View State Vs. Session State Vs. Application State

**Introduction**  
  
There are a number of articles and blogs available about ASP.NET state management. I have tried writing this article for beginners explaining these concepts in simple language and step-by-step.  
  
This is a very basic article for beginners that show various state management techniques. It is very confusing for the developers when working with states in ASP.NET web applications. This article will help to understand how practically we should use these. Working with ASP.NET it is very important to understand and use the various states maintenance techniques available. Maintaining states during ASP.NET requests is very important from the application perspective.  
  
**Stateless Nature of Web application**  
Web applications run on HTTP protocols and this HTTP protocol is stateless in nature, meaning it does not remember state or retain any state between requests and responses.  
  
**Web application Processing**  
  
Whenever a web application is compiled, the entire source code of the project is compiled into an intermediate language and generates an output assembly that is a DLL residing in the bin folder of the project directory.  
  
When the application URL is requested by a user, The web server loads the requested project DLL into memory and creates an instance of the web form requested that results in the creation of a new instance of web form and all the controls and variables available on that requested web form.  
  
After creation it completes the page life cycle and renders the output as HTML and sends back the HTML output to the browser as a response. Then the web form object is immediately destroyed, meaning that the web form with their control is immediately something after rendering.  
  
Sample Code:

1. **namespace** Test
2. {
3. **public** partial **class** \_Default : Page
4. {
5. **int** counter = 0;
6. **protected** **void** Page\_Load(**object** sender, EventArgs e)
7. {
8. **if**(!IsPostBack)
9. {
10. TextBox1.Text = "0";
11. }
12. }
14. **protected** **void** Button1\_Click(**object** sender, EventArgs e)
15. {
16. counter = counter + 1;
17. TextBox1.Text =counter.ToString();
18. }
19. }
20. }

Explanation:  
  
When the application URL is requested it will “get request” and it will check on page load event and sets the text box to the value 0.  
  
When the button is clicked it would be a “post request” to the web server that creates a new instance of webform 1 that completes the process to generate a HTML and sends to the client browser. It does not remember the values from the last post call made to the web server. So here the result would be incremented by the value 1 but no matter how many times the button is clicked the result will be 1 only because due to the stateless nature of the HTTP protocol it is not retaining the value between post-back or client server requests.  
  
Here the state management techniques are relevant since they provide a way to remember/retain values between client calls to the server.  
  
**Various state management techniques**  
  
**A.** Client-side State Management:

* View State
* Hidden Field
* Cookies
* Control State

**B.** Server-side State Management:

* Session
* Application Object
* Caching

We can see a number of ways of doing state management as listed above. But I am going to explain View state, Session State and application state in this article.  
  
**View State**  
View State is a technique to maintain the state of controls during page post-back, meaning it stores the page value at the time of post-back (sending and receiving information from the server) of your page and the view state data can be used when the page is posted back to the server and a new instance of the page is created.  
  
View state data is nothing but a serialized base-64 encoded string stored in a hidden input field on the page and it travels between the browser and the server on every user request and response.

1. ViewState["VarName"]= store any thing

On the web server it is encoded on the page init event of the page's life and assigned back to the variable and after the HTML rendering and sent back to the browser with the values to client.

1. <input type="hidden" name="\_\_VIEWSTATE" id="\_\_VIEWSTATE" value="mlqif/yufT121LcPxuR5TVSuWVDJ7aU+2ONZy5gYWjTgmggCv5ed4OlAOS+jpYLWSI1hLbIA0cyrLI2YOZPo4RIESahtyWmLMhXbfEJ/GvJIvbfEE+JSHtDaw2iFc/kmz73T0oifsuZN6JzufE1ZI+NL7qrjzpOc9PTadu+Qxxokyw7cfV6ISa+fu9qSmjpYsxVtyxg/Z0QTyZBRaUiMbxWEJNlH3csR1d8HCPtoZ2s=" />

  
  
Sample Code:

1. **namespace** Test
2. {
3. **public** partial **class** \_Default : Page
4. {
5. **int** TestCounter = 0;
6. **protected** **void** Page\_Load(**object** sender, EventArgs e)
7. {
8. **if**(!IsPostBack)
9. {
10. TextBox1.Text = "0";
11. TextBox2.Text = "0";
12. }
13. }
14. **protected** **void** Button1\_Click(**object** sender, EventArgs e)
15. {
16. //With out View State
17. TestCounter = TestCounter + 1;
18. TextBox1.Text = TestCounter.ToString();
20. **if** (ViewState["counter"] != **null**)
21. {
22. TestCounter = (**int**)ViewState["counter"] + 1;
23. TextBox2.Text = TestCounter.ToString();
24. }
25. ViewState["counter"] = TestCounter;
27. }
28. }
29. }

View State advantages:

* Very easy to implement.
* Stored on the client browser in a hidden field as a form of Base64 Encoding String not encrypted and can be decoded easily.
* Good with HTTP data transfer

View State disadvantages:

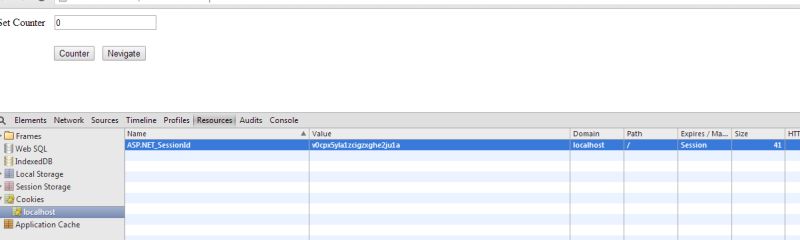
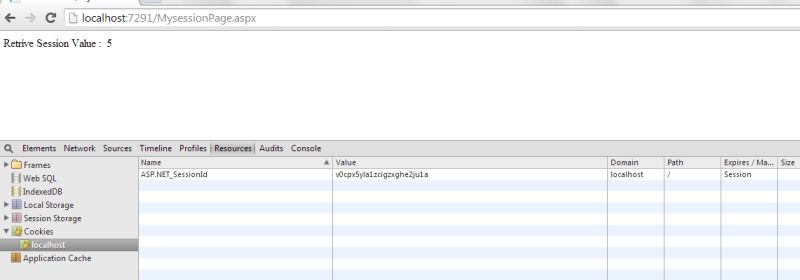
* The performance overhead for the page is larger data stored in the view state.
* Stored as encoded and not very safe to use with sensitive information.

**Where to Use**  
View state should be used when the user needs to store a small amount of data at the client browser with faster retrieval. The developer should not use this technique to retain state with larger data since it will create a performance overhead for the webpage. It should be used for sending data from one page to another. Not very secure to store sensitive information.  
  
**ASP.NET Session State**  
  
Session State is another state management technique to store state, meaning it helps in storing and using values from previous requests. Whenever the user requests a web form from a web application it will get treated as a new request. an ASP.NET session will be used to store the previous requests for a specified time period.

1. //Stored Textbox value
2. Session["Counter"] = TextBox3.Text;
4. //Stored Dataset
6. Session["ds"] = dsData;
7. Session variables are stored **in** a SessionStateItemCollection **object** that **is** exposed through the HttpContext.Session property.

Session variables are stored in a SessionStateItemCollection object that is exposed through the HttpContext.Session property.  
  
By default, an ASP.NET session state is enabled for all ASP.NET applications. Session state data is shared across all the web pages, in other words when navigating from one page session the data would available.

