

## 27.1 Introduction: Dealing with statelessness (part 2)

In [Section 26.3.4](#), you used a query string to maintain state information across different pages of your web application. Specifically, when the user entered the correct user name and password, the state variable **LI** was set to “True” and passed from page to page in the URL (e.g., `http://.../ASPTest.asp?LI=True`). Thus, it was possible for the server to recognize the users logged-in “state” even though the TCP/IP connection used by the Internet is stateless.

Of course, one downside of using query strings (or even hidden fields) to pass sensitive state information (such as whether the user has been authenticated) is that this information is in plain view. Thus, in order to make it more difficult to bypass the authorization logic, we changed the value of **LI** from “True” to something less obvious (a hash value) in [Section 26.3.5](#).

The use of hash values and query strings is extremely common on the Internet (just log in to a site that retains state information such as [www.expedia.com](http://www.expedia.com) and examine the contents of your browser’s URL window). However, server-side scripting environments (such as ASP and

JSP) provide a simpler mechanism for storing state information across pages: **sessions**.

### 27.1.1 ASP’s Session object

In [Lesson 26](#), we used two of ASP’s built-in objects: **Response** and **Request**. These two objects correspond exactly to the HTTP response and request constructs, so they are fairly easy to understand. There is no HTTP counterpart for the **session** object, however (hence the need for it in the first place).

A session is defined as a user’s visit to a web-based application. Within the session, the user may visit many different pages within the application and even leave the application briefly and return. The **session** object is simply an “after market” solution to the statelessness of the underlying HTTP protocol. It permits ASP designers to read and write server-side session variables that persist throughout the entire session, regardless of which pages are viewed.

### 27.1.2 Session variables and scope

The “scope” of normal ASP variables is the page on which they are created. Thus, although you could define a variable on a page:

```
NL <HTML>
```



```
NL <BODY>
NL <% strFavColor="Blue" %>
NL <P>My favorite color is:
NL <%= strFavColor %></P>
NL </BODY>
NL </HTML>
```

the value of `strFavColor` would not be available on any other page. Session variables, in contrast, are available on any page in the application for the duration of the session.

### 27.1.2.1 Declaring a session variable

To create a session variable, you simply give it a name within the collection of variables stored in the `Session` object. For example, the following assignment statement can be used to store the user's favorite color in a session variable called `strFavColor`:

```
NL <HTML>
NL <BODY>
NL <% Session("strFavColor") = "Blue"
NL %>
NL </BODY>
NL </HTML>
```

### 27.1.2.2 Using a session variable

Now, on a completely different page (within the same session), the value of the session variable can be recalled:

```
NL <HTML>
NL <HEAD>
```

```
NL <TITLE>A Different Page</TITLE>
NL </HEAD>
NL <BODY>
NL <P>Your favorite color is:
NL <%= Session("strFavColor") %></P>
NL </BODY>
NL </HTML>
```

Unlike a query string, the information in a session variable never gets passed back to the client. Instead, it is stored in the server's memory and can be made invisible to the user. Consequently, session variables can be used to implement a simple-but-robust authorization mechanism.

## 27.2 Learning objectives

- create session variable
- understand how session variables can be used to simplify authentication
- use `Session.Abandon` to free-up server resources
- understand the role of cookies in providing session functionality

## 27.3 Exercises

In this section you will modify the authorization scheme you created in [Lesson 26](#) to use a session variable instead of a query string.



In the exercises that follow, you will replace some of the code you wrote in [Lesson 26](#) with new code based on session variables. If you have a sentimental attachment to the versions of **Authorize.asp** and **Menu.asp** that use hash values and query strings, you should save a backup copy of these files to a different directory before continuing.

### 27.3.1 Controlling branching using session variables

- ➔ Edit **Authorize.asp** to replace the hash value and query string code with the following code:

```
NL <%
NL  strUserName=Request.Form.
NL  Item("txtUserName")
NL  strPassword=Request.Form.
NL  Item("txtPassword")
NL  If strUserName=strPassword Then
NL    Session("LI") = "True"
NL    Response.Redirect "Menu.asp"
NL  Else
NL    Session("LI") = "Fail"
NL    Response.Redirect "Login.asp"
NL  End if
NL %>
```

- ➔ Edit **Menu.asp** to use the value of the session variable instead of the hash value to control access:

```
NL <%
NL  If Session("LI") <> "True" Then
NL    Response.Redirect "Login.asp"
NL  End If
NL %>
NL <HTML>
NL ...
NL </HTML>
```



Remember that a **Redirect** method will not work if the server has already written something to the HTTP response. Thus, you must ensure that the VBSCRIPT code is located *above* the **<HTML>** tag.

