then no cross-page postback exists.
- ASP.NET uses some interesting sleight of hand to make this system work.
- The first time the second page accesses Page.PreviousPage, ASP.NET needs to create the previous page object.
- To do this, it actually starts the page processing but interrupts it just before the PreRender stage.
- Along the way, a stand-in Response object is created to silently catch and ignore any Response.Write() commands from the previous page.
- However, you still get some interesting side effects.
- For example, all the page events of the previous page are fired, including Page.Load, Page.Init, and even the Button.
- Click event, for the button that triggered the postback (if it's defined).
- Firing these events is mandatory, because they are required to properly initialize the page.

- Trace messages aren't ignored like Response messages are, which means you may see tracing information from both pages in a cross-posting situation

5.1.4 Cookies (create, set, add and expire cookie)

- A cookie is a small amount of data that is stored on user's computer
- Usually, information is stored as name-value pairs in Cookies.
- Cookies are used by websites to keep track of visitors. Every time a user visits a website, cookies are retrieved from user machine and help identify the user.
- A cookie is a small file that stores user information. Whenever a user makes a request for a page the first time, the server creates a cookie and sends it to the client along with the requested page and the client browser receives that cookie and stores it on the client machine either permanently or temporarily (persistent or non persistence).
- The next time the user makes a request for the same site, either the same or another page, the browser checks the existence of the cookie for that site in the folder. If the cookie exists it sends a request with the same cookie, else that request is treated as a new request.
- Cookies can be either temporary or persistent. Temporary cookies are stored in the client's browser. These cookies are saved only while your web browser is running.
- Persistent cookies, on the other hand, are stored on the hard disk of the client computer as a text file. They stay on your hard disk and can be accessed by web servers until they are deleted or have expired. These cookies can be retrieved by the Web application when the client establishes another session with it.
- Cookies are limited to 4096 bytes(4KB) in size and are only capable of storing strings.
- Before you can use cookies, you should import the System.Net namespace so you can easily work with the appropriate types, as shown here:
 - Imports System.Net

- Both the Request and Response objects (which are provided through Page properties) provide a Cookies collection.
- The important trick to remember is that you retrieve cookies from the Request object, and you set cookies using the Response object.
- To set a cookie, just create a new System.Net. HttpCookies object.

Syntax:**Creating Cookies Object**

```
Dim aCookie As New HttpCookie("lastVisit")
```

Setting value in Cookies Object

```
aCookie.Value("name") = "mike"  
aCookie.Expires = DateTime.Now.AddDays(1)
```

Add it to the current web response.

```
Response.Cookies.Add(aCookie)
```

Retrieve value from cookie

```
Dim userName As New String  
Dim userCookie As New HttpCookie= Request.Cookies("UserInfo")  
userName= userCookie("UserName")
```

OR

```
Request.Cookies("lastVisit").Value("name").ToString()
```

Example: 1**Design Code :**

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

```
<table>

  <tr><td>city :</td>

    <td>

      <asp:TextBox ID="txtcity" runat="server"></asp:TextBox>

    </td>

  </tr>

  <tr><td>country : </td>

    <td>

      <asp:TextBox ID="txtcountry" runat="server"></asp:TextBox>

    </td>

  </tr>

  <tr>

    <td>

      <asp:Button ID="btnSubmit" runat="server" Text="store" />

    </td>

  </tr>

</table>

<table>

  <tr>

    <td>
```

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

</td>

<td>

    <asp:Label ID="lblcountry" runat="server"></asp:Label>

</td>

</tr>

<tr>

    <asp:Button ID="btnfetch" runat="server" Text="fetch" />

</tr>

</table>

</div>

</form>
```

VB Code :

```
Partial Class cookie_store Inherits System.Web.UI.Page

    Protected Sub btnSubmit_Click(ByVal sender As Object, ByVal e As
    System.EventArgs) Handles btnSubmit.Click

        Dim cookie As New HttpCookie("information")

        cookie.Values("city") = txtcity.Text

        cookie.Values("country") = txtcountry.Text

        cookie.Expires = DateTime.Now.AddDays(1)
```



```
Response.Cookies.Add(cookie)

Response.Write("cookie generate")

End Sub

Protected Sub btnfetch_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles btnfetch.Click

    lblcity.Text = Request.Cookies("information").Values("city").ToString

    lblcountry.Text =
Request.Cookies("information").Values("country").ToString()

