ASP.NET Cookies Overview

12/04/2014 • 26 minutes to read

In this article

What's New
See Also

A cookie is a small bit of text that accompanies requests and pages as they go between the Web server and browser. The cookie contains information the Web application can read whenever the user visits the site.

A Visual Studio project with source code is available to accompany this topic: Download .

This topic contains the following:

Scenarios

Cookies provide a means in Web applications to store user-specific information. For example, when a user visits your site, you can use cookies to store user preferences or other information. When the user visits your Web site another time, the application can retrieve the information it stored earlier.

Back to top

Background

A cookie is a small bit of text that accompanies requests and pages as they go between the Web server and browser. The cookie contains information the Web application can read whenever the user visits the site.

For example, if a user requests a page from your site and your application sends not just a page, but also a cookie containing the date and time, when the user's browser gets the page, the browser also gets the cookie, which it stores in a folder on the user's hard disk.

Later, if user requests a page from your site again, when the user enters the URL the browser looks on the local hard disk for a cookie associated with the URL. If the cookie exists, the browser sends the cookie to your site along with the page request. Your application can then determine the date and time that the user last visited the site. You might use the information to display a message to the user or check an expiration date.

Cookies are associated with a Web site, not with a specific page, so the browser and server will exchange cookie information no matter what page the user requests from your site. As the user visits different sites, each site might send a cookie to the user's browser as well; the browser stores all the cookies separately.

Cookies help Web sites store information about visitors. More generally, cookies are one way of maintaining continuity in a Web application—that is, of performing state management. Except for the brief time when they are actually exchanging information, the browser and Web server are disconnected. Each request a user makes to a Web server is treated independently of any other request. Many times, however, it's useful for the Web server to recognize users when they request a page. For example, the Web server on a shopping site keeps track of individual shoppers so the site can manage shopping carts and other user-specific information. A cookie therefore acts as a kind of calling card, presenting pertinent identification that helps an application know how to proceed.

Cookies are used for many purposes, all relating to helping the Web site remember users. For example, a site conducting a poll might use a cookie simply as a Boolean value to indicate whether a user's browser has already participated in voting so that the user cannot vote twice. A site that asks a user to log on might use a cookie to record that the user already logged on so that the user does not have to keep entering credentials.

Cookie Limitations

Most browsers support cookies of up to 4096 bytes. Because of this small limit, cookies are best used to store small amounts of data, or better yet, an identifier such as a user ID. The user ID can then be used to identify the user and read user information from a database or other data store. (See the section "Cookies and Security" below for information about security implications of storing user information.)

Browsers also impose limitations on how many cookies your site can store on the user's computer. Most browsers allow only 20 cookies per site; if you try to store more, the oldest cookies are discarded. Some browsers also put an absolute limit, usually 300, on the number of cookies they will accept from all sites combined.

A cookie limitation that you might encounter is that users can set their browser to refuse cookies. If you define a P3P privacy policy and place it in the root of your Web site, more browsers will accept cookies from your site. However, you might have to avoid cookies altogether and use a different mechanism to store user-specific information. A common method for storing user information is session state, but session state depends on cookies, as explained later in the section "Cookies and Session State."

> **Note**
>
> For more information on state management and options for saving information in a Web application, see **ASP.NET State Management Overview** and **ASP.NET State Management Recommendations**.

Although cookies can be very useful in your application, the application should not depend on being able to store cookies. Do not use cookies to support critical features. If your application must rely on cookies, you can test to see whether the browser will accept cookies. See the "Checking Whether a Browser Accepts Cookies" section later in this topic.

Writing Cookies

The browser is responsible for managing cookies on a user system. Cookies are sent to the browser via the HttpResponse object that exposes a collection called Cookies. You can access the HttpResponse object as the Response property of your Page class. Any cookies that you want to send to the browser must be added to this collection. When creating a

cookie, you specify a Name and Value. Each cookie must have a unique name so that it can be identified later when reading it from the browser. Because cookies are stored by name, naming two cookies the same will cause one to be overwritten.

You can also set a cookie's date and time expiration. Expired cookies are deleted by the browser when a user visits the site that wrote the cookies. The expiration of a cookie should be set for as long as your application considers the cookie value to be valid. For a cookie to effectively never expire, you can set the expiration date to be 50 years from now.

