

ASP.NET State Management Recommendations

.NET Framework 4

State management is the process by which you maintain state and page information over multiple requests for the same or different pages. As is true for any HTTP-based technology, Web Forms pages are stateless, which means that they do not automatically indicate whether the requests in a sequence are all from the same client or even whether a single browser instance is still actively viewing a page or site. Furthermore, pages are destroyed and re-created with each round trip to the server; therefore, page information will not exist beyond the life cycle of a single page. For more information about server round trips and the life cycle of Web Forms pages, see [ASP.NET Page Life Cycle Overview](#).

ASP.NET provides multiple ways to maintain state between server round trips. Which of these options you choose depends heavily upon your application, and it should be based on the following criteria:

- How much information do you need to store?
- Does the client accept persistent or in-memory cookies?
- Do you want to store the information on the client or on the server?
- Is the information sensitive?
- What performance and bandwidth criteria do you have for your application?
- What are the capabilities of the browsers and devices that you are targeting?
- Do you need to store information per user?
- How long do you need to store the information?
- Do you have a Web farm (multiple servers), a Web garden (multiple processes on one machine), or a single process that serves the application?

Client-Side State Management Options

Storing page information using client-side options doesn't use server resources. These options typically have minimal security but fast server performance because the demand on server resources is modest. However, because you must send information to the client for it to be stored, there is a practical limit on how much information you can store this way.

The following are the client-side state management options that ASP.NET supports:

- View state
- Control state
- Hidden fields
- Cookies
- Query strings

View State

Web Forms pages provide the [ViewState](#) property as a built-in structure for automatically retaining values between multiple requests for the same page. View state is maintained as a hidden field in the page. For more information, see [ASP.NET State Management Overview](#).

You can use view state to store your own page-specific values across round trips when the page posts back to itself. For example, if your application is maintaining user-specific information — that is, information that is used in the page but is not necessarily part of any control — you can store it in view state.

Advantages of using view state are:

- **No server resources are required** The view state is contained in a structure within the page code.
- **Simple implementation** View state does not require any custom programming to use. It is on by default to maintain state data on controls.
- **Enhanced security features** The values in view state are hashed, compressed, and encoded for Unicode implementations, which provides more security than using hidden fields.

Disadvantages of using view state are:

- **Performance considerations** Because the view state is stored in the page itself, storing large values can cause the page to slow down when users display it and when they post it. This is especially relevant for mobile devices, where bandwidth is often a limitation.

- **Device limitations** Mobile devices might not have the memory capacity to store a large amount of view-state data.
- **Potential security risks** The view state is stored in one or more hidden fields on the page. Although view state stores data in a hashed format, it can still be tampered with. The information in the hidden field can be seen if the page output source is viewed directly, creating a potential security issue. For more information, see [ASP.NET Web Application Security](#) and [Basic Security Practices for Web Applications](#).

For more information about using view state, see [View State Overview](#).

Control State

The ASP.NET page framework provides the [ControlState](#) property as a way to store custom control data between server trips. For example, if you have written a custom control that has different tabs showing different information, in order for that control to work as expected, the control needs to know which tab is selected between round trips. View state can be used for this purpose, but developers can turn view state off at the page level, effectively breaking your control. Unlike view state, control state cannot be turned off, so it provides a more reliable way to store control-state data.

Advantages of using control state are:

- **No server resources are required** By default, control state is stored in hidden fields on the page.
- **Reliability** Because control state cannot be turned off like view state, control state is a more reliable method for managing the state of controls.
- **Versatility** Custom adapters can be written to control how and where control-state data is stored.

Disadvantage of using control state are:

- **Some programming is required** While the ASP.NET page framework provides a foundation for control state, control state is a custom state-persistence mechanism. To fully utilize control state, you must write code to save and load control state.

Hidden Fields

You can store page-specific information in a hidden field on your page as a way of maintaining the state of your page.

If you use hidden fields, it is best to store only small amounts of frequently changed data on the client.

