

## FAQ's: Related to State in ASP.NET [ Answered Below ]

Q: Session state works on some browsers, but not on others. Why?

Q: In InProc mode, why do I lose all my session occasionally?

Q: Session states works on some web servers but not on others.

Q: Why isn't session state available?

Q: Why isn't Session\_End fired?

Q: Why are my Session variables lost frequently when using InProc mode?

Q: Why does the SessionID remain the same after the Session times out or abandoned?

Q: Why does the SessionID changes in every request?

Q: What is the difference between Session.Abandon() and Session.Clear()?

Q: Is the session Timeout attribute a sliding timeout value?

Q: Can I share session state between ASP.NET and ASP pages?

Q: Can I share session state between web applications (i.e. "virtual directories" or "applications" in IIS)?

Q: What kinds of object can I store in session state?

Q: Why did my request hang after I switch to SQLServer mode?

Q: How come Response.Redirect and Server.Transfer is not working in Session\_End?

Q: In Session\_End, do I have a valid HttpSessionState object and HttpContext object?

Q: How do I use session state with web services?

Q: I am writing my own HttpHandler. Why is session state not working?

Q: I am using a webfarm, and I lost session state when directed to some web servers.

Q: If using "cookieless", how can I redirect from a HTTP page to an HTTPS page?

Q: Does session state have a locking mechanism that serializes the access to state?

Q: How do I detect a session has expired and redirect it to another page?

Q: In Session\_End, I tried to do some cleanup job using SQL but it failed. Why?

Q: I am using SQLServer mode. Why aren't my sessions expiring?

Q: I have a frameset page which has an HTM extension, and I found out each frame it contains displays a different session id on the first request. Why?

Q: I set EnableSessionState to "ReadOnly", but in InProc mode I can still modify the session. Why is that?

Q: I set "cookieless" to true, and now my session variables are lost after a Redirect. Why?

Q: What are the disadvantages of setting cookieless to true?

Q: In InProc mode, I change the timeout value of a session programmatically, and that causes my Session\_End to be called. Why?

Q: In SqlServer mode, can I store my session state in a database other than tempdb?

Q: How can I avoid putting plain password for my sql connection?

Q: What SQL permissions do I need when using SqlServer mode?

Q: Can I write my own custom session state module?

Q: How does (de)serialization work in SqlServer and State Server mode?

Q: How can I secure my state server?

Q: Can I subscribe to SessionStateModule.End event using a non-global.asax handler method?

Q: Can different apps store their session state in different databases on the same SQL server?

---

**Q: Session state works on some browsers, but not on others. Why?**

A: Assume you aren't using cookieless, you should make sure your browsers support cookie. Also see this KB: See <http://support.microsoft.com/default.aspx?scid=kb;EN-US;q316112>

**Q: In Proc mode, why do I lose all my session occasionally?**

A: Please see the "Robustness" section in the "Understanding session state modes" section of of this article.

**Q: Session states works on some web servers but not on others.**

A: Maybe machine name problem.

See <http://support.microsoft.com/default.aspx?scid=kb;EN-US;q316112>

**Q: Why isn't session state available?**

A:

- First, check your web.config, machine.config and your page directive to make sure you have enabled session state.

Reference:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconsessionstate.asp>  
<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpgenref/html/cpconpage.asp>

- Also, please note that session state isn't available just everywhere, anytime. It is available only after the `HttpApplication.AcquireRequestState` event is called. For example, it is NOT available in the `Application_OnAuthenticateRequest` handler inside `global.asax`.

For details, see:

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/cpguide/html/cpconhandlingpublicevents.asp>

- Lastly, make sure `System.Web.SessionState.SessionStateModule` is included the `<httpModules>` in your config files. A common case is that SharePoint application will remove this module from their web.config files (for performance reason), and thus session state isn't available.

**Q: Why isn't Session\_End fired?**

A: This is one of the most frequently asked question.

1. Remember `Session_End` event is supported only in InProc mode.
2. `Session_End` won't be fired if you close your browser. HTTP is a stateless protocol, and the server has no way to know if your browser has closed or not.
3. `Session_End` will be fired only (i) after n minutes of inactivity (n = timeout value), or (ii) if someone calls `Session.Abandon()`.
4. For case (i) (pt. 3), `Session_End` will be run by a background thread, which implies:

a. Your code in `Session_End` is running using the worker process account. You may have permission problem if you're accessing resource such as database.

b. If an error happens in `Session_End`, it will fail silently.