End Sub

End Class
```

Output View :

cookie generate	city :	<input type="text" value="MEHSANA"/>	
city :	<input type="text" value="MEHSANA"/>	country :	<input type="text" value="INDIA"/>
country :	<input type="text" value="INDIA"/>	<input type="button" value="store"/>	<input type="button" value="fetch"/>
<input type="button" value="store"/>	MEHSANA INDIA		

Example: 2**Design Code :**

```
<html>
<head runat="server">
    <title> Cookies</title>
</head>
```

```

<body>
<form id="form1" runat="server">
<div>
    asp.net Cookie example: Create a cookie</h2>
<p style="color:Teal">&nbsp;</p>
<asp:Label ID="lblName" runat="server"></asp:Label>
    <asp:Label ID="lblCountry" runat="server"    ></asp:Label>
</div>
</form>
</body>
</html>

```

VB Code :

Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
Handles Me.Load

```

Dim userCookie As New HttpCookie("UserInfo")
userCookie("Country") = "India"
userCookie.Expires = DateTime.Now.AddDays(3)

```

Set Cookie

Expire Cookie

```
Response.Cookies.Add(userCookie)
```

```
Label1.Text = "Cookie create successful"
```

```

Dim cookie As HttpCookie = Request.Cookies("UserInfo")
If cookie IsNot Nothing Then
    Dim country As String = cookie("Country")

```

Read Cookie

```

    Label2.Text = "Cookie Found and read " & "Country: " & country
End If

```

```
End Sub
```

➤ **Advantages:**

- Server resources not required.
- Simple implementation
- Configurable expiration rules

- Data persistent.
- **Disadvantages:**
 - Size limitation: Most browsers support maximum 4096 bytes cookies.
 - User configuration refusal: User can disable cookies in their browser.
 - Security risk: Can be tempered.
 - Cookies can be disabled on user browsers
 - Cookies are transmitted for each HTTP request/response causing overhead on bandwidth

5.1.5 Session State

- Session state is allow information to be stored in one page and accessed in another and it support any type of object.
- For every client, session data is stored separately, which means session data is stored on a per client basis Shown in fig.
- Every time you make a new request, ASP.NET generates a new session ID until you actually use session state to store some information
- Which is provided in an ASP.NET web page as the built-in Session object.
- Session state has no size limitations.
- Session state is used to store this information.
- The session-state is defining in the <sessionstate> section of the web.config file and stores the data specific to a user session in session variable. The session variable can be accessed from any page of a web application. When a user accesses a web page a session id for the user is created. The session ID is transferred between the servers.
- Session state is similar to application state, except that it is scoped to the current browser session.
- A session is the time for which a particular user interacts with a web application.
- Every client that uses the application will have separate session.
- During a session the unique identity of the user is maintained internally.

Storing a value in the Session is as simple as:

```
Session("Name") = "BSPP"
```

Or

```
Session.Add("Name", "BSPP")
```

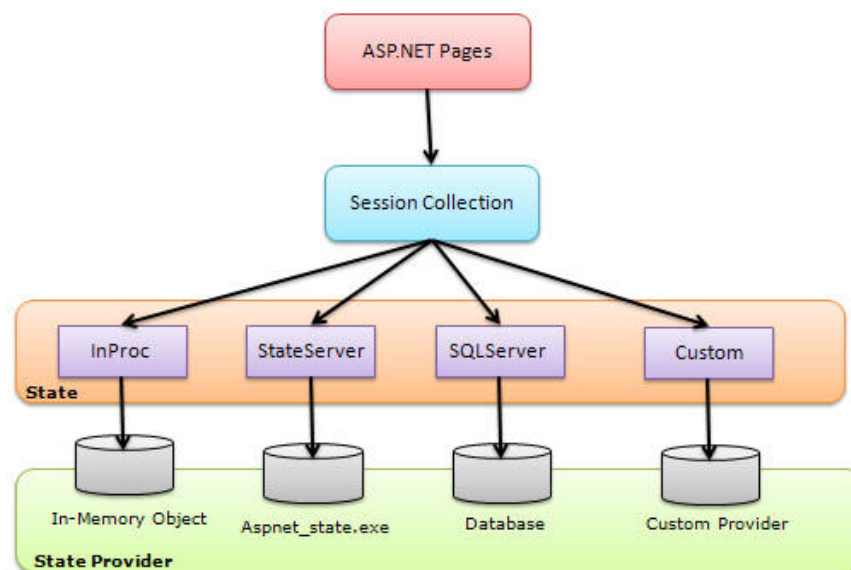
Retrieving a value in the Session is as simple as:

