**ASP.NET Session State Overview**

**.NET Framework 4**

[Other Versions](javascript:;)

Description: http://i.msdn.microsoft.com/Areas/Epx/Content/Images/ImageSprite.png

* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/ms178581(d=printer,v=vs.90).aspx)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/ms178581(d=printer,v=vs.85).aspx)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/ms178581(d=printer,v=vs.80).aspx)

Use ASP.NET session state to store and retrieve values for a user.

This topic contains:

* [Background](http://msdn.microsoft.com/en-us/library/ms178581(d=printer).aspx#Background)
* [Code Examples](http://msdn.microsoft.com/en-us/library/ms178581(d=printer).aspx#CodeExamples)
* [Class Reference](http://msdn.microsoft.com/en-us/library/ms178581(d=printer).aspx#ClassReference)

A Visual Studio project with source code is available to accompany this topic: [Download](http://go.microsoft.com/fwlink/?LinkId=191883).

[Background](javascript:void(0))

ASP.NET session state enables you to store and retrieve values for a user as the user navigates ASP.NET pages in a Web application. HTTP is a stateless protocol. This means that a Web server treats each HTTP request for a page as an independent request. The server retains no knowledge of variable values that were used during previous requests. ASP.NET session state identifies requests from the same browser during a limited time window as a session, and provides a way to persist variable values for the duration of that session. By default, ASP.NET session state is enabled for all ASP.NET applications.

Alternatives to session state include the following:

* Application state, which stores variables that can be accessed by all users of an ASP.NET application.
* Profile properties, which persists user values in a data store without expiring them.
* ASP.NET caching, which stores values in memory that is available to all ASP.NET applications.
* View state, which persists values in a page.
* Cookies.
* The query string and fields on an HTML form that are available from an HTTP request.

For a comparison of different state-management options, see [ASP.NET State Management Recommendations](http://msdn.microsoft.com/en-us/library/z1hkazw7.aspx).

**Session Variables**

Session variables are stored in a [SessionStateItemCollection](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstateitemcollection.aspx) object that is exposed through the [HttpContext.Session](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.session.aspx) property. In an ASP.NET page, the current session variables are exposed through the Session property of the Page object.

The collection of session variables is indexed by the name of the variable or by an integer index. Session variables are created by referring to the session variable by name. You do not have to declare a session variable or explicitly add it to the collection. The following example shows how to create session variables in an ASP.NET page for the first and last name of a user, and set them to values retrieved from [TextBox](http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.textbox.aspx) controls.

C#

[VB](http://msdn.microsoft.com/en-us/library/ms178581(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

Session["FirstName"] = FirstNameTextBox.Text;

Session["LastName"] = LastNameTextBox.Text;

Session variables can be any valid .NET Framework type. The following example stores an [ArrayList](http://msdn.microsoft.com/en-us/library/system.collections.arraylist.aspx) object in a session variable named StockPicks. The value returned by the StockPicks session variable must be cast to the appropriate type when you retrieve it from the [SessionStateItemCollection](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstateitemcollection.aspx).

C#

[VB](http://msdn.microsoft.com/en-us/library/ms178581(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

// When retrieving an object from session state, cast it to

// the appropriate type.

ArrayList stockPicks = (ArrayList)Session["StockPicks"];

// Write the modified stock picks list back to session state.

Session["StockPicks"] = stockPicks;

|  |
| --- |
| **Description: NoteNote** |
| When you use a session-state mode other than [InProc](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemode.aspx), the session-variable type must be either a primitive .NET type or serializable. This is because the session-variable value is stored in an external data store. For more information, see [Session-State Modes](http://msdn.microsoft.com/en-us/library/ms178586.aspx). |

**Session Identifiers**

Sessions are identified by a unique identifier that can be read by using the [SessionID](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid.aspx) property. When session state is enabled for an ASP.NET application, each request for a page in the application is examined for a [SessionID](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid.aspx) value sent from the browser. If no [SessionID](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid.aspx) value is supplied, ASP.NET starts a new session and the [SessionID](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid.aspx) value for that session is sent to the browser with the response.

By default, [SessionID](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid.aspx) values are stored in a cookie. However, you can also configure the application to store [SessionID](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid.aspx) values in the URL for a "cookieless" session.A session is considered active as long as requests continue to be made with the same [SessionID](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid.aspx) value. If the time between requests for a particular session exceeds the specified time-out value in minutes, the session is considered expired. Requests made with an expired [SessionID](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid.aspx) value result in a new session.

|  |
| --- |
| **Description: Security noteSecurity Note** |
| [SessionID](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid.aspx) values are sent in clear text, whether as a cookie or as part of the URL. A malicious user could get access to the session of another user by obtaining the [SessionID](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid.aspx) value and including it in requests to the server. If you are storing sensitive information in session state, it is recommended that you use SSL to encrypt any communication between the browser and server that includes the [SessionID](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid.aspx) value. |

**Cookieless SessionIDs**

By default, the [SessionID](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid.aspx) value is stored in a non-expiring session cookie in the browser. However, you can specify that session identifiers should not be stored in a cookie by setting the cookieless attribute to true in the [sessionState](http://msdn.microsoft.com/en-us/library/h6bb9cz9.aspx) section of the Web.config file.The following example shows a Web.config file that configures an ASP.NET application to use cookieless session identifiers.

<configuration>

<system.web>

<sessionState cookieless="true"

regenerateExpiredSessionId="true" />

</system.web>

</configuration>

ASP.NET maintains cookieless session state by automatically inserting a unique session ID into the page's URL. For example, the following URL has been modified by ASP.NET to include the unique session ID lit3py55t21z5v55vlm25s55:

http://www.example.com/(S(lit3py55t21z5v55vlm25s55))/orderform.aspx

When ASP.NET sends a page to the browser, it modifies any links in the page that use an application-relative path by embedding a session ID value in the links. (Links with absolute paths are not modified.) Session state is maintained as long as the user clicks links that have been modified in this manner. However, if the client rewrites a URL that is supplied by the application, ASP.NET may not be able to resolve the session ID and associate the request with an existing session. In that case, a new session is started for the request.

The session ID is embedded in the URL after the slash that follows the application name and before any remaining file or virtual directory identifier. This enables ASP.NET to resolve the application name before involving the [SessionStateModule](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.aspx) in the request.

|  |
| --- |
| **Description: NoteNote** |
| To improve the security of your application, you should allow users to log out of your application, at which point the application should call the [Abandon](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstatecontainer.abandon.aspx) method. This reduces the potential for a malicious user to get the unique identifier in the URL and use it to retrieve private user data stored in the session. |

**Regenerating Expired Session Identifiers**

By default, the session ID values that are used in cookieless sessions are recycled. That is, if a request is made with a session ID that has expired, a new session is started by using the [SessionID](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid.aspx) value that is supplied with the request. This can result in a session unintentionally being shared when a link that contains a cookieless [SessionID](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid.aspx) value is used by multiple browsers. (This can occur if the link is passed through a search engine, through an e-mail message, or through another program.) You can reduce the chance of session data being shared by configuring the application not to recycle session identifiers. To do this, set the regenerateExpiredSessionId attribute of the [sessionState](http://msdn.microsoft.com/en-us/library/h6bb9cz9.aspx) configuration element to true. This generates a new session ID when a cookieless session request is made with an expired session ID.

|  |
| --- |
| **Description: NoteNote** |
| If the request that is made with the expired session ID is made by using the HTTP POST method, any posted data will be lost when regenerateExpiredSessionId is true. This is because ASP.NET performs a redirect to make sure that the browser has the new session identifier in the URL. |

**Custom Session Identifiers**

You can implement a custom class to supply and validate [SessionID](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid.aspx) values. To do so, create a class that inherits the [SessionIDManager](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionidmanager.aspx) class and override the [CreateSessionID](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionidmanager.createsessionid.aspx) and [Validate](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionidmanager.validate.aspx) methods with your own implementations. For an example, see the example provided for the [CreateSessionID](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionidmanager.createsessionid.aspx) method.