> **Note**
>
> Users can clear the cookies on their computer at any time. Even if you store cookies with long expiration times, a user might decide to delete all cookies, wiping out any settings you might have stored in cookies.

If you do not set the cookie's expiration, the cookie is created but it is not stored on the user's hard disk. Instead, the cookie is maintained as part of the user's session information. When the user closes the browser, the cookie is discarded. A non-persistent cookie like this is useful for information that needs to be stored for only a short time or that for security reasons should not be written to disk on the client computer. For example, non-persistent cookies are useful if the user is working on a public computer, where you do not want to write the cookie to disk.

You can add cookies to the Cookies collection in a number of ways. The following example shows two methods to write cookies:

VB Copy

```
Response.Cookies("userName").Value = "patrick"
Response.Cookies("userName").Expires = DateTime.Now.AddDays(1)

Dim aCookie As New HttpCookie("lastVisit")
aCookie.Value = DateTime.Now.ToString()
aCookie.Expires = DateTime.Now.AddDays(1)
Response.Cookies.Add(aCookie)
```

The example adds two cookies to the Cookies collection, one named userName and the other named lastVisit. For the first cookie, the values of the Cookies collection are set directly. You can add values to the collection this way because Cookies derives from a specialized collection of type NameObjectCollectionBase.

For the second cookie, the code creates an instance of an object of type HttpCookie, sets its properties, and then adds it to the Cookies collection via the Add method. When you instantiate an HttpCookie object, you must pass the cookie name as part of the constructor.

Both examples accomplish the same task, writing a cookie to the browser. In both methods, the expiration value must be of type DateTime. However, the lastVisited value is also a date-time value. Because all cookie values are stored as strings, the date-time value has to be converted to a String.

Cookies with More Than One Value

You can store one value in a cookie, such as user name and last visit. You can also store multiple name-value pairs in a single cookie. The name-value pairs are referred to as subkeys. (Subkeys are laid out much like a query string in a URL.) For example, instead of creating two separate cookies named userName and lastVisit, you can create a single cookie named userInfo that has the subkeys userName and lastVisit.

You might use subkeys for several reasons. First, it is convenient to put related or similar information into a single cookie. In addition, because all the information is in a single cookie, cookie attributes such as expiration apply to all the information. (Conversely, if you want to assign different expiration dates to different types of information, you should store the information in separate cookies.)

A cookie with subkeys also helps you limit the size of cookie files. As noted earlier in the "Cookie Limitations" section, cookies are usually limited to 4096 bytes and you can't store more than 20 cookies per site. By using a single cookie with subkeys, you use fewer of those 20 cookies that your site is allotted. In addition, a single cookie takes up about 50 characters for overhead (expiration information, and so on), plus the length of the value that you store in it, all of which counts toward the 4096-byte limit. If you store five subkeys instead of five separate cookies, you save the overhead of the separate cookies and can save around 200 bytes.

To create a cookie with subkeys, you can use a variation of the syntax for writing a single cookie. The following example shows two ways to write the same cookie, each with two subkeys:

VB Copy

```
Response.Cookies("userInfo")("userName") = "patrick"
Response.Cookies("userInfo")("lastVisit") = DateTime.Now.ToString()
Response.Cookies("userInfo").Expires = DateTime.Now.AddDays(1)

Dim aCookie As New HttpCookie("userInfo")
aCookie.Values("userName") = "patrick"
aCookie.Values("lastVisit") = DateTime.Now.ToString()
aCookie.Expires = DateTime.Now.AddDays(1)
Response.Cookies.Add(aCookie)
```

Controlling Cookie Scope

By default, all cookies for a site are stored together on the client, and all cookies are sent to the server with any request to that site. In other words, every page in a site gets all of the cookies for that site. However, you can set the scope of cookies in two ways:

- Limit the scope of cookies to a folder on the server, which allows you to limit cookies to an application on the site.
- Set scope to a domain, which allows you to specify which subdomains in a domain can access a cookie.

Limiting Cookies to a Folder or Application

To limit cookies to a folder on the server, set the cookie's Path property, as in the following example:

VB Copy

```
Dim appCookie As New HttpCookie("AppCookie")
appCookie.Value = "written " & DateTime.Now.ToString()
appCookie.Expires = DateTime.Now.AddDays(1)
appCookie.Path = "/Application1"
Response.Cookies.Add(appCookie)
```

> **Note**
>
> You can also write cookies by adding them to the **Cookies** collection directly as shown in earlier examples.