```
Dim Name As String = Session("Name")
```

Take a look at the pictorial flow:



Session Architecture



Session Mode and State Provider:

- We can choose the session state provider based on which session mode we are selecting
- By default the Session will be created within the same process that your web site runs in (InProc). This is controlled by a setting in the web.config file

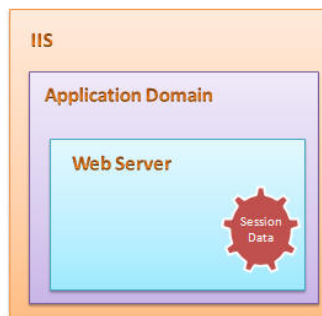
<sessionState mode="InProc" />

Off	Disables session state management.
InProc	Session state is stored locally in memory of ASP.NET worker process.
StateServer	Session state is stored outside ASP.NET worker process and is managed by Windows service. Location of this service is specified by

	stateConnectionString attribute.
SQLServer	Session state is stored outside ASP.NET worker process in SQL Server database. Location of this database is represented by sqlConnectionString attribute.

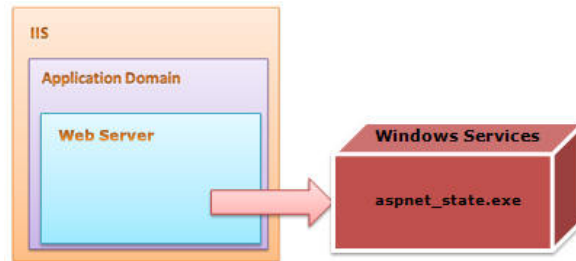
InProc Session Mode

- This is the default session mode in ASP.NET.
- Its stores session information in the current Application Domain.
- This is the best session mode for web application performance.
- But the main disadvantage is that, it will lose data if we restart the server



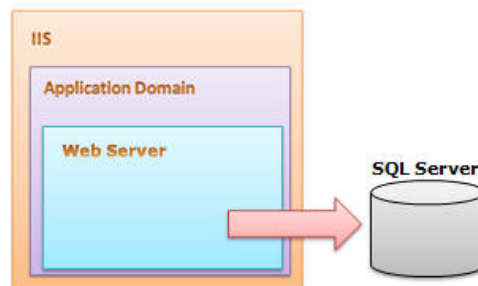
StateServer Session Mode

- This is also called Out-Proc session mode.
- StateServer uses a stand-alone Windows Service which is independent of IIS and can also be run on a separate server.
- This session state is totally managed by aspnet_state.exe.
- If you restart your ASP.NET process, your session data will still be alive.
- It also increases the cost of data access because every time the user retrieves session data, our application hits a different process.



SQLServer Session Mode

- This session mode provides 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.
- The main disadvantage of this session storage method is the overhead related with data serialization and de-serialization.



Member	Description
Count	The number of item in the current session collection.
SessionID	Provides a string with the unique session identifier for the current client.
Timeout	Enables you to specify the amount of time in minutes before the web server assume that the user has left and discards the session. The maximum value is 1 year.

Methods:

Session.Abandon()	Abandons (cancels) current session.
Session.Clear()	Enable you to clear all item from session state.
Session.Remove()	Enable you to remove a particular item from session state. Ex: Session.Remove(uname)
Session.RemoveAll()	Deletes all session state items.

You add items to session state by using the session object. For example, the page in add new item named message to session state that has the value 'Hello Word!'

Example: SessionSet**Design Code : SessionSet.aspx Page**

```
<form id="form1" runat="server">
    <h1> Session item added!!</h1>
</form>
```

VB Code : SessionSet.aspx.vb Page

```
Partial Class SessionState Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As
        System.EventArgs) Handles Me.Load

        Session("message") = "Hello Word !!"