5. For case (ii), please note that in order for `Session_End` to be fired, your session state has to exist first. That means you have to store some data in the session state and has completed at least one request.

6. Again for case (ii), `Session_End` will be called only if the abandoned session is actually found. As a result, if you create and abandon a session inside the same request, because the session hasn't been saved and thus can't be found, `Session_End` won't be called. This is a bug in v1 and upcoming v1.1.

**Q: Why are my Session variables lost frequently when using InProc mode?**

A: Probably because of application recycle.

See <http://support.microsoft.com/default.aspx?scid=kb;en-us;Q316148>

In v1, there is also a bug that will cause worker process to restart. It's fixed in SP2 and v1.1. See <http://support.microsoft.com/default.aspx?scid=kb;EN-US;321792>

For more details about app recycling, see my other FAQ:

<http://www.asp.net/Forums/ShowPost.aspx?tabindex=1&PostID=232621>

**Q: Why does the SessionID remain the same after the Session times out or abandoned?**

A: Even though the session state expires after the indicated timeout period, the session ID lasts as long as the browser session. What this implies is that the same session ID can represent multiple sessions over time where the instance of the browser remain the same.

**Q: Why does the SessionID changes in every request?**

A: This may happen if your application has never stored anything in the session state. In this case, a new session state (with a new ID) is created in every request, but is never saved because it contains nothing.

However, there are two exceptions to this same session ID behavior:

- If the user has used the same browser instance to request another page that uses the session state, you will get the same session ID every time. For details, see "Why does the SessionID remain the same after the Session times out?"
- If the Session\_OnStart event is used, ASP.NET will save the session state even when it is empty.

**Q: What is the difference between Session.Abandon() and Session.Clear()?**

A: The major practical difference is that if you call Session.Abandon(), Session\_End will be fired (for InProc mode), and in the next request, Session\_Start will be fired. Session.Clear() just clears the session data without killing it.

**Q: Is the session Timeout attribute a sliding timeout value?**

A: The session Timeout is a sliding expiration time, meaning whenever your page access session state, the expiration time will be moved forward. Please note that as long as a page has NOT disabled session state, it will access the session automatically when requested.

**Q: Can I share session state between ASP.NET and ASP pages?**

A: No. But there is an MSDN article on how to work around it:

<http://www.msdn.microsoft.com/library/default.asp?url=/library/en-us/dnasp/html/ConvertToASPNET.asp>

There are also some 3rd party solutions:

(Disclaimer: the 3rd party solutions listed below is only for the convenience of customers, and does NOT imply any endorsement from Microsoft)

<http://www.consonica.com/solutions/dotnet/statestitch/index.html> <http://www.sessionbridge.com>

**Q: Can I share session state between web applications (i.e. "virtual directories" or "applications" in IIS)?**

A: No. Sorry.

**Q: What kinds of object can I store in session state?**

A: It depends on which mode you are using:

- If you are using InProc mode, objects stored in session state are actually live objects, and so you can store whatever object you have created.
- If you are using State Server or SQL Server mode, objects in the session state will be serialized and deserialized when a request is processed. So make sure your objects are serializable and their classes must be marked as so. If not, the session state will not be

saved successfully. In v1, there is a bug which makes the problem happen unnoticed in SQLServer mode and will make your request hang. The hanging problem is fixed in v1.1. The QFE fix for KB 324479 also contains the fix for this problem. The problem will be fixed in v1 SP3 too.

See this KB for more info:

<http://support.microsoft.com/directory/article.asp?ID=KB;EN-US;q312112>

**Q: Why did my request hang after I switch to SQLServer mode?**

A: Please read the answer for the question: "What kinds of object can I store in session state?"

**Q: How come Response.Redirect and Server.Transfer is not working in Session\_End?**

A: Session\_End is fired internally by the server, based on an internal timer. And thus there is no HttpRequest associated when that happens. That is why Response.Redirect or Server.Transfer does not make sense and will not work.

**Q: In Session\_End, do I have a valid HttpSessionState object and HttpContext object?**

A: You will have the HttpSessionState object available. Just use 'Session' to access it. For HttpContext, it is not available because this event is not associated with any request.

**Q: Will my session state be saved when my page hit an error?**

A: If you're using StateServer or SQLServer mode, for data integrity reason session state module will not save any changes to session state if there is an error. To work around this, you can call Server.ClearError in your exception handler.