The path can either be a physical path under the site root or a virtual root. The effect will be that the cookie is available only to pages in the Application1 folder or virtual root. For example, if your site is called www.contoso.com, the cookie created in the previous example will be available to pages with the path http://www.contoso.com/Application1/ and to any pages beneath that folder. However, the cookie will not be available to pages in other applications such as http://www.contoso.com/Application2/ or just http://www.contoso.com/.

> **Note**
>
> In some browsers, the path is case sensitive. You cannot control how users type URLs into their browsers, but if your application depends on cookies tied to a specific path, be sure that the URLs in any hyperlinks you create match the case of the **Path** property value.

Limiting Cookie Domain Scope

By default, cookies are associated with a specific domain. For example, if your site is www.contoso.com, the cookies you write are sent to the server when users request any page from that site. (This might not include cookies with a specific path value.) If your site has subdomains—for example, contoso.com, sales.contoso.com, and support.contoso.com—then you can associate cookies with a specific subdomain. To do so, set the cookie's Domain property, as in this example:

VB

Copy

```
Response.Cookies("domain").Value = DateTime.Now.ToString()
Response.Cookies("domain").Expires = DateTime.Now.AddDays(1)
Response.Cookies("domain").Domain = "support.contoso.com"
```

When the domain is set in this way, the cookie will be available only to pages in the specified subdomain. You can also use the Domain property to create a cookie that can be shared among multiple subdomains, as shown in the following example:

VB

Copy

```
Response.Cookies("domain").Value = DateTime.Now.ToString()
Response.Cookies("domain").Expires = DateTime.Now.AddDays(1)
Response.Cookies("domain").Domain = "contoso.com"
```

The cookie will then be available to the primary domain as well as to sales.contoso.com and support.contoso.com domains.

Reading Cookies

When a browser makes a request to the server, it sends the cookies for that server along with the request. In your ASP.NET applications, you can read the cookies using the HttpRequest object, which is available as the Request property of your Page class. The structure of the HttpRequest object is essentially the same as that of the HttpResponse object, so you can read cookies out of the HttpRequest object much the same way you wrote cookies into the HttpResponse object. The following code example shows two ways to get the value of a cookie named username and display its value in a Label control:

VB

Copy

```
If Not Request.Cookies("userName") Is Nothing Then
    Label1.Text = Server.HtmlEncode(Request.Cookies("userName").Value)
```

```
End If

If Not Request.Cookies("userName") Is Nothing Then
    Dim aCookie As HttpCookie = Request.Cookies("userName")
    Label1.Text = Server.HtmlEncode(aCookie.Value)
End If
```

Before trying to get the value of a cookie, you should make sure that the cookie exists; if the cookie does not exist, you will get a NullReferenceException exception. Notice also that the HtmlEncode method was called to encode the contents of a cookie before displaying it in the page. This makes certain that a malicious user has not added executable script into the cookie. For more about cookie security, see the "Cookies and Security" section.

> **Note**
>
> Because different browsers store cookies differently, different browsers on the same computer won't necessarily be able to read each other's cookies. For example, if you use Internet Explorer to test a page one time, but then later use a different browser to test again, the second browser won't find the cookies saved by Internet Explorer.

Reading the value of a subkey in a cookie is likewise similar to setting it. The following code example shows one way to get the value of a subkey:

VB Copy

```
If Not Request.Cookies("userInfo") Is Nothing Then
    Label1.Text = _
        Server.HtmlEncode(Request.Cookies("userInfo")("userName"))
    Label2.Text = _
        Server.HtmlEncode(Request.Cookies("userInfo")("lastVisit"))
End If
```

In the preceding example, the code reads the value of the subkey lastVisit, which was set earlier to the string representation of a DateTime value. Cookies store values as strings, so if you want to use the lastVisit value as a date, you have to convert it

to the appropriate type, as in this example:

VB Copy

```
Dim dt As DateTime
dt = DateTime.Parse(Request.Cookies("userInfo")("lastVisit"))
```

The subkeys in a cookie are typed as a collection of type NameValueCollection . Therefore, another way to get an individual subkey is to get the subkeys collection and then extract the subkey value by name, as shown in the following example:

VB Copy

```
If Not Request.Cookies("userInfo") Is Nothing Then
    Dim UserInfoCookieCollection As _
        System.Collections.Specialized.NameValueCollection
    UserInfoCookieCollection = Request.Cookies("userInfo").Values
    Label1.Text = _
        Server.HtmlEncode(UserInfoCookieCollection("userName"))
    Label2.Text = _
        Server.HtmlEncode(UserInfoCookieCollection("lastVisit"))
End If
```

Changing a Cookie's Expiration Date

The browser is responsible for managing cookies, and the cookie's expiration time and date help the browser manage its store of cookies. Therefore, although you can read the name and value of a cookie, you cannot read the cookie's expiration date and time. When the browser sends cookie information to the server, the browser does not include the expiration information. (The cookie's Expires property always returns a date-time value of zero.) If you are concerned about the expiration date of a cookie, you must reset it, which is covered in the "Modifying and Deleting Cookies" section.

Note

You can read the **Expires** property of a cookie that you have set in the **HttpResponse** object, before the cookie has been sent to the browser. However, you cannot get the expiration back in the **HttpRequest** object.

Reading Cookie Collections

You might occasionally need to read through all the cookies available to the page. To read the names and values of all the cookies available to the page, you can loop through the Cookies collection using code such as the following.

VB Copy

```
Dim i As Integer
Dim output As System.Text.StringBuilder = New System.Text.StringBuilder
Dim aCookie As HttpCookie
For i = 0 to Request.Cookies.Count - 1
    aCookie = Request.Cookies(i)
    output.Append("Cookie name = " & Server.HtmlEncode(aCookie.Name) _
        & "<br />")
    output.Append("Cookie value = " & _
        Server.HtmlEncode(aCookie.Value) & "<br /><br />")
Next
Label1.Text = output.ToString()
```

Note

When you run this code, you might see a cookie named ASP.NET_SessionId. That is a cookie that ASP.NET uses to store a unique identifier for your session. The session cookie is not persisted on your hard disk. For more about session cookies, see the "Cookies and Session State" later in this topic.

A limitation of the preceding example is that if the cookie has subkeys, the display shows the subkeys as a single name/value string. You can read a cookie's HasKeys property to determine whether the cookie has subkeys. If so, you can read the

subkey collection to get individual subkey names and values. You can read subkey values from the Values collection directly by index value. The corresponding subkey names are available in the AllKeys member of the Values collection, which returns an array of strings. You can also use the Keys member of the Values collection. However, the AllKeys property is cached the first time it is accessed. In contrast, the Keys property builds an array each time it is accessed. For this reason, the AllKeys property is much faster on subsequent accesses within the context of the same page request.

The following example shows a modification of the preceding example. It uses the HasKeys property to test for subkeys, and if subkeys are detected, the example gets subkeys from the Values collection:

VB Copy

```
Dim i As Integer
Dim j As Integer
Dim output As System.Text.StringBuilder = New StringBuilder()
Dim aCookie As HttpCookie
Dim subkeyName As String
Dim subkeyValue As String
For i = 0 To Request.Cookies.Count - 1
    aCookie = Request.Cookies(i)
    output.Append("Name = " & aCookie.Name & "<br />")
    If aCookie.HasKeys Then
        For j = 0 To aCookie.Values.Count - 1
            subkeyName = Server.HtmlEncode(aCookie.Values.AllKeys(j))
            subkeyValue = Server.HtmlEncode(aCookie.Values(j))
            output.Append("Subkey name = " & subkeyName & "<br />")
            output.Append("Subkey value = " & subkeyValue & _
                "<br /><br />")
        Next
    Else
        output.Append("Value = " & Server.HtmlEncode(aCookie.Value) & _
            "<br /><br />")
    End If
Next
Label1.Text = output.ToString()
```

Alternatively, you can extract the subkeys as a NameValueCollection object as shown in the following example:

VB

Copy

```
Dim i As Integer
Dim j As Integer
Dim output As System.Text.StringBuilder = New StringBuilder()
Dim aCookie As HttpCookie
Dim subkeyName As String
Dim subkeyValue As String
For i = 0 To Request.Cookies.Count - 1
    aCookie = Request.Cookies(i)
    output.Append("Name = " & aCookie.Name & "<br />")
    If aCookie.HasKeys Then
        Dim CookieValues As _
            System.Collections.Specialized.NameValueCollection = _
                aCookie.Values
        Dim CookieValueNames() As String = CookieValues.AllKeys
        For j = 0 To CookieValues.Count - 1
            subkeyName = Server.HtmlEncode(CookieValueNames(j))
            subkeyValue = Server.HtmlEncode(CookieValues(j))
            output.Append("Subkey name = " & subkeyName & "<br />")
            output.Append("Subkey value = " & subkeyValue & _
                "<br /><br />")
        Next
    Else
        output.Append("Value = " & Server.HtmlEncode(aCookie.Value) & _
            "<br /><br />")
    End If
Next
Label1.Text = output.ToString
```