        Response.Redirect("SessionGet.aspx")

    End Sub

End Class
```

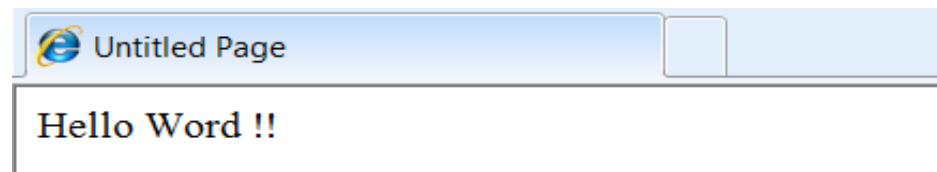
The page in listing illustrates how you can retrieve the value of an item that you have stored in session state.

Session: Get**Design Code : SessionGet.aspx Page**

```
<form id="form1" runat="server">  
    <asp:Label ID="lblMessage" runat="server" Text="Label"></asp:Label>  
</form>
```

VB Code : SessionGet.aspx.vb Page

```
Partial Class SessionGet Inherits System.Web.UI.Page  
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As  
        System.EventArgs) Handles Me.Load  
        lblMessage.Text = Session("message").ToString()  
    End Sub  
End Class
```

Output View :

- If the session time out because of inactivity. By default, a session time out after 20 idle minutes.
- If the programmer ends the session by calling Session.Abandon().

Example: 2

Design Code : Session.aspx

```
<html>
    <head runat="server">
        <title>Session Example</title>
    </head>
    <body>
        <form id="form1" runat="server">
            <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
            <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
            <br />
            <asp:Button ID="Button1" runat="server" Text="Store Session" />
            <asp:Button ID="Button2" runat="server" Text="Read Session" />

        </form>
    </body>
</html>
```

VB Code : session.aspx

```
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Session ("Name") = Textbox1.text
End Sub

Protected Sub Button2_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Label1.text = Session ("Name")
End Sub
```

Advantages and disadvantages of Session?

Following are the basic advantages and disadvantages of using session. I have describe in details with each type of session at later point of time.

Advantages:

- It helps maintain user state and data all over the application.
- It is easy to implement and we can store any kind of object.
- Stores client data separately.
- Session is secure and transparent from the user.

Disadvantages:

- Performance overhead in case of large volumes of data/user, because session data is stored in server memory.
- Overhead involved in serializing and de-serializing session data, because in the case of StateServer and SQLServer session modes, we need to serialize the objects before storing them.

5.1.6 Application State

- The application state is used to store the data corresponding to the global variables of an ASP.NET web application. the data in the application state is stored once and read many times. The application state uses the `HttpApplicationstate` class to store and share the data through the application. you can access the information stored in the application state by using the `HttpApplication` class property.
- Application state provides a method of storing data on server memory and it is global to whole application.
- These data are visible to entire application and shared by all active clients.
- That's why Application state variables are global variables for an ASP.Net application.
- Application state is based on the `System.Web.HttpApplicationState` class, which is provided in all web pages through the built-in `Application` object.
- Application state is similar to session state but Application State share data for entire application instead of particular user.

- A common example with application state is a global counter that tracks how many times an operation has been performed by all of the web application's clients.
- For example, you could create a global.aspx event handler that tracks how many sessions have been created or how many requests have been received into the application. Or you can use similar logic in the Page_Load event handler to track how many times a given page has been requested by various clients. Here's an example of the latter:

Creating Application Object

```
Application.Lock();  
Application["mydata"]="mydata";  
Application.Unlock();
```

Read Application Object

```
Value= Application["mydata"]
```

Example: 1**Design Code : session_application_state.aspx Page**

```
<form id="form1" runat="server">  
  <div>  
    enter name  
    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>  
    <br />  
    <asp:Button ID="Button1" runat="server" Text="Button" />  
    <br />
```

```
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
<br />
<asp:Label ID="Label2" runat="server" Text="Label"></asp:Label>
<br />
<asp:Label ID="Label3" runat="server" Text="Label"></asp:Label>
</div>
</form>
```

VB Code : session_application_state.aspx.vb Page

```
Partial Class session_application_vieq_state Inherits System.Web.UI.Page
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Me.Load
        Application("visits") = Int(Application("visits") + 1)
        Label2.Text = Application("visits")
        Label3.Text = Session("text").ToString()
    End Sub
    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Button1.Click
        Dim txtvalue As String = TextBox1.Text
        ViewState.Add("item", txtvalue)
        Dim item As String = ViewState("item").ToString
        Label1.Text = item
    End Sub
End Class
```

Global.asax Page:

```
<%@ Application Language="VB" %>

<script runat="server">

    Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)

        ' Code that runs on application startup
```

```
Application("visits") = 0

End Sub

Sub Application_End(ByVal sender As Object, ByVal e As EventArgs)

    ' Code that runs on application shutdown

End Sub

Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)

    ' Code that runs when an unhandled error occurs

End Sub

Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)

    ' Code that runs when a new session is started

    Session("text") = "jpp !!"