You can replace the [SessionIDManager](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionidmanager.aspx) class by creating a class that implements the [ISessionIDManager](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.isessionidmanager.aspx) interface. For example, you might have a Web application that associates a unique identifier with non-ASP.NET pages (such as HTML pages or images) by using an ISAPI filter. You can implement a custom [SessionIDManager](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionidmanager.aspx) class to use this unique identifier with ASP.NET session state. If your custom class supports cookieless session identifiers, you must implement a solution for sending and retrieving session identifiers in the URL.

**Session Modes**

ASP.NET session state supports several storage options for session variables. Each option is identified as a session-state [Mode](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.mode.aspx) type. The default behavior is to store session variables in the memory space of the ASP.NET worker process. However, you can also specify that session state should be stored in a separate process, in a SQL Server database, or in a custom data source. If you do not want session state enabled for your application, you can set the session mode to [Off](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemode.aspx).

For more information, see [Session-State Modes](http://msdn.microsoft.com/en-us/library/ms178586.aspx).

**Session Events**

ASP.NET provides two events that help you manage user sessions. The **Session\_OnStart** event is raised when a new session starts, and the **Session\_OnEnd** event is raised when a session is abandoned or expires. Session events are specified in the Global.asax file for an ASP.NET application.

The **Session\_OnEnd** event is not supported if the session [Mode](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.mode.aspx) property is set to a value other than [InProc](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemode.aspx), which is the default mode.

|  |
| --- |
| **Description: NoteNote** |
| If the Global.asax file or Web.config file for an ASP.NET application is modified, the application will be restarted and any values stored in application state or session state will be lost. Be aware that some anti-virus software can update the last-modified date and time of the Global.asax or Web.config file for an application. |

For more information, see [Session-State Events](http://msdn.microsoft.com/en-us/library/ms178583.aspx).

**Configuring Session State**

Session state is configured by using the [sessionState](http://msdn.microsoft.com/en-us/library/h6bb9cz9.aspx) element of the system.web configuration section. You can also configure session state by using the [EnableSessionState](http://msdn.microsoft.com/en-us/library/system.web.configuration.pagessection.enablesessionstate.aspx) value in the @ Page directive.

The sessionState element enables you to specify the following options:

* The mode in which the session will store data.
* The way in which session identifier values are sent between the client and the server.
* The session [Timeout](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.timeout.aspx) value.
* Supporting values that are based on the session [Mode](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.mode.aspx) setting.

The following example shows a sessionState element that configures an application for [SQLServer](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemode.aspx) session mode. It sets the [Timeout](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.timeout.aspx) value to 30 minutes, and specifies that session identifiers are stored in the URL.

<sessionState mode="SQLServer"

cookieless="true "

regenerateExpiredSessionId="true "

timeout="30"

sqlConnectionString="Data Source=MySqlServer;Integrated Security=SSPI;"

stateNetworkTimeout="30"/>

You can disable session state for an application by setting the session-state mode to [Off](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemode.aspx). If you want to disable session state for only a particular page of an application, you can set the [EnableSessionState](http://msdn.microsoft.com/en-us/library/system.web.configuration.pagessection.enablesessionstate.aspx) value in the @ Page directive to false. The [EnableSessionState](http://msdn.microsoft.com/en-us/library/system.web.configuration.pagessection.enablesessionstate.aspx) value can also be set to ReadOnly to provide read-only access to session variables.

**Concurrent Requests and Session State**

Access to ASP.NET session state is exclusive per session, which means that if two different users make concurrent requests, access to each separate session is granted concurrently. However, if two concurrent requests are made for the same session (by using the same [SessionID](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid.aspx) value), the first request gets exclusive access to the session information. The second request executes only after the first request is finished. (The second session can also get access if the exclusive lock on the information is freed because the first request exceeds the lock time-out.) If the [EnableSessionState](http://msdn.microsoft.com/en-us/library/system.web.configuration.pagessection.enablesessionstate.aspx) value in the @ Page directive is set to ReadOnly, a request for the read-only session information does not result in an exclusive lock on the session data. However, read-only requests for session data might still have to wait for a lock set by a read-write request for session data to clear.

[Code Examples](javascript:void(0))

[How to: Save Values in Session State](http://msdn.microsoft.com/en-us/library/6ad7zeeb.aspx)

[How to: Read Values from Session State](http://msdn.microsoft.com/en-us/library/03sekbw5.aspx)

[Implementing a Session-State Store Provider](http://msdn.microsoft.com/en-us/library/ms178587.aspx)

[Class Reference](javascript:void(0))

The following table lists key classes that relate to session state are in the [System.Web.SessionState](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.aspx) namespace.

|  |  |
| --- | --- |
| **Member** | **Description** |
| [SessionIDManager](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionidmanager.aspx) | Manages unique identifiers for ASP.NET session state. |
| [SessionStateItemCollection](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstateitemcollection.aspx) | Used to store session state variables. |

**Session-State Modes**

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-US/library/ms178586(d=printer,v=vs.90).aspx)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-US/library/ms178586(d=printer,v=vs.85).aspx)
* [Visual Studio 2005](http://msdn.microsoft.com/en-US/library/ms178586(d=printer,v=vs.80).aspx)

ASP.NET session state supports several different storage options for session data. Each option is identified by a value in the [SessionStateMode](http://msdn.microsoft.com/en-US/library/system.web.sessionstate.sessionstatemode.aspx) enumeration. The following list describes the available session state modes:

* InProc mode, which stores session state in memory on the Web server. This is the default.
* StateServer mode, which 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.
* SQLServer mode stores session state in a SQL Server database. 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.
* Custom mode, which enables you to specify a custom storage provider.
* Off mode, which disables session state.

You can specify which mode you want ASP.NET session state to use by assigning a [SessionStateMode](http://msdn.microsoft.com/en-US/library/system.web.sessionstate.sessionstatemode.aspx) enumeration values to the mode attribute of the [sessionState](http://msdn.microsoft.com/en-US/library/h6bb9cz9.aspx) element in your application's Web.config file. Modes other than InProc and Off require additional parameters, such as connection-string values as discussed later in this topic. You can view the currently selected session state by accessing the value of the [HttpSessionState.Mode](http://msdn.microsoft.com/en-US/library/system.web.sessionstate.httpsessionstate.mode.aspx) property.

[In-Process Mode](javascript:void(0))

In-process mode is the default session state mode and is specified using the InProc [SessionStateMode](http://msdn.microsoft.com/en-US/library/system.web.sessionstate.sessionstatemode.aspx) enumeration value. In-process mode stores session state values and variables in memory on the local Web server. It is the only mode that supports the **Session\_OnEnd** event. For more information about the Session\_OnEnd event, see [Session-State Events](http://msdn.microsoft.com/en-US/library/ms178583.aspx).

|  |
| --- |
| **Caution noteCaution** |
| If you enable Web-garden mode by setting the webGarden attribute to true in the [processModel](http://msdn.microsoft.com/en-US/library/7w2sway1.aspx) element of the application's Web.config file, do not use InProc session state mode. If you do, data loss can occur if different requests for the same session are served by different worker processes. |

[State Server Mode](javascript:void(0))

StateServer mode stores session state in a process, referred to as the ASP.NET state service, that is separate from the ASP.NET worker process or IIS application pool. Using this mode 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.