Note

If you use hidden fields, you must submit your pages to the server using the HTTP POST method rather than requesting the page via the page URL (the HTTP GET method).

Advantages of using hidden fields are:

- **No server resources are required** The hidden field is stored and read from the page.
- **Widespread support** Almost all browsers and client devices support forms with hidden fields.
- **Simple implementation** Hidden fields are standard HTML controls that require no complex programming logic.

Disadvantages of using hidden fields are:

- **Potential security risks** The hidden field can be tampered with. The information in the hidden field can be seen if the page output source is viewed directly, creating a potential security issue. You can manually encrypt and decrypt the contents of a hidden field, but doing so requires extra coding and overhead. If security is a concern, consider using a server-based state mechanism so that no sensitive information is sent to the client. For more information, see [ASP.NET Web Application Security](#) and [Basic Security Practices for Web Applications](#).
- **Simple storage architecture** The hidden field does not support rich data types. Hidden fields offer a single string value field in which to place information. To store multiple values, you must implement delimited strings and the code to parse those strings. You can manually serialize and de-serialize rich data types to and from hidden fields, respectively. However, it requires extra code to do so. If you need to store rich data types on the client, consider using view state instead. View state has serialization built-in, and it stores data in hidden fields.
- **Performance considerations** Because hidden fields are stored in the page itself, storing large values can cause the page to slow down when users display it and when they post it.
- **Storage limitations** If the amount of data in a hidden field becomes very large, some proxies and firewalls will prevent access to the page that contains them. Because the maximum amount can vary with different firewall and proxy implementations, large hidden fields can be sporadically problematic. If you need to store many items of data, consider doing one of the following:
 - Put each item in a separate hidden field.
 - Use view state with view-state chunking turned on, which automatically separates data into multiple hidden fields.
 - Instead of storing data on the client, persist the data on the server. The more data you send to the client, the slower the apparent response time of your application will be because the browser will need to download or send more data.

Cookies

Cookies are useful for storing small amounts of frequently changed information on the client. The information is sent with the request to the server. For details about creating and reading cookies, see [ASPNET Cookies Overview](#).

Advantages of using cookies are:

- **Configurable expiration rules** The cookie can expire when the browser session ends, or it can exist indefinitely on the client computer, subject to the expiration rules on the client.
- **No server resources are required** The cookie is stored on the client and read by the server after a post.
- **Simplicity** The cookie is a lightweight, text-based structure with simple key-value pairs.
- **Data persistence** Although the durability of the cookie on a client computer is subject to cookie expiration processes on the client and user intervention, cookies are generally the most durable form of data persistence on the client.

Disadvantages of using cookies are:

- **Size limitations** Most browsers place a 4096-byte limit on the size of a cookie, although support for 8192-byte cookies is becoming more common in newer browser and client-device versions.
- **User-configured refusal** Some users disable their browser or client device's ability to receive cookies, thereby limiting this functionality.
- **Potential security risks** Cookies are subject to tampering. Users can manipulate cookies on their computer, which can potentially cause a security risk or cause the application that is dependent on the cookie to fail. Also, although cookies are only accessible by the domain that sent them to the client, hackers have historically found ways to access cookies from other domains on a user's computer. You can manually encrypt and decrypt cookies, but it requires extra coding and can affect application performance because of the time that is required for encryption and decryption. For more information, see [ASPNET Web Application Security](#) and [Basic Security Practices for Web Applications](#).

Note

Cookies are often used for personalization, where content is customized for a known user. In most of these cases, identification is the issue rather than authentication. Thus, you can typically secure a cookie that is used for identification by storing the user name, account name, or a unique user ID (such as a GUID) in the cookie and then using the cookie to access the user personalization infrastructure of a site.

Query Strings

A query string is information that is appended to the end of a page URL. For more information, see [ASPNET State Management Overview](#).

You can use a query string to submit data back to your page or to another page through the URL. Query strings provide a simple but limited way of maintaining some state information. For example, query strings are an easy way to pass information from one page to another, such as passing a product number to another page where it will be processed.