End Sub

Sub Session_End(ByVal sender As Object, ByVal e As EventArgs)

    ' Code that runs when a session ends.

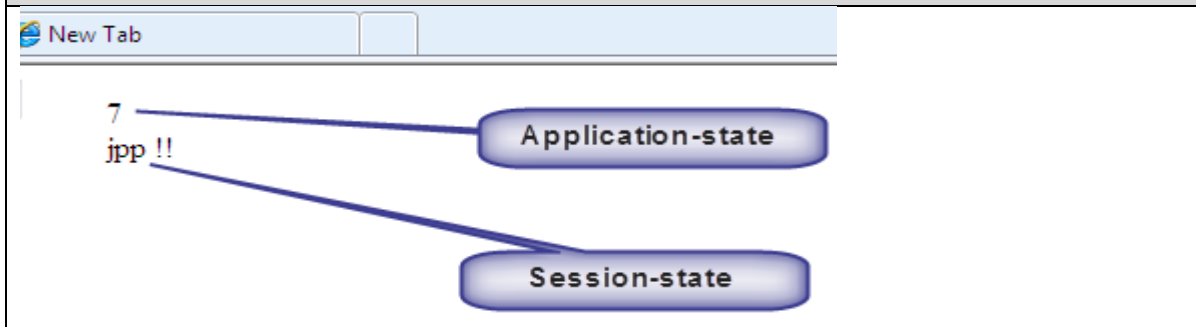
    ' Note: The Session_End event is raised only when the sessionstate mode

    ' is set to InProc in the Web.config file. If session mode is set to StateServer

    ' or SQLServer, the event is not raised.

End Sub

</script>
```

Output:**Example 1:****Design Code : Application.aspx**

```
<html>
  <head runat="server">
    <title>Application State</title>
  </head>
  <body>
    <form id="form1" runat="server">
      <div>
        asp.net application state example: Lock and UnLock<br />
        <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label><br />
        <asp:Button ID="Button1" runat="server" Text="Button" />
      </div>
    </form>
  </body>
</html>
```

VB Code : Application.aspx.vb

```
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Application.Lock()
```

```
Dim clickCounter As Integer = 0
If Application("ClickCounter") IsNot Nothing Then
    clickCounter = CInt(Application("ClickCounter"))
End If
clickCounter = clickCounter + 1
Application("ClickCounter") = clickCounter
Application.Unlock()
Label1.Text = "Button Clicked: " & clickCounter.ToString() & " times"
End Sub
```

- In other words, one request isn't counted because two clients access the counter at the same time.
- To prevent this problem, you need to use the Lock() and Unlock() methods, which explicitly allow only one client to access the Application state collection at a time, as follows.

Advantages:

- Simple implementation
- Application object memory released when we removed.
- Multi user can able to access application variable.
- Application wide scope-Other application can't access this application values.

Disadvantages:

- Application variable will exists until exit our application.
- If we do not have Lock() and Unlock, deadlock will occur.
- Its global variable so anyone can access within this application.
- Requires server memory.

5.2 Global.asax Application file

- The Global.asax, also known as the ASP.NET application file, is an optional file used to declare and handle application and session-level events.
- It contains code that is executed in response to application-level and session-level events raised by ASP.NET or by HTTP modules.
- The Global.asax file resides in the root directory of an ASP.NET application.

- At run time, Global.asax is parsed and compiled into a dynamically generated .NET Framework class derived from the `HttpApplication` base class.
- ASP.NET is configured so that any direct URL request for the Global.asax file is automatically rejected; external users cannot download or view the code in it.
- The Global.asax file is optional. You create it only if you want to handle application or session events.
- **How to create Global.asax**
 - Open Visual Studio 2005 or 2008 > Create a new website > Go to the Solution Explorer > Add New Item > Global Application Class > Add

5.2.1 Application Events

- You can handle two types of events in the global.asax file:
 - Events that always occur for every request. These include request-related and response-related events.
 - Events that occur only under certain conditions.
- Following fig show the sequence of application event which occur during application request.

The following are important events in the Global.asax file:

Event	Description
<code>Application_Start()</code>	Fires the first time an application starts.
<code>Application_BeginRequest()</code>	This event is called at the start of every request.
<code>Application_AuthenticateRequest()</code>	This method is called just before authentication is performed.
<code>Application_AuthorizeRequest()</code>	After the user is authenticated (identified), it's time to determine the user's permissions. You can use this method to assign special privileges.
<code>Application_ResolveRequestCache()</code>	This method is commonly used in conjunction with output caching
<code>Application_AcquireRequestState()</code>	This method is called just before session-specific information is retrieved for the client and used

	to populate the Session collection.
Application_PreRequestHandlerExecute()	This method is called before the appropriate HTTP handler executes the request.
Application_PostRequestHandlerExecute()	This method is called just after the request is handled.
Application_ReleaseRequestState()	This method is called when the session-specific information is about to be serialized from the Session collection so that it's available for the next request.
Application_UpdateRequestCache()	This method is called just before information is added to the output cache.
Application_EndRequest()	This method is called at the end of the request
Session_Start	Fires the first time when a user's session is started.
Session_End	Fires whenever a single user Session ends or times out.
Application_Error	Fires when an unhandled error occurs within the application.
Application_End:	Fires when the application ends or times out (Typically used for application cleanup logic).

Syntax