1. **namespace** Test
2. {
3. **public** partial **class** SessionTest : System.Web.UI.Page
4. {
5. **protected** **void** Page\_Load(**object** sender, EventArgs e)
6. {
7. **if** (!IsPostBack)
8. {
9. **if** (Session["Counter"] == **null**)
10. {
11. Session["Counter"] = 0;
12. }
13. TextBox1.Text = Session["Counter"].ToString();
15. }
16. }
17. **protected** **void** Button1\_Click(**object** sender, EventArgs e)
18. {
19. **if** (Session["Counter"] != **null**)
20. {
21. **int** SessionCounter = (**int**)Session["Counter"] + 1;
22. TextBox1.Text = SessionCounter.ToString();
23. Session["Counter"] = SessionCounter;
24. }
25. }
27. **protected** **void** Button2\_Click(**object** sender, EventArgs e)
28. {
29. Response.Redirect("MysessionPage.aspx");
30. }
31. }
32. }
34. After navigating to the page mysessionpage.aspx and retrieving value from session.
36. **namespace** Test
37. {
38. **public** partial **class** MysessionPage : System.Web.UI.Page
39. {
40. **protected** **void** Page\_Load(**object** sender, EventArgs e)
41. {
42. **if** (Session["Counter"] != **null**)
43. {
44. Label1.Text = Session["Counter"].ToString();
45. }
46. }
47. }
48. }

Session variables are single-user global data stored on the web server, meaning by default a session state variable is stored in the web server memory and is available across all pages but it will be for a single session.  
  
**SessionID**  
  
Client Machine Cookies are being used by a session to store a session id. This Session ID is being used by the web server to differentiate among requests of the same user or different ones. SessionID is nothing but a property to uniquely identify a browser with session data on the server. The SessionID value is randomly generated by ASP.NET and stored in a non-expiring session cookie in the browser. The SessionID value is then sent in a cookie with each request to the ASP.NET application.  
  
**Definition Reference: MSDN**  
  
Dear friends, unfortunately I am not explaining much about cookies here but it is important to spend some time for cookies also.  
  
Cookies are nothing but a small bit of text that accompanies requests and pages as they go between the web server and the browser. The cookie contains information about the web application that can be read whenever the user visits the site.   
  
  
  
  
  
If the session mode is Inproc and cookieless=true then:

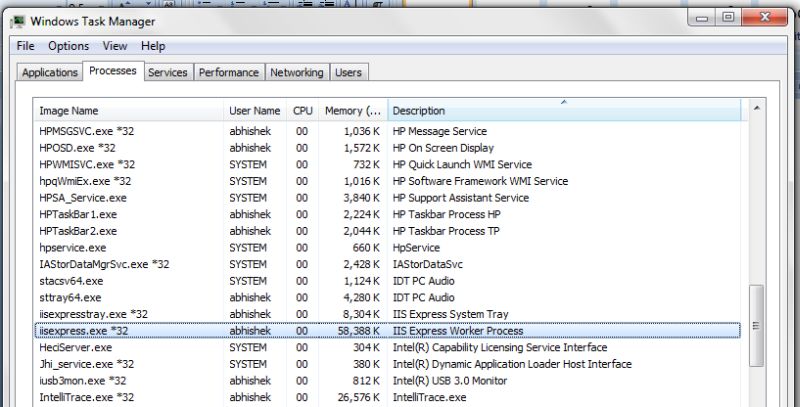
1. <sessionState mode="InProc" cookieless="true" customProvider="DefaultSessionProvider" >

The Session id will be sent as part of the URL in every request and if the session id is removed or changed it will be taken as a new request.   
  
<http://localhost:7291/(S(hkjtkowaucoyytjjfgha41ab))/Default.aspx>  
  
                                                     arrow  
                                               SessionID  
  
**Session Modes**  
  
Session modes explain how session variables are being stored, in other words what storage type is used by the variable and what is their behavior.  
  
The default behavior is in memory in an ASP .NET worker process.  
  
  
  
Here are the various session modes available in ASP.NET.

* Off
* InProc
* StateServer
* SQLServer
* Custom

Each mode has a different behavior in a web application. They have their own advantages and disadvantages.  
  
Guys, It is very important to understand about the session modes when you are working with an ASP.NET application with session variables as state management techniques. The session modes selected as mode in webconfig enables the ways session variable are stored and it will be then responsible for the application behavior.  
  
**Off**If an application has no requirement or need for session state then it's very important to use the off mode. By using this application performance will be better.  
  