For InProc mode, changes are made directly to the in-memory objects and so whatever changes you've made so far will stay in the memory. However, if it's a new session, ASP.NET will not insert your session state into the internal table, and thus your session state will NOT be saved, unless you call Server.ClearError in your exception handler.

**Q: How do I use session state with web services?**

A: The extra trick needed is on the caller side. You have to save and store the cookies used by the web service. See the MSDN documentation on HttpWebClientProtocol.CookieContainer property.

However, please note if you're using proxy object to call a web service from your page, the web service and your page cannot share the same session state due to architecture limitation.

This can be done if you call your web service through redirect.

**Q: I am writing my own HttpHandler. Why is session state not working?**

A: Your HttpHandler has to implement the "marker" interface IRequiresSessionState or IReadOnlySessionState in order to use session state.

**Q: I am using a webfarm, and I lost session state when directed to some web servers.** A: For session state to be maintained across different web servers in the web farm, the Application Path of the website (For example \LM\W3SVC\2) in the IIS Metabase should be identical (case sensitive) in all the web servers in the web farm. See KB 325056 for details.

**Q: If using "cookieless", how can I redirect from a HTTP page to an HTTPS page?** A: Try this:

```
String originalUrl = "/fxtest3/sub/foo2.aspx";  
String modifiedUrl = "https://localhost" + Response.ApplyAppPathModifier(originalUrl);  
Response.Redirect(modifiedUrl);
```

**Q: Does session state have a locking mechanism that serialize the access to state?**

A: Session state implements a reader/writer locking mechanism:

- A page (or frame) that has session state write access (e.g. <%@ Page EnableSessionState="True" %>) will hold a writer lock on the session until the request finishes.
- A page (or frame) that has session state read access (e.g. <%@ Page

EnableSessionState="ReadOnly" %>) will hold a reader lock on the session until the request finishes.  
- Reader lock will block a writer lock; Reader lock will NOT block reader lock; Writer lock will block all reader and writer lock.  
- That's why if two frames both have session state write access, one frame has to wait for the other to finish first.

**Q: How do I detect a session has expired and redirect it to another page?**

A: It's a much requested feature, and unfortunately there is no easy way to do it right now. We will look into in the next major version. In the meantime, if you are using cookie, you can store a marker in your cookie so you can tell the difference between "fresh browser + new session" and "old browser + expired session". Below is a sample code that will redirect the page to an expired page if the session has expired.

```
void Session_OnStart(Object sender, EventArgs e) {
    HttpContext context = HttpContext.Current;
    HttpCookieCollection cookies = context.Request.Cookies;

    if (cookies["starttime"] == null) {
        HttpCookie cookie = new HttpCookie("starttime", DateTime.Now.ToString());
        cookie.Path = "/";
        context.Response.Cookies.Add(cookie);
    }
    else {
        context.Response.Redirect("expired.aspx");
    }
}
```

**Q: In Session\_End, I tried to do some cleanup job using SQL but it failed. Why?**

A: First, Session\_End is supported only in InProc mode.  
Second, Session\_End is run using the account which runs the worker process (aspnet\_wp.exe), which can be specified in machine.config. Therefore, in your Session\_End, if you connect to SQL using integrated security, it will use that worker process account credential to connect, and may fail depending on your SQL security settings.

**Q: I am using SQLServer mode. Why aren't my sessions expiring?**

A: In SQLServer mode, session expiration is carried out by the SQL Agent using a registered job. Make sure your SQL Agent is running.

**Q: I have a frameset page which has an HTM extension, and I found out each frame it contains display a different session id on the first request. Why?**

A: The reason is that your frameset page is an HTM file instead of an ASPX file.

In normal case, if the frameset is an aspx file, when you request the page, it will first send the request to the web server, receive an asp.net session cookie (which holds the session id), and then the browser will send individual requests for the frames, and each request will carry the same session id.

However, since your frameset page is an htm file, the first request comes back without any session cookie because the page was serviced by ASP and not ASP.NET. Then again your browser sends out individual requests for each frame. But this time each individual request will NOT carry any session id, and so each individual frame will create its own new session. That's why you will see different session ids in each frame. The last request that comes back will win by overwriting the cookie written by the previous two requests. If you do a refresh, you will see them having the same session id.

This behaviour is by-design, and the simple solution is to change your frameset page to .aspx.