```
<%@ Application Language="VB" %>
<script runat="server">

    Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
        ' Code that runs on application startup
    End Sub

    Sub Application_End(ByVal sender As Object, ByVal e As EventArgs)
```

```
' Code that runs on application shutdown
End Sub

Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)
    ' Code that runs when an unhandled error occurs
End Sub

Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
    ' Code that runs when a new session is started
End Sub

Sub Session_End(ByVal sender As Object, ByVal e As EventArgs)
    ' Code that runs when a session ends.
    ' Note: The Session_End event is raised only when the sessionstate mode
    ' is set to InProc in the Web.config file. If session mode is set to StateServer
    ' or SQLServer, the event is not raised.
End Sub
</script>
```

Example:**Design Code :****Global.asax**

```
<%@ Application Language="VB" %>
<script runat="server">
    Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
        Application("OnlineNow") = 0
    End Sub

    Sub Application_End(ByVal sender As Object, ByVal e As EventArgs)
        Application("OnlineNow") = Nothing
    End Sub
End Sub
```

```
End Sub

Sub Application_Error(ByVal sender As Object, ByVal e As EventArgs)
    ' Code that runs when an unhandled error occurs
End Sub

Sub Session_Start(ByVal sender As Object, ByVal e As EventArgs)
    Application.Lock()
    Application("OnlineNow") = Application("OnlineNow") + 1
    Application.Unlock()
End Sub

Sub Session_End(ByVal sender As Object, ByVal e As EventArgs)
    Application.Lock()
    Application("OnlineNow") = Application("OnlineNow") - 1
    Application.Unlock()
End Sub
</script>
```

Globaldemo.aspx

```
<%@ Page Language="VB" %>
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:Label id="lblText" runat="server"></asp:Label><br />
        </div>
    </form>
</body>
</html>
```

VB Code : Globaldemo.aspx.vb

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As  
System.EventArgs) Handles Me.Load  
    Me.lblText.Text = Application("OnlineNow") & " Online"  
End Sub
```

5.3 ASP.NET Configuration

- Configuration in ASP.NET is managed with XML configuration files.
- All the information needed to configure an ASP.NET application, is stored in these configuration files
- The ASP.NET configuration files have several advantages
 - They are never locked
 - They are easily accessed and replicated
 - They are easy to edit and understand
- There are two types of configuration files:
 1. machine.config
 2. web.config

5.3.1 The Web.config File

- Web.config file is a configuration file for ASP.Net web application.
- A web application has one web.config file which keeps the configuration required for the application.
- It generate automatically when you create new web application.
- This file is written in different XML tags.