**InProc Mode**  
InProc mode can be done in an ASP.NET web application using a configuration file by setting the mode attribute in the element SessionState.

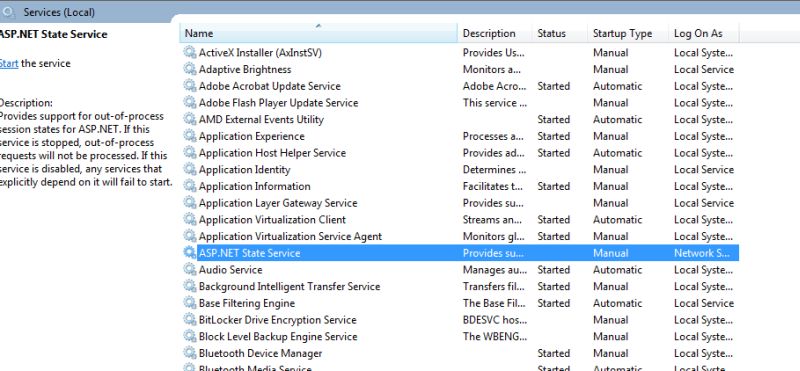
1. <sessionState mode="InProc" customProvider="DefaultSessionProvider">

When Inproc session state is used the session variables are being stored on the web server memory inside the ASP.NET worker process.  
  
Confused?  
  
Here is the MSDN Definition that says:  
  
ASP.NET runs within a process known as the ASP.NET worker process. All ASP.NET functionality runs within the scope of this process.  
  
  
  
Basically session variables are stored in the executable that is an IIS worker process. So if the request is coming from the same user the data will be available.  
  
Advantages of InProc:

* Easy to Implement.
* Complex Objects can be added without serialization.
* Best in performance compared to out-of-process modes.

Disadvantages of InProc :

* Not able to sustain the session values when the worker process/IIS is restarted. In that case data loss will happen witch make the application break.
* Scalability is a major problem.
* Not good for applications with a large user base.

**Where to Use**  
In Proc mode is best suited for the application that is hosted on a single server and mid size use base or the session variable used is not big, to avoid data loss and scalability issues. When there is a requirement for a web farm  or web garden deployments the “out of process “modes like state server or SQL Server modes are the best option.  
  
**State Server Session Mode**  
The disadvantage of session data loss is due to the worker process recycle that can be reduced using another mode, the state server mode.  
  
**Reference MSDN Definition**StateServer mode, that stores session state in a separate process called the ASP.NET state service. This ensures that session state is preserved if the web application is restarted and also makes session state available to multiple Web servers in a Web farm.  
  
ASP.NET is a Windows services that stores the session variable data in their process.  
  
**Procedure to set up state server mode**  
Go to Run then enter "Services.msc" then Start ASP.NET State Service.  
  
By default ASP.NET state service is in manual mode.  
  


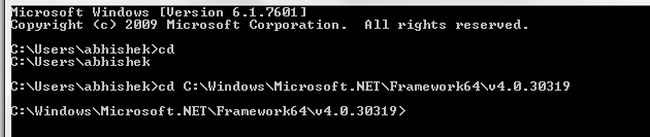
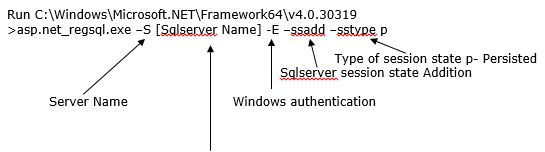
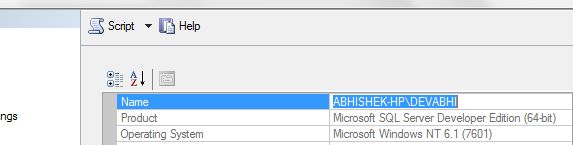
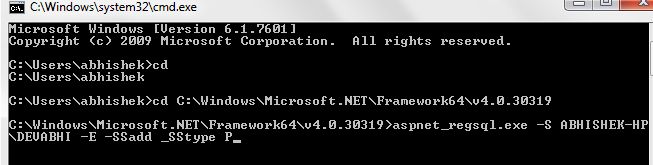
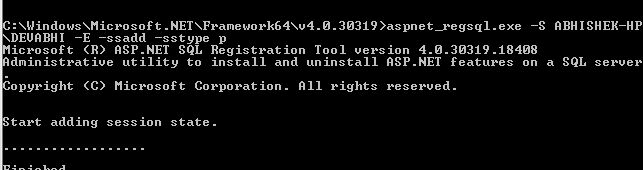
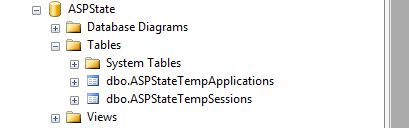
1. <sessionState mode="StateServer" customProvider="DefaultSessionProvider" stateConnectionString="tcpip=localhost:42424">
3. stateConnectionString="tcpip=localhost : 42424"