Advantages of using query strings are:

- **No server resources are required** The query string is contained in the HTTP request for a specific URL.
- **Widespread support** Almost all browsers and client devices support using query strings to pass values.
- **Simple implementation** ASPNET provides full support for the query-string method, including methods of reading query strings using the [Params](#) property of the [HttpRequest](#) object.



Disadvantages of using query strings are:

- **Potential security risks** The information in the query string is directly visible to the user via the browser's user interface. A user can bookmark the URL or send the URL to other users, thereby passing the information in the query string along with it. If you are concerned about any sensitive data in the query string, consider using hidden fields in a form that uses POST instead of using query strings. For more information, see [ASPNET Web Application Security](#) and [Basic Security Practices for Web Applications](#).
- **Limited capacity** Some browsers and client devices impose a 2083-character limit on the length of URLs.

Client-Side Method State Management Summary

The following table lists the client-side state management options that are available with ASPNET, and provides recommendations about when you should use each option.

State management option	Recommended usage
View state	Use when you need to store small amounts of information for a page that will post back to itself. Using the ViewState property provides functionality with basic security.

Control state	Use when you need to store small amounts of state information for a control between round trips to the server.
Hidden fields	Use when you need to store small amounts of information for a page that will post back to itself or to another page, and when security is not an issue.  Note You can use a hidden field only on pages that are submitted to the server.
Cookies	Use when you need to store small amounts of information on the client and security is not an issue.
Query string	Use when you are transferring small amounts of information from one page to another and security is not an issue.  Note You can use query strings only if you are requesting the same page, or another page via a link.

Server-Side State Management Options

Server-side options for storing page information typically have higher security than client-side options, but they can use more Web server resources, which can lead to scalability issues when the size of the information store is large. ASP.NET provides several options to implement server-side state management. For more information, see [ASP.NET State Management Overview](#).

The following are the server-side state management options that ASP.NET supports:

- Application state
- Session state
- Profile properties
- Database support

Application State

ASP.NET provides application state via the [HttpApplicationState](#) class as a method of storing global application-specific information that is visible to the entire application. Application-state variables are, in effect, global variables for an ASP.NET application. For more information, see [ASP.NET Application State Overview](#).

You can store your application-specific values in application state, which is then managed by the server. For more information, see [ASP.NET State Management Overview](#).

Data that is shared by multiple sessions and does not change often is the ideal type of data to insert into application-state variables.

Advantages of using application state are:

- **Simple implementation** Application state is easy to use, familiar to ASP developers, and consistent with other .NET Framework classes.
- **Application scope** Because application state is accessible to all pages in an application, storing information in application state can mean keeping only a single copy of the information (for instance, as opposed to keeping copies of information in session state or in individual pages).

Disadvantages of using application state are:

- **Application scope** The scope of application state can also be a disadvantage. Variables stored in application state are global only to the particular process the application is running in, and each application process can have different values. Therefore, you cannot rely on application state to store unique values or update global counters in Web-garden and Web-farm server configurations.
- **Limited durability of data** Because global data that is stored in application state is volatile, it will be lost if the Web server process containing it is destroyed, such as from a server crash, upgrade, or shutdown.
- **Resource requirements** Application state requires server memory, which can affect the performance of the server as well as the scalability of the application.

Careful design and implementation of application state can increase Web application performance. For example, placing a commonly used, relatively static dataset in application state can increase site performance by reducing the overall number of data requests to a database. However, there is a performance trade-off. Application state variables containing large blocks of information reduce Web server performance as server load increases. The memory occupied by a variable stored in application state is not released until the value is either removed or replaced. Therefore, it is best to use application-state variables only with small, infrequently changed datasets. For more information, see [ASP.NET Performance Overview](#).