- The application configuration file, Web.config, contains additional settings which are applied to that particular web application.
- Web.config files reside in the application directory.
- Web.config files override machine.config settings for particular applications.
- Web.config files are written in XML with specific tags having specific meanings.
- We can create more than one web configuration file in the same application in separate folders.
- We can create database connections using the Web.config file.
- According to the user log in system environment, we can modify the Authentication and Authorization.
- We can set image paths for uploading images.
- Also, we can manage larger size files when uploading to the web server.
- If the System.Net.Mail email sending, we should configure the Web.config file.
- The Web.config file must contain only entries for configuration items that override the settings in the Machine.config file.
- At a minimum, the Web.config file must have the <configuration> element and the <system.web> element. These elements will contain individual configuration elements.
- The following example shows a minimal Web.config file:

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    'Child Element'
  </system.web>
</configuration>
```
- The first line of the Web.config file describes the document as XML-formatted and specifies the character encoding type. This first line must be the same for all .config files.
- The lines that follow mark the beginning and the end of the <configuration> element and the <system.web> element of the Web.config file.
- By themselves, these lines do nothing. However, the lines provide a structure that permits you to add future configuration settings.

- You add the majority of the ASP.NET configuration settings between the `<system.web>` and `</system.web>` lines.
- These lines mark the beginning and the end of the ASP.NET configuration settings.
- For example, consider the web request `http://localhost/A/B/C/MyPage.aspx`, where A is the root directory for the web application. In this case, multiple levels of settings come into play
 - The default `machine.config` settings are applied first.
 - The `web.config` settings from the computer root are applied next. This `web.config` file is in the same `Config` directory as the `machine.config` file.
 - If there is a `web.config` file in the application root A, these settings are applied next.
 - If there is a `web.config` file in the subdirectory B, these settings are applied next.
 - If there is a `web.config` file in the subdirectory C, these settings are applied last

5.3.2 Storing Custom setting in the Web.config File

Tag (Element)	Description
<code><authentication></code>	Configures the ASP.Net authentication.
<code><authorization></code>	Control the client access to web application.
<code><ConnectionString></code>	Configure the connection string for database connection.
<code><browserCaps></code>	Set to detect capabilities of browser.
<code><compilation></code>	Configures all compilation setting that ASP.Net uses.
<code><customErrors></code>	Provides information about error message for an ASP.Net application.
<code><httpRuntime></code>	Configures ASP.Net HTTP runtime setting.
<code><webServices></code>	Controls the setting of XML web services.
<code><Configuration></code>	The root element of a <code>web.config</code> file is always a <code><configuration></code> Every <code>web.config</code> file must have a configuration element.

<code><appSettings></code>	the “appsetting” section provides a way to define custom application setting for an ASP.NET application.
----------------------------------	--

There are some important setting that can be stored in the web.config file are as following The application settings allow storing application-wide name-value pairs for read-only access. For example, you can define a custom application setting as:

<Configuration>

The root element of a web.config file is always a `<configuration>`
Every web.config file must have a configuration element.

<appSettings>

the “appsetting” section provides a way to define custom application setting for an ASP.NET application.

```
<configuration>
  <appSettings>
    <add key="Application Name" value="MyApplication" />
  </appSettings>
</configuration>
```

For example,

```
<configuration>
  <appSettings>
    <add key="name" value="hello" />
  </appSettings>
</configuration>
```

.VB Code

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As
System.EventArgs)Handles Me.Load
```



```
Label1.Text = ConfigurationManager.AppSettings("name")
End Sub
```

<ConnectionString>

- The database connection string can be store in web.config file.
- Advantages of storing the connection string in web.config are that the modification in connection can be maintained at this file only, for whole application.
- The connection strings shows which database connection strings are available to the website. For example:

```
<connectionStrings>
  <add name="ApplicationServices" connectionString="data
source=. \SQLEXPRESS;Integrated
Security=SSPI;AttachDBFilename=|DataDirectory| \aspnetdb.mdf;User
Instance=true"
  providerName="System.Data.SqlClient" />
</connectionStrings>
```

The System.Web Element:

- The system.web element specifies the root element for the ASP.NET configuration section and contains configuration elements that configure ASP.NET Web applications and control how the applications behave.
- It holds most of the configuration elements needed to be adjusted in common applications. The basic syntax for the element:
- It's a fixed area of wev.config file. Programmer cannot add any elements to the structure.

```
<system.web>
  <anonymousIdentification>
```

```
<authentication>  
<authorization>  
<browserCaps>  
<compilation>  
<customErrors>  
<httpCookies>  
<httpHandlers>  
<httpRuntime>  
<pages>  
<sessionState>  
<webControls>  
<webServices>  
</system.web>
```