To use StateServer mode, you must first be sure the ASP.NET state service is running on the server used for the session store. The ASP.NET state service is installed as a service when ASP.NET and the .NET Framework are installed. The ASP.Net state service is installed at the following location: systemroot\Microsoft.NET\Framework\versionNumber\aspnet\_state.exe

To configure an ASP.NET application to use StateServer mode, in the application's Web.config file do the following:

* Set the mode attribute of the [sessionState](http://msdn.microsoft.com/en-US/library/h6bb9cz9.aspx) element to StateServer.
* Set the stateConnectionString attribute to tcpip=serverName:42424.

|  |
| --- |
| **NoteNote** |
| To improve the security of your application when using StateServer mode, it is recommended that you protect your stateConnectionString value by encrypting the [sessionState](http://msdn.microsoft.com/en-US/library/h6bb9cz9.aspx) section of your configuration file. For details, see [Encrypting Configuration Information Using Protected Configuration](http://msdn.microsoft.com/en-US/library/53tyfkaw.aspx). |

The following example shows a configuration setting for StateServer mode where session state is stored on a remote computer named SampleStateServer:

<configuration>

<system.web>

<sessionState mode="StateServer"

stateConnectionString="tcpip=SampleStateServer:42424"

cookieless="false"

timeout="20"/>

</system.web>

</configuration>

|  |
| --- |
| **NoteNote** |
| Objects stored in session state must be serializable if the mode is set to StateServer. For information on serializable objects, see the [SerializableAttribute](http://msdn.microsoft.com/en-US/library/system.serializableattribute.aspx) class. |

To use StateServer mode in a Web farm, you must have the same encryption keys specified in the [machineKey](http://msdn.microsoft.com/en-US/library/w8h3skw9.aspx) element of your Web configuration for all applications that are part of the Web farm. For information on how to create machine keys, see article 313091, "How to create keys by using Visual Basic .NET for use in Forms authentication," in the Microsoft Knowledge Base at http://support.microsoft.com.

[SQL Server Mode](javascript:void(0))

SQLServer mode stores session state in a SQL Server database. Using this mode 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.

|  |
| --- |
| **NoteNote** |
| Objects stored in session state must be serializable if the mode is SQL Server. For information on serializable objects, see the [SerializableAttribute](http://msdn.microsoft.com/en-US/library/system.serializableattribute.aspx) class. |

To use SQLServer mode, you must first be sure the ASP.NET session state database is installed on SQL Server. You can install the ASP.NET session state database using the Aspnet\_regsql.exe tool, as described later in this topic.

To configure an ASP.NET application to use SQLServer mode, do the following in the application's Web.config file:

* Set the mode attribute of the [sessionState](http://msdn.microsoft.com/en-US/library/h6bb9cz9.aspx) element to SQLServer.
* Set the sqlConnectionString attribute to a connection string for your SQL Server database.

|  |
| --- |
| **NoteNote** |
| To improve the security of your application when using SQLServer mode, it is recommended that you protect your sqlConnectionString value by encrypting the [sessionState](http://msdn.microsoft.com/en-US/library/h6bb9cz9.aspx) section of your configuration file. For details, see [Encrypting Configuration Information Using Protected Configuration](http://msdn.microsoft.com/en-US/library/53tyfkaw.aspx). |

The following example shows a configuration setting for SQLServer mode where session state is stored on a SQL Server named "SampleSqlServer":

<configuration>

<system.web>

<sessionState mode="SQLServer"

sqlConnectionString="Integrated Security=SSPI;data

source=SampleSqlServer;" />

</system.web>

</configuration>

|  |
| --- |
| **NoteNote** |
| If you specify a trusted connection to your SQL Server in the configuration file using the [sessionState](http://msdn.microsoft.com/en-US/library/h6bb9cz9.aspx) element's sqlConnectionString attribute, the [SessionStateModule](http://msdn.microsoft.com/en-US/library/system.web.sessionstate.sessionstatemodule.aspx) will connect to SQL Server using SQL Server integrated security. The connection will be made using the ASP.NET process identity or the user credentials supplied for the [identity](http://msdn.microsoft.com/en-US/library/72wdk8cc.aspx) configuration element, if they exist. You can specify that the IIS impersonated identity be used instead by specifying <identity impersonate="true" /> and setting the useHostingIdentity attribute of the [sessionState](http://msdn.microsoft.com/en-US/library/h6bb9cz9.aspx) configuration element to false. For more information on the ASP.NET process identity, see [Configuring ASP.NET Process Identity](http://msdn.microsoft.com/en-US/library/dwc1xthy.aspx) and [ASP.NET Impersonation](http://msdn.microsoft.com/en-US/library/xh507fc5.aspx). |

To configure SQLServer mode for a Web farm, in the configuration file for each Web server, set the [sessionState](http://msdn.microsoft.com/en-US/library/h6bb9cz9.aspx) element's sqlConnectionString attribute to point to the same SQL Server database. The path for the ASP.NET application in the IIS metabase must be identical on all Web servers that share session state in the SQL Server database. For information on steps to resolve the issue when application paths differ between servers, see article 325056, "PRB: Session State Is Lost in Web Farm If You Use SqlServer or StateServer Session Mode," in the Microsoft Knowledge Base at http://support.microsoft.com.

**Installing the Session State Database Using the Aspnet\_regsql.exe Tool**

To install the session state database on SQL Server, run the Aspnet\_regsql.exe tool located in the systemroot\Microsoft.NET\Framework\versionNumber folder on your Web server. Supply the following information with the command:

* Thename of the SQL Server instance, using the **-S** option.
* The logon credentials for an account that has permission to create a database on SQL Server. Use the **-E** option to use the currently logged-on user, or use the **-U** option to specify a user ID along with the **-P** option to specify a password.
* The **-ssadd** command-line option to add the session state database.

By default, you cannot use the Aspnet\_regsql.exe tool to install the session state database on SQL Server Express. In order to run the Aspnet\_regsql.exe tool to install a SQL Server Express database, you must first enable the Agent XPs SQL Server option using Transact-SQL commands like the following:

EXECUTE sp\_configure 'show advanced options', 1

RECONFIGURE WITH OVERRIDE

GO

EXECUTE sp\_configure 'Agent XPs', 1

RECONFIGURE WITH OVERRIDE

GO

EXECUTE sp\_configure 'show advanced options', 0

RECONFIGURE WITH OVERRIDE

GO

You must run these Transact-SQL commands for any instance of SQL Server Express where the Agent XPs option is disabled.

By default, the Aspnet\_regsql.exe tool will create a database named ASPState containing stored procedures that support SQLServer mode. Session data itself is stored in the tempdb database by default. You can optionally use the -sstype option to change the storage location of session data. The following table specifies the possible values for the -sstype option:

|  |  |
| --- | --- |
| **Option** | **Description** |
| T | Stores session data in the SQL Server tempdb database. This is the default. If you store session data in the tempdb database, the session data is lost if SQL Server is restarted. |
| P | Stores session data in the ASPState database instead of in the tempdb database. |
| C | Stores session data in a custom database. If you specify the c option, you must also include the name of the custom database using the -d option. |

For example, the following command creates a database named ASPState on a SQL Server instance named "SampleSqlServer" and specifies that session data is also stored in the ASPState database:

**aspnet\_regsql.exe -S SampleSqlServer -E -ssadd -sstype p**

|  |
| --- |
| **NoteNote** |
| If you are running ASP.NET 1.0 or ASP.NET 1.1, you cannot use the Aspnet\_regsql.exe tool to configure ASP.NET to store session state in a persistent SQL Server database. However, you can obtain scripts to store session state in a persistent database. For details, see article 311209, "HOW TO: Configure ASP.NET for Persistent SQL Server Session State Management" in the Microsoft Knowledge Base at http://support.microsoft.com. As an alternative, Web servers running ASP.NET 1.0 or ASP.NET 1.1 can direct persistent session state to a SQL Server that has the ASP.NET 2.0 session state schema installed. |

In SQLServer mode, you can configure several computers running SQL Server to work as a failover cluster, which is two or more identical computers running SQL Server that store data for a single database. If one computer running SQL Server fails, another server in the cluster can take over and serve requests without session-data loss. To configure SQL Server mode for a failover cluster, you must specify -sstype p when you execute the Aspnet\_regsql.exe tool so that session state data is stored in the ASPState database instead of the tempdb database. Storing session state in the tempdb database is not supported for a SQL Server cluster. For more information about setting up SQL Server mode for a failover cluster, see article 323262, "How to use ASP.NET session state SQL Server Mode in a failover cluster" in the Microsoft Knowledge Base at http://support.microsoft.com.