**Q: I set EnableSessionState to "ReadOnly", but in InProc mode I can still modify the**

**session. Why is that?**

A: Even those enableSessionState is marked as ReadOnly, but in InProc state, the user can still modify the session. The only difference is that the session will not be locked during the request. This limitation is by-design. And I sorry that it's not documented in MSDN.

**Q: I set "cookieless" to true, and now my session variables are lost after a Redirect. Why?**

A: If you're using cookieless, you must use relative path (e.g. ../hello.aspx) instead of absolute path (e.g. \foo\bar\hello.aspx). If you use absolute path, ASP.NET cannot preserve your session ID in the URL.

**Q: What are the disadvantages of setting cookieless to true?**

A: Setting Cookieless=true implies some restrictions, mainly:

1. You cannot use absolute link in your pages.
2. You have to do extra thing to switch between http and https pages in your application.
3. If your customer send a link to a friend, the URL will contain the session ID and both users could be using the same session ID at the same time.

**Q: In InProc mode, I change the timeout value of a session programmatically, and that causes my Session\_End to be called. Why?**

A:It is a a bug of InProc mode. If you change a session's timeout to a different value, Session\_End will be called (but not Session\_Start). We will looking into that in v2 and see if we can fix it.

**Q: In SqlServer mode, can I store my session state in a database other than tempdb?**

A:Yes. See KB311209.

**Q: How can I avoid putting plain password for my sql connection?**

A: See sql trusted connection, or put the connection string as encrypted data in the registry. For details, KB 329250 and 329290.

**Q: What SQL permissions do I need when using SqlServer mode?**

A:The caller needs EXEC permission on the following stored procedures in ASPState:

dbo.TempGetAppID  
dbo.TempGetStateItem  
dbo.TempGetStateItemExclusive  
dbo.TempReleaseStateItemExclusive  
dbo.TempInsertStateItemLong  
dbo.TempInsertStateItemShort  
dbo.TempUpdateStateItemLong  
dbo.TempUpdateStateItemShort  
dbo.TempUpdateStateItemShortNullLong  
dbo.TempUpdateStateItemLongNullShort  
dbo.TempRemoveStateItem  
dbo.TempResetTimeout

For version 1.1, you also need EXEC permission on the following stored procs in ASPState:

dbo.TempGetStateItem2  
dbo.TempGetStateItemExclusive2

Please note that the owner of the sprocs must have SELECT/INSERT/UPDATE/DELETE permissions on the session state tables (dbo.ASPStateTempSessions and dbo.ASPStateTempApplications). The owner is the account which ran the installsqlstate.sql (or the persistent version (see KB311209)) to install the tables/sprocs/databases needed for sql session state.

Also note that if your session state table lives in tempdb (by default), any permission settings on that table will be lost if you recycle SQL server.

### **Q: Can I write my own custom session state module?**

A: In v1 and v1.1, there are very limited support for writing your own custom session state module. I can suggest two approaches:

Approach #1 - replace SessionStateModule

In this approach, you write your own custom module and replace the original one in <httpModules>. This way, you simply roll out your own implementation.

Drawbacks:

- i. Since the constructor of HttpSessionState isn't public, you can't create your own instance of it and stick it to HttpContext like the way ASP.NET SessionStateModule does. Instead, you have to expose your session through some custom class, and the user can't access it through Page.Session nor HttpContext.Session.
- ii. You have to implement all the code to handle session and session id creation/maintenance/expiry, etc. That can be quite tedious.

Approach #2 - complement SessionStateModule

In this approach, you still rely on original ASP.NET SessionStateModule for session and session id creation/maintenance/expiry, etc. But at the same time, you write your own custom module, and have it subscribed to at least the following HttpApplication events: (a) PreRequestHandlerExecute - When this event happens, AcquireRequestState has been fired already, and thus HttpContext.Session is available. In your even handler, you read your data from your own store and copy them to HttpContext.Session.

(b) PostRequestHandlerExecute - This event is fired before ReleaseRequestState. Your event handler will save all the data in HttpContext.Session to your own store, and call HttpContext.Session.Clear() to remove all items so they won't be saved by ASP.NET session state.

(c) EndRequest - If an error occurs during the execution of the page, PostRequestHandlerExecute might be skipped and EndRequest will be fired directly. Your module might need to do some maintenance work here.

Please note that you shouldn't subscribe to AcquireRequestState and ReleaseRequestState like ASP.NET session state module does because if 2 modules subscribe to an event, there is no guarantee which module will get called first.