Modifying and Deleting Cookies

You cannot directly modify a cookie. Instead, changing a cookie consists of creating a new cookie with new values and then sending the cookie to the browser to overwrite the old version on the client. The following code example shows how you can change the value of a cookie that stores a count of the user's visits to the site:

VB Copy

```
Dim counter As Integer
If Request.Cookies("counter") Is Nothing Then
    counter = 0
Else
    counter = Int32.Parse(Request.Cookies("counter").Value)
End If
counter += 1
Response.Cookies("counter").Value = counter.ToString
Response.Cookies("counter").Expires = DateTime.Now.AddDays(1)
```

Deleting Cookies

Deleting a cookie—physically removing it from the user's hard disk—is a variation on modifying it. You cannot directly remove a cookie because the cookie is on the user's computer. However, you can have the browser delete the cookie for you. The technique is to create a new cookie with the same name as the cookie to be deleted, but to set the cookie's expiration to a date earlier than today. When the browser checks the cookie's expiration, the browser will discard the now-outdated cookie. The following code example shows one way to delete all the cookies available to the application:

VB Copy

```
Dim aCookie As HttpCookie
Dim i As Integer
Dim cookieName As String
Dim limit As Integer = Request.Cookies.Count - 1
For i = 0 To limit
    cookieName = Request.Cookies(i).Name
    aCookie = New HttpCookie(cookieName)
```

```
    aCookie.Expires = DateTime.Now.AddDays(-1)
    Response.Cookies.Add(aCookie)
Next
```

Modifying or Deleting Subkeys

Modifying an individual subkey is the same as creating it, as shown in the following example:

VB Copy

```
Response.Cookies("userInfo")("lastVisit") = DateTime.Now.ToString()
Response.Cookies("userInfo").Expires = DateTime.Now.AddDays(1)
```

To delete an individual subkey, you manipulate the cookie's Values collection, which holds the subkeys. You first recreate the cookie by getting it from the Cookies object. You can then call the Remove method of the Values collection, passing to the Remove method the name of the subkey to delete. You then add the cookie to the Cookies collection so it will be sent in its modified form back to the browser. The following code example shows how to delete a subkey. In the sample, the name of the subkey to remove is specified in a variable.

VB Copy

```
Dim subkeyName As String
subkeyName = "userName"
Dim aCookie As HttpCookie = Request.Cookies("userInfo")
aCookie.Values.Remove(subkeyName)
aCookie.Expires = DateTime.Now.AddDays(1)
Response.Cookies.Add(aCookie)
```

Cookies and Security

The security issues with cookies are similar to those of getting data from the client. In your application, cookies are another form of user input and are therefore subject to examining and spoofing. A user can as a minimum see the data that you store in a cookie, since the cookie is available on the user's own computer. The user can also change the cookie before the browser sends it to you.

You should never store sensitive data in a cookie, such as user names, passwords, credit card numbers, and so on. Do not put anything in a cookie that should not be in the hands of a user or of someone who might somehow steal the cookie.

Similarly, be suspicious of information you get out of a cookie. Do not assume that the data is the same as when you wrote it out; use the same safeguards in working with cookie values that you would with data that a user has typed into a Web page. The examples earlier in this topic showed HTML-encoding the contents of a cookie before displaying the value in a page, as you would before displaying any information you get from users.

Cookies are sent between browser and server as plain text, and anyone who can intercept your Web traffic can read the cookie. You can set a cookie property that causes the cookie to be transmitted only if the connection uses the Secure Sockets Layer (SSL). SSL does not protect the cookie from being read or manipulated while it is on the user's computer, but it does prevent the cookie from being read while in transit because the cookie is encrypted. For more information, see Basic Security Practices for Web Applications.

Determining Whether a Browser Accepts Cookies

Users can set their browser to refuse cookies. No error is raised if a cookie cannot be written. The browser likewise does not send any information to the server about its current cookie settings.