Session State

ASP.NET provides a session state, which is available as the [HttpSessionState](#) class, as a method of storing session-specific information that is visible only within the session. ASP.NET session state identifies requests from the same browser during a limited time window as a session, and provides the ability to persist variable values for the duration of that session. For more information, see [ASP.NET State Management Overview](#) and [ASP.NET Session State Overview](#).

You can store your session-specific values and objects in session state, which is then managed by the server and available to the browser or client device. The ideal data to store in session-state variables is short-lived, sensitive data that is specific to an individual session.

Advantages of using session state are:

- **Simple implementation** The session-state facility is easy to use, familiar to ASP developers, and consistent with other .NET Framework classes.
- **Session-specific events** Session management events can be raised and used by your application.
- **Data persistence** Data placed in session-state variables can be preserved through Internet Information Services (IIS) restarts and worker-process restarts without losing session data because the data is stored in another process space. Additionally, session-state data can be persisted across multiple processes, such as in a Web farm or a Web garden.
- **Platform scalability** Session state can be used in both multi-computer and multi-process configurations, therefore optimizing scalability scenarios.
- **Cookieless support** Session state works with browsers that do not support HTTP cookies, although session state is most commonly used with cookies to provide user identification facilities to a Web application. Using session state without cookies, however, requires that the session identifier be placed in the query string, which is subject to the security issues stated in the query string section of this topic. For more information about using session state without cookies, see [ASP.NET Web Site Administration](#).
- **Extensibility** You can customize and extend session state by writing your own session-state provider. Session state data can then be stored in a custom data format in a variety of data storage mechanisms, such as a database, an XML file, or even to a Web service. For more information, see [Implementing a Session-State Store Provider](#).

Disadvantage of using session state are:

- **Performance considerations** Session-state variables stay in memory until they are either removed or replaced, and therefore can degrade server performance. Session-state variables that contain blocks of information, such as large datasets, can adversely affect Web-server performance as server load increases.

Profile Properties

ASP.NET provides a feature called profile properties, which allows you to store user-specific data. It is similar to session state, except that unlike session state, the profile data is not lost when a user's session expires. The profile properties feature uses an ASP.NET profile, which is stored in a persistent format and associated with an individual user. The ASP.NET profile allows you to easily manage user information without requiring you to create and maintain your own database. In addition, the profile makes the user information available using a strongly typed API that you can access from anywhere in your application. You can store objects of any type in the profile. The ASP.NET profile feature provides a generic storage system that allows you to define and maintain almost any kind of data while still making the data available in a type-safe manner. For more information, see [ASP.NET Profile Properties Overview](#).

Advantages of using profile properties are:

- **Data persistence** Data placed in profile properties is preserved through IIS restarts and worker-process restarts without losing data because the data is stored in an external mechanism. Additionally, profile properties can be persisted across multiple processes, such as in a Web farm or a Web garden.
- **Platform scalability** Profile properties can be used in both multi-computer and multi-process configurations, therefore optimizing scalability scenarios.
- **Extensibility** In order to use profile properties, you must configure a profile provider. ASP.NET includes a [SqlProfileProvider](#) class that allows you to store profile data in a SQL database, but you can also create your own profile provider class that stores profile data in a custom format and to a custom storage mechanism, such as an XML file, or even to a Web service. For more information, see [ASP.NET Profile Providers](#) and [Implementing a Profile Provider](#).

Disadvantages of using profile properties are:

- **Performance considerations** Profile properties are generally slower than using session state because instead of storing data in memory, the data is persisted to a data store.
- **Additional configuration requirements** Unlike session state, the profile properties feature requires a considerable amount of configuration to use. To use profile properties, you must not only configure a profile provider, but you must pre-configure all of the profile properties that you want to store. For more information, see [ASP.NET Profile Properties Overview](#) and [Defining ASP.NET Profile Properties](#).
- **Data maintenance** Profile properties require a certain amount of maintenance. Because profile data is persisted to non-volatile storage, you must make sure that your application calls the appropriate cleanup mechanisms, which are provided by the profile provider, when data becomes stale.