[Custom Mode](javascript:void(0))

Custom mode specifies that you want to store session state data using a custom session state store provider. When you configure your ASP.NET application with a [Mode](http://msdn.microsoft.com/en-US/library/system.web.sessionstate.httpsessionstate.mode.aspx) of Custom, you must specify the type of the session state store provider using the providers sub-element of the [sessionState](http://msdn.microsoft.com/en-US/library/h6bb9cz9.aspx) configuration element. You specify the provider type using an add sub-element and include both a type attribute that specifies the provider's type name and a name attribute that specifies the provider instance name. The name of the provider instance is then supplied to the customProvider attribute of the [sessionState](http://msdn.microsoft.com/en-US/library/h6bb9cz9.aspx) element to configure ASP.NET session state to use that provider instance for storing and retrieving session data.

The following example shows elements from a Web.config file that specify that ASP.NET session state use a custom session state store provider:

<configuration>

<connectionStrings>

<add name="OdbcSessionServices"

connectionString="DSN=SessionState;" />

</connectionStrings>

<system.web>

<sessionState

mode="Custom"

customProvider="OdbcSessionProvider">

<providers>

<add name="OdbcSessionProvider"

type="Samples.AspNet.Session.OdbcSessionStateStore"

connectionStringName="OdbcSessionServices"

writeExceptionsToEventLog="false" />

</providers>

</sessionState>

</system.web>

</configuration>

For more information on custom session state store providers, see [Implementing a Session-State Store Provider](http://msdn.microsoft.com/en-US/library/ms178587.aspx).

|  |
| --- |
| **NoteNote** |
| A custom session state store provider will access any secured resource, such as SQL Server, using the ASP.NET process identity or the user credentials supplied to the [identity](http://msdn.microsoft.com/en-US/library/72wdk8cc.aspx) configuration element, if they exist. You can specify that the IIS impersonated identity be used instead by specifying <identity impersonate="true" /> and setting the useHostingIdentity attribute of the [sessionState](http://msdn.microsoft.com/en-US/library/h6bb9cz9.aspx) configuration element to false. For more information on the ASP.NET process identity, see [Configuring ASP.NET Process Identity](http://msdn.microsoft.com/en-US/library/dwc1xthy.aspx) and [ASP.NET Impersonation](http://msdn.microsoft.com/en-US/library/xh507fc5.aspx). |

**HttpSessionState.SessionID Property**

**.NET Framework 4.5**

[Other Versions](javascript:;)



* [.NET Framework 4](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid(d=printer,v=vs.100).aspx)
* [.NET Framework 3.5](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid(d=printer,v=vs.90).aspx)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid(d=printer,v=vs.85).aspx)
* [.NET Framework 2.0](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid(d=printer,v=vs.80).aspx)
* [.NET Framework 1.1](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid(d=printer,v=vs.71).aspx)

Gets the unique identifier for the session.

**Namespace:**  [System.Web.SessionState](http://msdn.microsoft.com/en-us/library/system.web.sessionstate(v=vs.110).aspx)  
**Assembly:**  System.Web (in System.Web.dll)

[Syntax](javascript:void(0))

C#

[C++](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid(d=printer,v=vs.110).aspx?cs-save-lang=1&cs-lang=cpp#code-snippet-1)

[F#](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid(d=printer,v=vs.110).aspx?cs-save-lang=1&cs-lang=fsharp#code-snippet-1)

[VB](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid(d=printer,v=vs.110).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

public string SessionID { get; }

**Property Value**

Type: [System.String](http://msdn.microsoft.com/en-us/library/system.string(v=vs.110).aspx)  
The unique session identifier.

[Remarks](javascript:void(0))

The SessionID property is used 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.

If you want to disable the use of cookies in your ASP.NET application and still make use of session state, you can configure your application to store the session identifier in the URL instead of a cookie by setting the cookieless attribute of the [sessionState](http://msdn.microsoft.com/en-us/library/h6bb9cz9(v=vs.110).aspx) configuration element to true, or to [UseUri](http://msdn.microsoft.com/en-us/library/system.web.httpcookiemode(v=vs.110).aspx), in the Web.config file for your application. You can have ASP.NET determine whether cookies are supported by the browser by specifying a value of [UseDeviceProfile](http://msdn.microsoft.com/en-us/library/system.web.httpcookiemode(v=vs.110).aspx) for the cookieless attribute. You can also have ASP.NET determine whether cookies are enabled for the browser by specifying a value of [AutoDetect](http://msdn.microsoft.com/en-us/library/system.web.httpcookiemode(v=vs.110).aspx) for the cookieless attribute. If cookies are supported when [UseDeviceProfile](http://msdn.microsoft.com/en-us/library/system.web.httpcookiemode(v=vs.110).aspx) is specified, or enabled when [AutoDetect](http://msdn.microsoft.com/en-us/library/system.web.httpcookiemode(v=vs.110).aspx) is specified, then the session identifier will be stored in a cookie; otherwise the session identifier will be stored in the URL. For more information, see the [IsCookieless](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.iscookieless(v=vs.110).aspx) property.

The SessionID is sent between the server and the browser in clear text, either in a cookie or in the URL. As a result, an unwanted source could gain access to the session of another user by obtaining the SessionID value and including it in requests to the server. If you are storing private or sensitive information in session state, it is recommended that you use SSL to encrypt any communication between the browser and server that includes the SessionID.

When using cookie-based session state, ASP.NET does not allocate storage for session data until the [Session](http://msdn.microsoft.com/en-us/library/system.web.ui.page.session(v=vs.110).aspx) object is used. As a result, a new session ID is generated for each page request until the session object is accessed. If your application requires a static session ID for the entire session, you can either implement the Session\_Start method in the application's Global.asax file and store data in the [Session](http://msdn.microsoft.com/en-us/library/system.web.ui.page.session(v=vs.110).aspx) object to fix the session ID, or you can use code in another part of your application to explicitly store data in the [Session](http://msdn.microsoft.com/en-us/library/system.web.ui.page.session(v=vs.110).aspx) object.

If your application uses cookieless session state, the session ID is generated on the first page view and is maintained for the entire session.

[Examples](javascript:void(0))

The following code example shows a Web.config file that configures session state to use cookieless session identifiers. For more information, see the [IsCookieless](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.iscookieless(v=vs.110).aspx) property.

<configuration>

<system.web>

<sessionState

cookieless="true"

regenerateExpiredSessionId="true"

timeout="30" />

</system.web>

</configuration>

**How to: Save Values in Session State**

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/6ad7zeeb(d=printer,v=vs.90).aspx)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/6ad7zeeb(d=printer,v=vs.85).aspx)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/6ad7zeeb(d=printer,v=vs.80).aspx)

This example uses the [HttpSessionState](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.aspx) object to persist values within an individual session.

[Example](javascript:void(0))

C#

[VB](http://msdn.microsoft.com/en-us/library/6ad7zeeb(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

string firstName = "Jeff";

string lastName = "Smith";

string city = "Seattle";

Session["FirstName"] = firstName;

Session["LastName"] = lastName;

Session["City"] = city;

[Compiling the Code](javascript:void(0))

This example requires:

* A Web Forms page or class that has access to the current request context using the [Current](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.current.aspx) property in an ASP.NET application that has session state enabled.

[Robust Programming](javascript:void(0))

Session state can expire (by default, after 20 minutes of inactivity), and the information that you store there can be lost. You can control session-state lifetime using the timeout attribute of the [sessionState](http://msdn.microsoft.com/en-us/library/h6bb9cz9.aspx) configuration section.