> **Note**
>
> The **Cookies** property does not indicate whether cookies are enabled. It indicates only whether the current browser inherently supports cookies.

One way to determine whether cookies are accepted is by trying to write a cookie and then trying to read it back again. If you cannot read the cookie you wrote, you assume that cookies are turned off in the browser.

The following code example shows how you might test whether cookies are accepted. The sample consists of two pages. The first page writes out a cookie, and then redirects the browser to the second page. The second page tries to read the cookie. It in turn redirects the browser back to the first page, adding to the URL a query string variable with the results of the test.

The code for the first page (AcceptsCookies.aspx) looks like the following example:

VB Copy

```
Protected Sub Page_Load(ByVal sender As Object, _
        ByVal e As EventArgs) Handles Me.Load
    If Not Page.IsPostBack Then
        If Request.QueryString("AcceptsCookies") Is Nothing Then
            Response.Cookies("TestCookie").Value = "ok"
            Response.Cookies("TestCookie").Expires = _
                DateTime.Now.AddMinutes(1)
            Response.Redirect("TestForCookies.aspx?redirect=" & _
                Server.UrlEncode(Request.Url.ToString))
        Else
            Label1.Text = "Accept cookies = " & _
                Server.UrlEncode(Request.QueryString("AcceptsCookies"))
        End If
    End If
End Sub
```

The page first tests to see if this is a postback, and if not, the page looks for the query string variable name AcceptsCookies that contains the test results. If there is no query string variable, the test has not been completed, so the code writes out a cookie named TestCookie. After writing out the cookie, the sample calls Redirect to transfer to the test page TestForCookies.aspx. Appended to the URL of the test page is a query string variable named redirect containing the URL of the current page; this will allow you to redirect back to this page after performing the test.

The test page can consist entirely of code; it does not need to contain controls. The following example illustrates the test page (TestForCookies.aspx).

VB

Copy

```
Sub Page_Load()
    Dim redirect As String = Request.QueryString("redirect")
    Dim acceptsCookies As String
    If Request.Cookies("TestCookie") Is Nothing Then
        acceptsCookies = "no"
    Else
        acceptsCookies = "yes"
        ' Delete test cookie.
        Response.Cookies("TestCookie").Expires = _
            DateTime.Now.AddDays(-1)
    End If
    Response.Redirect(redirect & "?AcceptsCookies=" & acceptsCookies, _
       True)
End Sub
```

After reading the redirect query string variable, the code tries to read the cookie. For housekeeping purposes, if the cookie exists, it is immediately deleted. When the test is finished, the code constructs a new URL from the URL passed to it in the redirect query string variable. The new URL also includes a query string variable containing test results. The final step is to use the new URL to redirect the browser to the original page.

An improvement in the example would be to keep the cookie test results in a persistent store such as a database so that the test does not have to be repeated each time the user views the original page. (Storing the test results in session state by default requires cookies.)

Cookies and Session State

When a user navigates to your site, the server establishes a unique session for that user that lasts for the duration of the user's visit. For each session, ASP.NET maintains session state information where applications can store user-specific information. For more information, see ASP.NET Session State Overview topic.

ASP.NET must track a session ID for each user so that it can map the user to session state information on the server. By default, ASP.NET uses a non-persistent cookie to store the session state. However, if a user has disabled cookies on the browser, session state information cannot be stored in a cookie.

ASP.NET offers an alternative in the form of cookieless sessions. You can configure your application to store session IDs not in a cookie, but in the URLs of pages in your site. If your application relies on session state, you might consider configuring it to use cookieless sessions. However, under some limited circumstances, if the user shares the URL with someone else—perhaps to send the URL to a colleague while the user's session is still active—then both users can end up sharing the same session, with unpredictable results. For more information on configuring your application to use cookieless sessions, see the ASP.NET State Management Overview topic.

Back to top

Code Examples

QuickStarts

Managing Application State

How to and Walkthrough Topics

How to: Customize the Authentication Cookie from the WCF Authentication Service

Back to top

Class Reference

HttpCookie	Provides a type-safe way to create and manipulate individual HTTP cookies.
Cookies	Gets the response cookie collection.
Cookies	Gets a collection of cookies sent by the client.

Back to top

Additional Resources

Back to top

What's New

Back to top

See Also

Reference

Back to top

Concepts

ASP.NET State Management Overview