Arrow             Arrow  
**Server Name/IP Port**  
  
StateConnectionstring basically consists of the following;  
  
**Server/system:** where an ASP.NET Windows service is running.  
**Port:** By default the state service listens to the TCP port 42424 that is configurable using the registry.  
  
Here in my example I am using my machine's Windows service to implement the state server mode. It could be a separate dedicated machine or a web server in the network depending on requirements.  
  
So now the session state variables are stored in a Widows service of the my machine so the recycling process of the ASP.NET worker process does not have any impact on the session variables.  
  
Advantages of State Server Mode:

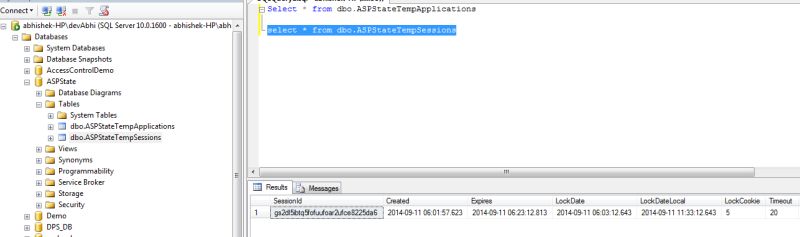
* Worker process recycling does not impact session variable data
* Can be stored on the same web server or different dedicated machine

Disadvantage of State Server Mode:

* Restart of sate service could lead to session data loss.
* Slower than in proc mode

**SQL Server Session Mode**  
When the Session mode of an ASP.NET application is set by SQL Server then the session variables are being stored in the SQL Server database. This mode also provides a reliable way to store session data and preserve values after an IIS or web server restart and best suited for the web farm like application deployment.  
  
Since we are storing the session variables in a SQL Server database we need a databse and table to store the data.  
  
At the location "C:\Windows\Microsoft.NET\Framework64\v4.0.30319" an exe named aspnet\_regsql is present.  
  
It depends on the Window's operating system and the version of .Net framework installed on the machine. In my case I have Windows 7 and framework version 4.0 installed.  
  
**Procedure**  
  
Go to Run then enter cmd then open a command window then enter "cd C:\Windows\Microsoft.NET\Framework64\v4.0.30319".  
  
This will navigate to the location where aspnet\_regsql.exe is present.  
  
  
  
Run the tool from that location using the command below.  
  
   
  
  
  
  
  
  
  
  
**Web Config**  
Do change the mode to SQL Server and provide the SQL connection string.

1. <sessionState mode="SQLServer"  customProvider="DefaultSessionProvider"
2. sqlConnectionString="Data Source=abhishek-HP\devAbhi;integrated security=SSPI">

After doing all the required set up now our application is ready to run with SQL Server session mode.  
  
  
  
Advantages of SQL Server Mode

* Very secure and most reliable option for the session management.
* Session data will be able to survive after worker process restart or state window service restart.
* Most scalable compared to the other modes.
* Most suited for web garden or web farm type deployments and able to handle larger data in the session.