Depending on your application requirements, you may want to consider an alternative to session state for storing information for each user. ASP.NET provides several other options for persisting data within an application. For a comparison of each, see [ASP.NET State Management Recommendations](http://msdn.microsoft.com/en-us/library/z1hkazw7.aspx).

**How to: Read Values from Session State**

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/03sekbw5(d=printer,v=vs.90).aspx)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/03sekbw5(d=printer,v=vs.85).aspx)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/03sekbw5(d=printer,v=vs.80).aspx)

This example accesses the [Item](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.item.aspx) property to retrieve the values in session state.

[Example](javascript:void(0))

C#

[VB](http://msdn.microsoft.com/en-us/library/03sekbw5(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-1)

string firstName = (string)(Session["First"]);

string lastName = (string)(Session["Last"]);

string city = (string)(Session["City"]);

[Compiling the Code](javascript:void(0))

This example requires:

* A Web Forms page or class that has access to the current request context using the [Current](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.current.aspx) property in an ASP.NET application that has session state enabled.

[Robust Programming](javascript:void(0))

No exception is thrown if you attempt to get a value out of session state that does not exist. To be sure that the value you want is in session state, check first for the existence of the object with a test such as the following:

C#

[VB](http://msdn.microsoft.com/en-us/library/03sekbw5(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-2)

if (Session["City"] == null)

// No such value in session state; take appropriate action.

If you attempt to use a nonexistent session state entry in some other way (for example, to examine its type), a [NullReferenceException](http://msdn.microsoft.com/en-us/library/system.nullreferenceexception.aspx) exception is thrown.

Session values are of type [Object](http://msdn.microsoft.com/en-us/library/system.object.aspx). In Visual Basic, if you set Option Strict On, you must cast from type [Object](http://msdn.microsoft.com/en-us/library/system.object.aspx) to the appropriate type when getting values out of session state, as shown in the example. In C#, you should always cast to the appropriate type when reading session values.

**Implementing a Session-State Store Provider**

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/ms178587(d=printer,v=vs.90).aspx)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/ms178587(d=printer,v=vs.85).aspx)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/ms178587(d=printer,v=vs.80).aspx)

Describes a custom session-state store provider implementation and demonstrates implementing a sample provider.

ASP.NET session state is designed to enable you to store user session data in different sources. By default, session state values and information are stored in memory within the ASP.NET process. One alternative is to store session data in a state server, which keeps session data in a separate process and retains it if the ASP.NET application is shut down and restarted. Another alternative is to store session data in a SQL Server database, where it can be shared by multiple Web servers.

You can use the session-state stores that are included with ASP.NET, or you can implement your own session-state store provider. You might create a custom session-state store provider for the following reasons:

* You need to store session-state information in a data source other than SQL Server, such as a FoxPro database or an Oracle database.
* You need to manage session-state information using a database schema that is different from the database schema used by the providers that ship with the .NET Framework. An example of this would be shopping cart data that is stored with a predefined schema in your existing SQL Server database.

You can implement a custom session-state store provider by creating a class that inherits the [SessionStateStoreProviderBase](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.aspx) class. For more information, see the "Required Classes" section later in this topic.

[The Session State Module](javascript:void(0))

Session state is managed by the [SessionStateModule](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.aspx) class, which calls the session-state store provider to read and write session data to the data store at different times during a request. At the beginning of a request, the [SessionStateModule](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.aspx) instance retrieves data from the data source by calling the [GetItemExclusive](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.getitemexclusive.aspx) method, or if the [EnableSessionState](http://msdn.microsoft.com/en-us/library/system.web.configuration.pagessection.enablesessionstate.aspx) page attribute has been set to ReadOnly, by calling the [GetItem](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.getitem.aspx) method. At the end of a request, if the session-state values have been modified, the [SessionStateModule](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.aspx) instance calls the [SessionStateStoreProviderBase.SetAndReleaseItemExclusive](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.setandreleaseitemexclusive.aspx) method to write the updated values to the session-state store. [SessionStateModule](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.aspx) calls additional members of the [SessionStateStoreProviderBase](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.aspx) implementation to initialize a new session as well as to delete session data from the data store when the [HttpSessionState.Abandon](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.abandon.aspx) method is called. Each member of the [SessionStateStoreProviderBase](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.aspx) class is discussed in more detail in the "Required Classes" section later in this topic.

The [SessionStateModule](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.aspx) class determines the [SessionID](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid.aspx) value itself, rather than relying on the session-state store provider to do so. If needed, you can implement a custom [SessionIDManager](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionidmanager.aspx) by creating a class that inherits the [ISessionIDManager](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.isessionidmanager.aspx) interface. For more information, see the "Remarks" section in [ISessionIDManager](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.isessionidmanager.aspx).

[SessionStateModule](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.aspx) will revert to the ASP.NET process identity to access any secured resource, such as a database server. You can specify that the [SessionStateModule](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.aspx) instance impersonate the identity supplied by IIS by setting the useHostingIdentity attribute of the [<sessionState>](http://msdn.microsoft.com/en-us/library/h6bb9cz9.aspx) configuration element to false. For example, if you have configured your IIS application to use Windows Integrated security and you want ASP.NET to impersonate the identity provided by IIS for session management, specify <identity impersonate="true" /> in the [<system.web>](http://msdn.microsoft.com/en-us/library/dayb112d.aspx) configuration section of the Web.config file for the application, and set the useHostingIdentity attribute of the [<sessionState>](http://msdn.microsoft.com/en-us/library/h6bb9cz9.aspx) configuration element to false. If the useHostingIdentity attribute is true, ASP.NET will impersonate the process identity, or the user credentials supplied to the [<identity>](http://msdn.microsoft.com/en-us/library/72wdk8cc.aspx) configuration element (if they exist) when connecting to the data source. For more information on the ASP.NET process identity, see [Configuring ASP.NET Process Identity](http://msdn.microsoft.com/en-us/library/dwc1xthy.aspx) and [ASP.NET Impersonation](http://msdn.microsoft.com/en-us/library/xh507fc5.aspx).

[Locking Session-Store Data](javascript:void(0))

ASP.NET applications are multithreaded so they can respond to multiple concurrent requests. Multiple concurrent requests might attempt to access the same session information. Consider a scenario where multiple frames in a frameset all reference ASP.NET Web pages in the same application. The separate requests for each frame in the frameset might be executed on the Web server concurrently on different threads. If the ASP.NET pages for each frame access session-state variables, you could have multiple threads accessing the session store concurrently. To avoid data collisions at the session store and unexpected session-state behavior, the [SessionStateModule](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.aspx) and [SessionStateStoreProviderBase](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.aspx) classes include functionality that exclusively locks the session-store item for a particular session during the execution of an ASP.NET page. Note that no lock is set on a session-store item if the [EnableSessionState](http://msdn.microsoft.com/en-us/library/system.web.configuration.pagessection.enablesessionstate.aspx) attribute is marked as ReadOnly. However, other ASP.NET pages in the same application might be able to write to the session store, so a request for read-only session data from the store might still have to wait for locked data to be freed.

A lock is set on session-store data at the beginning of the request in the call to the [GetItemExclusive](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.getitemexclusive.aspx) method. When the request completes, the lock is released during the call to the [SetAndReleaseItemExclusive](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.setandreleaseitemexclusive.aspx) method.

If the [SessionStateModule](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.aspx) instance encounters locked session data during the call to either the [GetItemExclusive](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.getitemexclusive.aspx) or [GetItem](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.getitem.aspx) method, it will re-request the session data at half-second intervals until either the lock is released or the amount of time specified in the [ExecutionTimeout](http://msdn.microsoft.com/en-us/library/system.web.configuration.httpruntimesection.executiontimeout.aspx) property has elapsed. If the request times out, [SessionStateModule](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.aspx) calls the [ReleaseItemExclusive](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.releaseitemexclusive.aspx) method to free the session-store data and request the session-store data at that time.