The following table provides brief description of some of common sub elements of the **system.web** element:

Compilation

- Configure various options like debug assemblies, default language to use in dynamic compilation model and other compiler option compiling custom resource files.

```
<compilation debug="true" targetFramework="4.0"/>
```

Authentication

- Authentication for ASP.Net application can be configure into web.config file.
- Window, form and passport authentication is configures in web.config file.
- It configures the authentication support. Basic syntax:

```
<authentication mode="[Windows | Forms | Passport | None]">  
  <forms>...</forms>  
  <passport/>
```

```
</authentication>
<authentication mode="Forms">
    <forms loginUrl="~/Account/Login.aspx" timeout="2880"/>
</authentication>
```

Authorization

- It configures the authorization support. Basic syntax:

```
<authorization>

    <allow .../>

    <deny .../>

</authorization>
```

CustomErrors:

- Defines custom error messages. Basic syntax:
- Error handling is main part in any web application ,error should be find and take suitable action should be taken.

```
<customErrors defaultRedirect="url" mode="On | Off | RemoteOnly">
    <error. . ./>

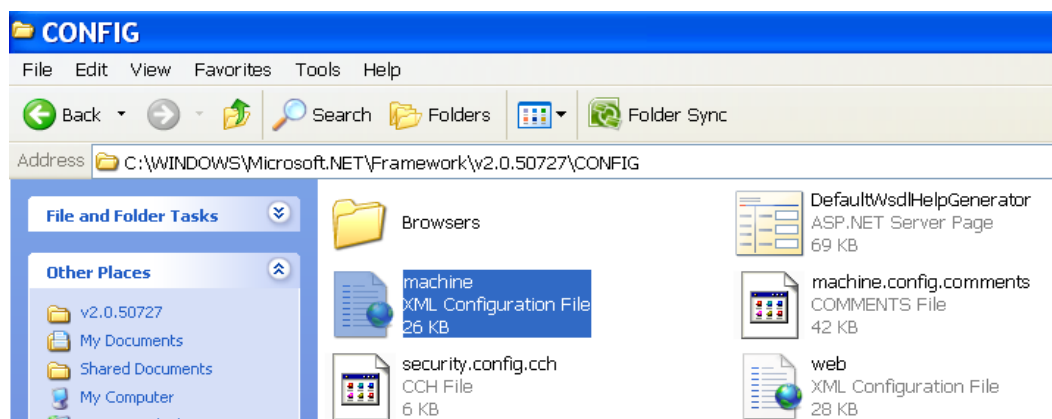
</customErrors>
```

```
<customErrors defaultRedirect="GenericError.html" mode=" RemoteOnly" >
    <error ststusCode="500" redirect="InternalError.html"/>
    <error ststusCode="404" redirect="FileNotFound.html"/>

</customErrors>
```

❖ UNDERSTANDING OF MACHINE.CONFIG FILE

- As web.config file is used to configure one ASP.Net application, same way Machine.config file is used to configure the application according to a particular machine. That is, configuration done in machine.config file is effected on any application that runs on a particular machine. Usually, this file is not altered and only web.config is used which configuration applications.
 - The setting made in the web.config file are applied to that particular web application only whereas the settings of machine.config file are applied to the whole asp.net application.
 - Only one Machine.Config file.
 - ASP.Net 2.0 provides another two files Machine.config.default and Machine.Config.comments.
 - The Machine.config.default as a bakup for the Machine.config file.
 - The Machine.config.comments file contain a description for each configuration section and explicit setting for the most commonly used value.
-
- This file is resides in the directory
c:\[WinDir]\Microsoft.NET\Framework\[Version]\Config.



Exercise

1. What is State Management? What is the requirement of it?
2. Explain client side state management with example.
3. Explain server side state management with example.
4. Compare Session & QueryString.
5. Compare application state & session state.
6. Is it necessary to write expiry date in cookies?
7. Can we pass multiple values with QueryString?
8. What is cross page posting? Explain with example.
9. Create a web page that stores the data of a web page in session state displays that data into another page.
10. Explain web.config with all necessary setting tags.

*******END OF CHAPTER*******