Drawbacks:

- i. Your custom session data won't be available before PreRequestHandlerExecute and after PostRequestHandlerExecute.
- ii. You need to figure out how to cleanup during session expiry or abandonment (in InProc, may piggyback Session\_End; hard to resolve in StateServer or SQLServer mode), etc.

In the next major release ASP.NET will increase the support for customizing session state module. Stay tuned.

### **Q: How does (de)serialization work in SqlServer and State Server mode?**

A: SqlServer and State Server modes use serialization to store objects in a remote store. Understanding how it works will help you write your application correctly.

Assume your page foo.aspx has this piece of code:

```
line 1  MyClass class = (MyClass)Session["abc"];
line 2  if (class == null) {
line 3      class = new MyClass();
line 4      Session["abc"] = class;
line 5  }
line 6
line 7  class.LastAccess = DateTime.Now; // This change will reflect in Session["abc"]
line 8
line 9  int total;
line 10 if (Session["def"] == null) {
line 11     total = 0;
```

```
line 12 }  
line 13  
line 14 total = total + 1;  
line 15 Session["def"] = total;  
line 16 total = total + 100;    // This change will NOT reflect in Session["def"]
```

First, let me explain one area which confuses many people.

Assume the page hits line 1 and it gets null. Then the page will create the object (line 3), and put it in session (line 4). Please note that what Session has is just a reference to your object (and actually the local var 'class' is also just a reference to your object). So when you modify the object in line 7, you are actually modifying the same object pointed to by Session["abc"]. Bottom line is that 'Session["abc"]' and 'class' are just two references that point to the same object.

However, the situation in line 9-15 is different. Line 16 has no effect on Session["def"]. It's because 'total' is a value type, and so 'Session["def"]' and 'total' are two integers which are independent of each other.

Please note that the above difference has nothing to do with session state, but with the fundamental difference between a value type (e.g. int), and an object type (e.g. MyClass). You will have the same behavior if you replace 'Session' by, for example, a hashtable object.

Now, back to the actual question. When do deserialization and serialization happen?

Assume you hit the above page with a brand new empty session. After the HttpHandler is done executing your page, ASP.NET will later fire the HttpApplication.ReleaseRequestState event. At that point, the session state module will go thru all the items in session state, serialize them all into a single binary blob, and save it to a remote store.

Now, when you hit the page again, before HttpHandler execute your page, ASP.NET will fire the HttpApplication.AcquireRequestState event. At that point, the session state module will read the saved binary blob from the remote store, deserialize it into individual items, and stick them into Session. But once again, what it puts into Session["abc"] is just a reference to an object (of type MyClass) which was created during the deserialization.

#### **Q: How can I secure my state server?**

A: If the state server is running on the same machine as web server, run state server in local only mode by setting the DWORD registry value HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\aspnet\_state\Parameters\AllowRemoteConnection to 0. This will prevent remote clients from connecting to the State Server. This feature is available in v1.1, and in the to-be-shipped v1 SP3.

The state server must be protected by a firewall from external connections to truly guarantee security. The default port is TCP 42424, although it can be changed by setting HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\aspnet\_state\Parameters\Port. Block all external connections on this port except 127.0.0.1 if in local mode, or explicitly disallow all addresses other than the connecting web server if in remote mode.

Using IPSec is another way of protecting your state server.

#### **Q: Can I subscribe to SessionStateModule.End event using a non-global.asax handler method?**

A: The answer is NO. When SessionStateModule raises the End event, only the method defined in global.asax will be called.

It's limited this way for a technical safety reason. Suppose asp.net allows a user to use any arbitrary handler to the handle the End event. In the case, users will usually use a page method as a handler. When you pass in that handler during event subscription, that handler (unless it's static) is associated with the HttpApplication instance your request is running on. Please note that HttpApplication

instances are recycled to serve other requests. So later on, when the End event gets fired, asp.net will call the handler, and the HttpApplication instance it is associated with is now being used by another request. Such a situation will cause all sorts of problem. So to avoid this danger, in v1 a decision was made to call only the method defined in global.asax. Hope you all can bear with this limitation.

**Q: Can different apps store their session state in different databases on the same SQL server?**

A: The answer is yes. See this KB for more details.

<http://support.microsoft.com/default.aspx?scid=kb;EN-US;836680>

Nagaraj.NET