Locked session-store data might have been freed by a call to the [ReleaseItemExclusive](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.releaseitemexclusive.aspx) method on a separate thread, before the call to the [SetAndReleaseItemExclusive](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.setandreleaseitemexclusive.aspx) method for the current response. This could cause the [SessionStateModule](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.aspx) instance to set and release session-state store data that has already been released and modified by another session. To avoid this situation, [SessionStateModule](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.aspx) includes a lock identifier with each request to modify locked session-store data. Session-store data is only modified if the lock identifier in the data store matches the lock identifier supplied by [SessionStateModule](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.aspx).

[Deleting Expired Session-Store Data](javascript:void(0))

When the [Abandon](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.abandon.aspx) method is called for a session, the data for that session is deleted from the data store using the [RemoveItem](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.removeitem.aspx) method. Otherwise, the data remains in the session data store to serve future requests for the session.

The mechanism for deleting expired session data depends on the capabilities of your data source. If your data source can be configured to delete expired session data according to the session [Timeout](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.timeout.aspx) property, you can use the [SetItemExpireCallback](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.setitemexpirecallback.aspx) method to reference the delegate for the Session\_OnEnd event and raise it when deleting expired session data.

[ApplicationName](javascript:void(0))

To maintain session scope, session-state providers store session information uniquely for each application. This allows multiple ASP.NET applications to use the same data source without running into a conflict if duplicate session identifiers are encountered.

Because session-state store providers store session information uniquely for each application, you must ensure that your data schema, queries, and updates include the application name. For example, the following command might be used to retrieve session data from a database.

SELECT \* FROM Sessions

WHERE SessionID = 'ABC123' AND ApplicationName = 'MyApplication'

Alternatively, you can store a combination of the session identifier and the application name as the unique identifier for an item in the session-state data store.

[Required Classes](javascript:void(0))

To implement a session-state store provider, create a class that inherits the [SessionStateStoreProviderBase](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.aspx) abstract class. The [SessionStateStoreProviderBase](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.aspx) class in turn inherits the [ProviderBase](http://msdn.microsoft.com/en-us/library/system.configuration.provider.providerbase.aspx) abstract class, so you must implement the required members of the [ProviderBase](http://msdn.microsoft.com/en-us/library/system.configuration.provider.providerbase.aspx) class as well. The following tables list the properties and methods that you must implement from the [ProviderBase](http://msdn.microsoft.com/en-us/library/system.configuration.provider.providerbase.aspx) and [SessionStateStoreProviderBase](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.aspx) abstract classes and provides a description of each. To view an implementation of each member, see [Sample Session-State Store Provider](http://msdn.microsoft.com/en-us/library/ms178588.aspx).

**Required ProviderBase Members**

|  |  |
| --- | --- |
| **Member** | **Description** |
| [Initialize](http://msdn.microsoft.com/en-us/library/system.configuration.provider.providerbase.initialize.aspx) method | Takes as input the name of the provider and a [NameValueCollection](http://msdn.microsoft.com/en-us/library/system.collections.specialized.namevaluecollection.aspx) instance of configuration settings. This method is used to set property values for the provider instance, including implementation-specific values and options specified in the configuration file (Machine.config or Web.config). |

**Required SessionStateStoreProvider Members**

|  |  |
| --- | --- |
| **Member** | **Description** |
| [InitializeRequest](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.initializerequest.aspx) method | Takes as input the [HttpContext](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.aspx) instance for the current request and performs any initialization required by your session-state store provider. |
| [EndRequest](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.endrequest.aspx) method | Takes as input the [HttpContext](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.aspx) instance for the current request and performs any cleanup required by your session-state store provider. |
| [Dispose](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.dispose.aspx) method | Frees any resources no longer in use by the session-state store provider. |
| [GetItemExclusive](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.getitemexclusive.aspx) method | Takes as input the [HttpContext](http://msdn.microsoft.com/en-us/library/system.web.httpcontext.aspx) instance for the current request and the [SessionID](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.httpsessionstate.sessionid.aspx) value for the current request. Retrieves session values and information from the session data store and locks the session-item data at the data store for the duration of the request. The [GetItemExclusive](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.getitemexclusive.aspx) method sets several output-parameter values that inform the calling [SessionStateModule](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.aspx) about the state of the current session-state item in the data store.  If no session item data is found at the data store, the [GetItemExclusive](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.getitemexclusive.aspx) method sets the locked output parameter to false and returns null. This causes [SessionStateModule](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.aspx) to call the [CreateNewStoreData](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.createnewstoredata.aspx) method to create a new [SessionStateStoreData](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoredata.aspx) object for the request.  If session-item data is found at the data store but the data is locked, the [GetItemExclusive](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.getitemexclusive.aspx) method sets the locked output parameter to true, sets the lockAge output parameter to the current date and time minus the date and time when the item was locked, sets the lockId output parameter to the lock identifier retrieved from the data store, and returns null. This causes [SessionStateModule](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.aspx) to call the [GetItemExclusive](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.getitemexclusive.aspx) method again after a half-second interval, to attempt to retrieve the session-item information and obtain a lock on the data. If the value that the lockAge output parameter is set to exceeds the [ExecutionTimeout](http://msdn.microsoft.com/en-us/library/system.web.configuration.httpruntimesection.executiontimeout.aspx) value, [SessionStateModule](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.aspx) calls the [ReleaseItemExclusive](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.releaseitemexclusive.aspx) method to clear the lock on the session-item data and then call the [GetItemExclusive](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.getitemexclusive.aspx) method again.  The actionFlags parameter is used with sessions whose [Cookieless](http://msdn.microsoft.com/en-us/library/system.web.configuration.sessionstatesection.cookieless.aspx) property is true, when the regenerateExpiredSessionId attribute is set to true. An actionFlags value set to [InitializeItem](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstateactions.aspx) (1) indicates that the entry in the session data store is a new session that requires initialization. Uninitialized entries in the session data store are created by a call to the [CreateUninitializedItem](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.createuninitializeditem.aspx) method. If the item from the session data store is already initialized, the actionFlags parameter is set to zero.  If your provider supports cookieless sessions, set the actionFlags output parameter to the value returned from the session data store for the current item. If the actionFlags parameter value |

**Sample Session-State Store Provider**

**.NET Framework 4**

[Other Versions](javascript:;)



* [Visual Studio 2008](http://msdn.microsoft.com/en-us/library/ms178588(d=printer,v=vs.90).aspx)
* [.NET Framework 3.0](http://msdn.microsoft.com/en-us/library/ms178588(d=printer,v=vs.85).aspx)
* [Visual Studio 2005](http://msdn.microsoft.com/en-us/library/ms178588(d=printer,v=vs.80).aspx)

Describes a custom session-state store-provider implementation that uses the ODBC .NET Framework data provider to manage session information in an Access database.

The following topics include the code for a sample session-state store-provider implementation. The sample provider uses the [System.Data.Odbc](http://msdn.microsoft.com/en-us/library/system.data.odbc.aspx) classes to store and retrieve session information by using an Access database.

This topic describes implementation details about the sample session-state store provider and describes how to build the sample and configure an ASP.NET application to use the sample provider.

The code for the sample provider can be found in the [How to: Sample Session-State Store Provider](http://msdn.microsoft.com/en-us/library/ms178589.aspx) topic.

[Database Schema](javascript:void(0))

The sample session-state provider uses a single table named Sessions to manage session information. To create the Access table used by the sample provider, issue the following data-definition query in a new or existing Access database.

CREATE TABLE Sessions

(

SessionId Text(80) NOT NULL,

ApplicationName Text(255) NOT NULL,

Created DateTime NOT NULL,

Expires DateTime NOT NULL,

LockDate DateTime NOT NULL,

LockId Integer NOT NULL,

Timeout Integer NOT NULL,

Locked YesNo NOT NULL,

SessionItems Memo,

Flags Integer NOT NULL,

CONSTRAINT PKSessions PRIMARY KEY (SessionId, ApplicationName)

)

[Event Log Access](javascript:void(0))

If the sample provider encounters an exception when working with the data source, it writes the details of the exception to the Application Event Log instead of returning the exception to the ASP.NET application. This is done as a security measure to avoid private information about the data source from being exposed in the ASP.NET application.