Disadvantages of SQL Server mode

* Slow performance
* Overhead for serialization and deserialization of complex data.

**Where to Use**  
Here we have learned about session state and various modes to store data in session variables. Every mode has some advantages and disadvantages for use in web applications. Basically it depends on the application behavior, use base and kind of deployment which session should be used. Guys, be careful when choosing the session modes since it leads to performance issues and data loss that hamper the web application.  
  
**Application State**  
  
**The MSDN Definition says:**Application state is a data repository available to all classes in an ASP.NET application. Application state is stored in memory on the server and is faster than storing and retrieving information in a database. Unlike session state, which is specific to a single user session, application state applies to all users and sessions.  
  
Application state is stored in an instance of the HttpApplicationState class. This class exposes a key-value dictionary of objects.  
  
Application state variables are also used to store data when navigatiing from one page to another. It's multi-user Global data meaning it will be accessible across all pages and all sessions. Application state variables are stored on the web server in ASP.NET worker process memory.  
  
**Sample Code**  
  
Addition of data in application variables.

1. **namespace** Test
2. {
3. **public** partial **class** applicationState : System.Web.UI.Page
4. {
5. **protected** **void** Page\_Load(**object** sender, EventArgs e)
6. {
7. **if** (!IsPostBack)
8. {
9. **if** (Application["Counter"] == **null**)
10. {
11. Application["Counter"] = 0;
12. }
13. TextBox1.Text = Application["Counter"].ToString();
15. }
16. }
18. **protected** **void** Button1\_Click(**object** sender, EventArgs e)
19. {
20. **if** (Application["Counter"] != **null**)
21. {
22. **int** ApplicationCounter = (**int**)Application["Counter"] + 1;
23. TextBox1.Text = ApplicationCounter.ToString();
24. Application["Counter"] = ApplicationCounter;
25. }
27. }
29. **protected** **void** Button2\_Click(**object** sender, EventArgs e)
30. {
31. Response.Redirect("ApplicationStateTest.aspx");
32. }
33. }
34. }

Reading data from application variables:

1. **namespace** Test
2. {
3. **public** partial **class** ApplicationStateTest : System.Web.UI.Page
4. {
5. **protected** **void** Page\_Load(**object** sender, EventArgs e)
6. {
7. **if** (Application["Counter"] != **null**)
8. {
9. Label1.Text = Application["Counter"].ToString();
10. }
11. }
12. }
13. }

An application state variable does not have any default time unlike session variables. It will end when the application hosting process is restarted or the application ends.  
  
Application state variable data is not reliable for web farms or web garden type deployment since they are not shared across multiple web servers or multiple worker processes on the same machine so using application variable data could cause the problem of data loss and the application breaks.  
  
An application state variable is not thread safe, meaning multiple users from a different session can access the variable and can manipulate the variable.  
  
To ensure the data integrity and resolve concurrency issues while using application state variables, lock and unlock methods should be used.  
  
**Important Note**  
  
**Web farm:** It's a situation where the web application is deployed on different servers with load balancer.  
 **Web garden**: It's a situation where the web application is deployed on the same server with multiple worker processes.  
  
Advantages of application state:

* Application variable data is multi-user global data stored in memory.
* Easy to access.
* Fast retrieval.

Disadvantages of application state:

* Application variable data is not able to survive the IIS restart and worker process recycling.
* Not suited for web farm and web garden like deployment situation.

**Where to Use**  
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.  
  
**Conclusion**  
Guys, in the preceding explanation of view state, the session state and application state management techniques all have some advantages and disadvantages in web applications. We should very intelligently pick the technique analyzing our application usage and functionality used in the application.  
  
**MSDN References:**[Session-State Modes](http://msdn.microsoft.com/en-us/library/ms178586.aspx)  
[ASP.NET Application State Overview](http://msdn.microsoft.com/en-us/library/vstudio/ms178594(v=vs.100).aspx)

<http://www.c-sharpcorner.com/UploadFile/de41d6/view-state-vs-session-state-vs-application-state/>