Database Support

In some cases, you might want to use database support to maintain state on your Web site. Typically, database support is used in conjunction with cookies or session state. For example, it is common for an e-commerce Web site to maintain state information by using a relational database for the following reasons:

- Security

- Personalization
- Consistency
- Data mining

The following are typical features of a cookie-supported database Web site:

- **Security** The visitor types an account name and password into a site logon page. The site infrastructure queries the database with the logon values to determine whether the user has rights to utilize your site. If the database validates the user information, the Web site will distribute a valid cookie containing a unique ID for that user on that client computer. The site grants access to the user.
- **Personalization** With security information in place, your site can distinguish each user by reading the cookie on the client computer. Typically, sites have information in the database that describes the preferences of a user (identified by a unique ID). This relationship is known as *personalization*. The site can research the user's preferences using the unique ID contained in the cookie, and then place content and information in front of the user that pertains to the user's specific wishes, reacting to the user's preferences over time.
- **Consistency** If you have created a commerce Web site, you might want to keep transactional records of purchases made for goods and services on your site. This information can be reliably saved in your database and referenced by the user's unique ID. It can be used to determine whether a purchase transaction has been completed, and to determine the course of action if a purchase transaction fails. The information can also be used to inform the user of the status of an order placed using your site.
- **Data mining** Information about your site usage, your visitors, or your product transactions can be reliably stored in a database. For example, your business development department might want to use the data collected from your site to determine next year's product line or distribution policy. Your marketing department might want to examine demographic information about users on your site. Your engineering and support departments might want to look at transactions and note areas where your purchasing process could be improved. Most enterprise-level relational databases, such as Microsoft SQL Server, contain an expansive toolset for most data mining projects.

By designing the Web site to repeatedly query the database by using the unique ID during each general stage in the above scenario, the site maintains state. In this way, the user perceives that the site is remembering and reacting to him or her personally.

Advantages of using a database to maintain state are:

- **Security** Access to databases requires rigorous authentication and authorization.
- **Storage capacity** You can store as much information as you like in a database.
- **Data persistence** Database information can be stored as long as you like, and it is not subject to the availability of the Web server.
- **Robustness and data integrity** Databases include various facilities for maintaining good data, including triggers and referential integrity, transactions, and so on. By keeping information about transactions in a database (rather than in session state, for example), you can recover from errors more readily.
- **Accessibility** The data stored in your database is accessible to a wide variety of information-processing tools.
- **Widespread support** There is a large range of database tools available, and many custom configurations are available.

Disadvantages of using a database to maintain state are:

- **Complexity** Using a database to support state management requires that the hardware and software configurations be more complex.
- **Performance considerations** Poor construction of the relational data model can lead to scalability issues. Also, leveraging too many queries to the database can adversely affect server performance.

Server-Side Method State Management Summary

The following table lists the server-side state management options that are available with ASP.NET, and provides recommendations about when you should use each option.

State management option	Recommended usage
Application state	Use when you are storing infrequently changed, global information that is used by many users, and security is not an issue. Do not store large quantities of information in application state.
Session state	Use when you are storing short-lived information that is specific to an individual session and security is an issue. Do not store large quantities of information in session state. Be aware that a session-state object will be created and maintained for the lifetime of every session in your application. In applications hosting many users, this can occupy significant server resources and affect scalability.
Profile properties	Use when you are storing user-specific information that needs to be persisted after the user session is expired and needs to be retrieved again on subsequent visits to your application.

Database support	Use when you are storing large amounts of information, managing transactions, or the information must survive application and session restarts. Data mining is a concern, and security is an issue.
------------------	---

▴ See Also

Concepts

[ASP.NET State Management Overview](#)

[ASP.NET Cookies Overview](#)

[ASP.NET Profile Properties Overview](#)

[ASP.NET Session State Overview](#)

[ASP.NET Application State Overview](#)

Other Resources

[What's New in ASP.NET State Management](#)

[View State Overview](#)

Community Additions