The sample provider specifies an event [Source](http://msdn.microsoft.com/en-us/library/system.diagnostics.eventlog.source.aspx) property value of "OdbcSessionStateStore." Before your ASP.NET application will be able to write to the Application Event Log successfully, you will need to create the following registry key:

HKEY\_LOCAL\_MACHINE\SYSTEM\CurrentControlSet\Services\Eventlog\Application\OdbcSessionStateStore

If you do not want the sample provider to write exceptions to the event log, then you can set the custom writeExceptionsToEventLog attribute to false in the Web.config file.

[Support for the Session\_OnEnd Event](javascript:void(0))

The sample session-state store provider does not support the Session\_OnEnd event defined in the Global.asax file, as there is no way for the Access database to notify the session-state store provider that the expiration date and time for a session has passed. The session-state store provider must query for this information. You cannot predict when the session-state store provider will be used, and it is therefore unlikely that the Session\_OnEnd event will be raised at the exact time that the session times out. As a result, the [SetItemExpireCallback](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatestoreproviderbase.setitemexpirecallback.aspx) method implementation in the sample session-state store provider returns false, to inform the [SessionStateModule](http://msdn.microsoft.com/en-us/library/system.web.sessionstate.sessionstatemodule.aspx) that the Session\_OnEnd event is not supported.

[Cleaning Up Expired Session Data](javascript:void(0))

Because the sample session-state store provider does not provide support for the Session\_OnEnd event, it does not automatically clean up expired session-item data. It is recommended that you periodically delete expired session information from the data store with the following code.

C#

[VB](http://msdn.microsoft.com/en-us/library/ms178588(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-3)

string commandString = "DELETE FROM Sessions WHERE Expires < ?";

OdbcConnection conn = new OdbcConnection(connectionString);

OdbcCommand cmd = new OdbcCommand(commandString, conn);

cmd.Parameters.Add("@Expires", OdbcType.DateTime).Value = DateTime.Now;

conn.Open();

cmd.ExecuteNonQuery();

conn.Close();

[Building the Sample Provider](javascript:void(0))

In order to use the sample provider, you can place your source code in the App\_Code directory of your application. Note that if you already have source code in the App\_Code directory of your application, you must add the version of the sample provider that is written in the same language as the existing code in the directory. The provider will be compiled by ASP.NET when your application is requested.

You can also compile the sample provider as a library and place it in the Bin directory of your Web application, or strongly name it and place it in the GAC. The following command shows how to compile the sample provider using the command-line compiler after the sample code has been copied to a file named OdbcSessionStateStore.vb for Visual Basic and OdbcSessionStateStore.cs for C#.

C#

[VB](http://msdn.microsoft.com/en-us/library/ms178588(d=printer).aspx?cs-save-lang=1&cs-lang=vb#code-snippet-4)

csc /out:OdbcSessionStateStore.dll /t:library OdbcSessionStateStore.cs /r:System.Web.dll /r:System.Configuration.dll

[Using the Sample Provider in an ASP.NET Application](javascript:void(0))

The following example shows the Web.config file for an ASP.NET application configured to use the sample provider. The example uses an ODBC DSN named "SessionState" to obtain connection information for the Access database. To use the sample provider, you will need to either create the "SessionState" System DSN or supply a valid ODBC connection string to your database.

The example configuration assumes that your Web site is set up to use forms authentication and includes an ASP.NET page called login.aspx that allows users to log in.

<configuration>

<connectionStrings>

<add name="OdbcSessionServices" connectionString="DSN=SessionState;" />

</connectionStrings>

<system.web>

<sessionState

cookieless="true"

regenerateExpiredSessionId="true"

mode="Custom"

customProvider="OdbcSessionProvider">

<providers>

<add name="OdbcSessionProvider"

type="Samples.AspNet.Session.OdbcSessionStateStore"

connectionStringName="OdbcSessionServices"

writeExceptionsToEventLog="false" />

</providers>

</sessionState>

</system.web>

</configuration>

# Cookieless ASP.NET

Dino Esposito  
Wintellect

May 2005

**Summary:** Dino reviews the pros and cons of cookieless sessions and discusses why you should avoid storing valuable information in the session state. (6 printed pages)

#### Contents

[Are Cookies a Problem?](http://msdn.microsoft.com/en-us/library/aa479314(d=printer).aspx#cookieless_topic1)  
[Enter Cookieless Sessions](http://msdn.microsoft.com/en-us/library/aa479314(d=printer).aspx#cookieless_topic2)  
[Implementation](http://msdn.microsoft.com/en-us/library/aa479314(d=printer).aspx#cookieless_topic3)   
[Thumbs Up](http://msdn.microsoft.com/en-us/library/aa479314(d=printer).aspx#cookieless_topic4)  
[Thumbs Down](http://msdn.microsoft.com/en-us/library/aa479314(d=printer).aspx#cookieless_topic5)  
[Summary](http://msdn.microsoft.com/en-us/library/aa479314(d=printer).aspx#cookieless_topic6)

Let's admit it—we're so accustomed to the idea of session state that we forget that session state is an artifice introduced with Active Server Pages (ASP) back in 1997. Session state gives you as the developer the ability to persist a piece of information about a user for the duration of time the user interacts with your application. The user-specific information is generally preserved for a 20-minute period that is renewed each time the user comes back to the site.

The first time the user connects to the site a brand new session state is created in the form of a block of memory to hold data, plus an ID to uniquely link it to the current user. On the next request, the user is expected to present the session ID so that the session state can be retrieved and properly restored. The session ID is an alphanumeric string that ASP and ASP.NET generate in total autonomy. How can the user manage it and make sure it is wrapped up with each subsequent request?

The HTTP protocol is stateless in nature, and nobody has done anything to change this fact. Almost two decades ago, while developing their first browser, Netscape Corporation "invented" a persistence mechanism to work over HTTP. They called it an HTTP cookie. It is interesting to note that the term "cookie" in computer science jargon just indicates an opaque piece of data held by an application that affects users but is never directly managed by users.

So cookies store the ID of the session and browsers transparently move their contents back and forth between the Web server and the local user's machine. When a cookie-enabled browser receives a response packet, it looks for attached cookies and stores their content to a text file in a particular folder in the local Windows directory. The cookie also contains information about the site of origin. Next, when the browser sends a request to the site, it looks in the cookies folder for a cookie that originated from that domain. If found, the cookie is automatically attached to the outgoing packet. The cookie hits the server application where it is detected, extracted, and processed.

In the end, cookies make Web sites much easier to navigate because they provide the illusion of continuity on top of a user's experience that necessarily spans over multiple requests.

## Are Cookies a Problem?

For several years, cookies were simply considered a technical feature and largely ignored. A few years ago, the worldwide push on Web security focused the spotlight on cookies. Cookies were alleged to contain dangerous programs capable of stealing valuable information even beyond the physical boundaries of the machine.

It goes without saying that cookies are not programs and therefore can't collect any information on their own, let alone any personal information about users. More simply, a cookie is a piece of text that a Web site can park on a user's machine to be retrieved and reused later. The information stored consists of harmless name-value pairs.

The point is, cookies are not part of the standard HTTP specification, so they imply a collaboration between browsers and Web sites to work. Not all browsers support cookies and, more importantly, not all users may have cookie support enabled in their own copy of the browser.

There are Web site features that are historically so tightly bound to cookies that they make it hard to distinguish which really came first. On the one hand, session state management and user authentication are much easier to code with cookies. On the other hand, if you take a look at your site's statistics regarding browsers used to access pages, you might be surprised to discover that a significant share of users connect with cookies disabled. This poses a point for you as a developer.

Summarizing, cookies are not a problem per se but their use undoubtedly gives some server code the ability to store a piece of data on client machines. This prefigures some potential security risks and an overall situation less then ideal. (In some cases and countries, it's even illegal for an application to require cookies to work.)

## Enter Cookieless Sessions

In ASP.NET, the necessary session-to-user link may optionally be established without using cookies. Interestingly enough, you don't have to change anything in your ASP.NET application to enable cookieless sessions, except the following configuration setting.

<sessionState cookieless="true" />

Default settings for ASP.NET session state are defined in the machine.config file and can be overridden in the web.config file in the application's root folder. By ensuring that the above line appears in the root web.config file, you enable cookieless sessions. That's it—easy and effective.

The **<sessionState>** node can also be used to configure other aspects of the session state management, including the storage medium and the connection strings. However, as far as cookies are concerned, setting the **cookieless** attribute to true (default is false) is all that you have to do.

Note that session settings are application-wide settings. In other words, all the pages in your site will, or will not, use cookies to store session IDs.

Where is ASP.NET storing the session ID when cookies are not being used? In this case, the session ID is inserted in a particular position within the URL. The figure below shows a snapshot from a real-world site that uses cookieless sessions.



**Figure 1. MapPoint using cookieless sessions**

Imagine you request a page like http://yourserver/folder/default.aspx. As you can see from the MapPoint screenshot, the slash immediately preceding the resource name is expanded to include parentheses with the session ID stuffed inside, as below.

http://yourserver/folder/(**session ID here**)/default.aspx

The session ID is embedded in the URL and there's no need to persist it anywhere. Well, not exactly.. Consider the following scenario.

You visit a page and get assigned a session ID. Next, you clear the address bar of the same browser instance, go to another application and work. Then, you retype the URL of the previous application and, guess what, retrieve your session values as you get in.

If you use cookieless sessions, in your second access to the application you're assigned a different session ID and lose all of your previous state. This is a typical side effect of cookieless sessions. To understand why, let's delve deep into the implementation of cookieless sessions.

## Implementation

The implementation of cookieless sessions results from the efforts of two runtime modules—the standard Session HTTP module named **SessionStateModule** and an executable known as aspnet\_filter.dll. The latter is a small piece of Win32 code acting as an ISAPI filter. HTTP modules and ISAPI filters realize the same idea, except that HTTP modules are made of managed code and require ASP.NET and CLR to trigger and work. Classic ISAPI filters like aspnet\_filter.dll are invoked by Internet Information Services (IIS). Both intercept IIS events fired during the processing of the request.

When the first request of a new browser session comes in, the session state module reads about the cookie support in the web.config file. If the **cookieless** attribute of the **<sessionState>** section is set to true, the module generates a new session ID, mangles the URL by stuffing the session ID just before the resource name, and redirects the browser to the new URL using the HTTP 302 command.

When each request arrives at the IIS gate—far before it is handed over to ASP.NET—aspnet\_filter.dll is given a chance to look at it. If the URL embeds a session ID in parentheses, then the session ID is extracted and copied into a request header named **AspFilterSessionId**. The URL is then rewritten to look like the originally requested resource and let go. This time the ASP.NET session state module retrieves the session ID from the request header and proceeds with session-state binding.

The cookieless mechanism works great as long as the URL contains information that can be used to obtain the session ID. As you'll see in a moment, this poses some usage restrictions.

Let's review the pros and cons of cookieless sessions.

## Thumbs Up

In ASP.NET, session management and forms authentication are the only two system features that use cookies under the hood. With cookieless sessions, you can now deploy stateful applications that work regardless of the user's preferences about cookies. As of ASP.NET 1.x, though, cookies are still required to implement forms authentication. The good news is that in ASP.NET 2.0 forms authentication can optionally work in a cookieless fashion.

Another common reason advanced against cookies is security. This is a point that deserves a bit more attention.

Cookies are inert text files and as such can be replaced or poisoned by hackers, should they gain access to a machine. The real threat lies not much in what cookies can install on your client machine, but in what they can upload to the target site. Cookies are not programs and never run like programs; other software that gets installed on your machine, though, can use the built-in browser support for cookies to do bad things remotely.

Furthermore, cookies are at risk of theft. Once stolen, a cookie that contains valuable and personal information can disclose its contents to malicious hackers and favor other types of Web attacks. In summary, by using cookies you expose yourself to risks that can be zeroed off otherwise. Really?

## Thumbs Down

Let's look at security from another perspective. Have you ever heard of session hijacking? If not, take a look at the TechNet Magazine article [Theft On The Web: Prevent Session Hijacking](http://www.microsoft.com/technet/technetmag/issues/2005/01/SessionHijacking/default.aspx). In brief, session hijacking occurs when an attacker gains access to the session state of a particular user. Basically, the attacker steals a valid session ID and uses that to get into the system and snoop into the data. One common way to get a valid session ID is stealing a valid session cookie. That said, if you think that cookieless sessions put your application on the safe side, you're deadly wrong. With cookieless sessions, in fact, the session ID shows up right in the address bar! Try the following:

1. Connect to a Web site that uses cookieless sessions—for example, MapPoint—and get a map. At this point, the address is stored in the session state.
2. Grab the URL up to the page name. Don't include the query string but make sure the URL includes the session ID.
3. Save the URL to a file and copy/send the file to another machine.
4. On the second machine, open the file and paste the URL in a new instance of the browser.
5. The same map shows up as long as the session timeout is still valid.

With cookieless sessions, stealing session IDs is easier than ever.

I believe we all agree that stealing sessions is a reprehensible action from an ethical point of view. But is it injurious as well? That depends on what is actually stored in the session state. Stealing a session ID per se doesn't execute an action out of the code control. But it could disclose private data to unauthorized users and enable the bad guy to execute unauthorized operations. Read [Wicked Code: Foiling Session Hijacking Attempts](http://msdn.microsoft.com/msdnmag/issues/04/08/WickedCode/default.aspx) for tips on how to block session hijacking in ASP.NET applications. (And, yes, it doesn't rely on cookieless sessions!)

Using cookieless sessions also raises issues with links. For example, you can't have absolute, fully qualified links in your ASP.NET pages. If you do this, each request that originates from that hyperlink will be considered as part of a new session. Cookieless sessions require that you always use relative URLs, like in ASP.NET postbacks. You can use a fully qualified URL only if you can embed the session ID in it. But how can you do that, since session IDs are generated at run time?

The following code breaks the session:

<a runat="server" href="/test/page.aspx">Click</a>

To use absolute URLs, resort to a little trick that uses the **ApplyAppPathModifier** method on the **HttpResponse** class:

<a runat="server"

href=<% =Response.ApplyAppPathModifier("/test/page.aspx")%> >Click</a>

The **ApplyAppPathModifier** method takes a string representing a URL and returns an absolute URL that embeds session information. For example, this trick is especially useful in situations in which you need to redirect from a HTTP page to an HTTPS page. Finally, be conscious that every time you type a path to a site from within the same browser you're going to lose your state with cookieless sessions. As a further warning, be aware that cookieless sessions can be problematic with mobile applications if the device can't handle the specially formatted URL.

## Summary

The main reason for cookieless sessions in ASP.NET is that users—for whatever reasons—may have cookies disabled on their browsers. Like it or not, this is a situation you have to face if your application requires session state. Cookieless sessions embed the session ID in the URL and obtain a two-fold result. On the one hand, they provide a way for the Web site to correctly identify the user making the request. On the other hand, though, they make the session ID clearly visible to potential hackers who can easily steal it and represent themselves as you.

To implement cookieless sessions you don't have to modify your programming model—a simple change in the **web.config** file does the trick—but refactoring your application to avoid storing valuable information in the session state is strongly recommended too. At the same time, reducing the lifetime of a session to less than the default 20 minutes can help in keeping your users and your site safe.