The changes to **Menu.asp** are shown in [Figure 27.1](#).

- ➔ Edit **Login.asp** to use the value of the session variable instead of the query string to display the "login incorrect" message:

```
NL <HTML>
NL ...
NL <BODY>
NL <% If Session("LI") = "Fail" Then
NL   %>
NL   <H3>Login incorrect: please try
NL     again</H3>
NL <% Else %>
```

FIGURE 27.1: Replace the hash value/query string authorization mechanism with one based on session variables.

1 Create a simple session variable and assign it the value "True" if authorization is successful.

```

Authorize.asp - Notepad
File Edit Format Help
strUserName=Request.Form.
Item("txtUserName")
strPassword=Request.Form.
Item("txtPassword")
If strUserName=strPassword Then
  Session("LI") = "True"
  Response.Redirect "Menu.asp"
Else
  Session("LI") = "Fail"
  Response.Redirect "Login.asp"
End if
%>
    
```

2 Use the same session variable to store whether the authorization attempt was unsuccessful.

3 Use the session variable in the conditional redirect at the top of the target page(s).

```

menu.asp - Notepad
File Edit Format Help
If Session("LI") <> "True" Then
  Response.Redirect "Login.asp"
End If
%>
<HTML>
<HEAD>
<TITLE>Kitchen Supply Co. Extranet: Main Menu</TITLE>
</HEAD>

<BODY>
<H2>Kitchen Supply
<FORM METHOD="post"
ACTION="http://bry
<TABLE>
<TR>
<TD>Update Customer Profiles/10/
    
```

Since session variables cannot be viewed or manipulated by the client-side, a hash value

```

NL <H3>Please enter your user name
and password</H3>
NL <% End if %>
NL ...
NL </HTML>
    
```

Clearly, the **session** object makes keeping track of a user's progress through the application much easier.

### 27.3.2 Ending a session

Although the server will eventually end a session if a specified amount of time elapses without activity (e.g., 20 minutes), it is important to realize that each session ties up a certain amount of server resources. If there are many thousands of users connected to the site simultaneously, then the server has to keep track of many thousand individual sessions. In extreme circumstances, the requirements of



managing all the sessions can overwhelm the web server. Having a logoff mechanism that allows users to explicitly end their session has two advantages:

1. It frees session-specific resources as soon as possible.
2. It eliminates the security risk created by users walking away from machines while an authorized session is still active.



You will notice that most secure web applications (such as on-line banking) have explicit logoff procedures.

The easiest way to end a session in ASP is to transfer the user to a page that contains a `Session.Abandon` statement. The `Abandon` method destroys the session and any session-level variables stored with the session.

- ➔ Rename your `Logoff.html` file to `Logoff.asp`.
- ➔ Add a message similar to "Thank you, you are now logged off" to the body of the document.
- ➔ Add the following code to the end of the document (following the `</HTML>` tag):

```
NL <HTML>
NL ...
```

```
NL <P>Thank you, you are now logged
off</P>
NL ...
NL </HTML>
NL <% Session.Abandon %>
```

Once the session is abandoned, the value of `Session("LI")` no longer equals "True". Thus, even if an unauthorized user uses the browser's **Back** button or opens a cached copy of page, he or she will be unable to get past the conditional redirects that you will place at the top of all your content pages in [Section 27.5](#).

### 27.3.3 Limiting page caching

To create the illusion of speed, web browsers typically "cache" pages as they are visited. Thus, when you hit the back button or revisit a page in other ways, there is a good chance that you are looking at the copy of the page that is cached on your local hard drive, not the page on the web server. Thus, it is possible so see the contents of the cached pages even after logging out. If you have sensitive data on a page (e.g., banking information) you can use the ASP `Response` object to tell browser not to cache the page:

```
NL <% Response.Expires = 0 %>
NL <HTML>
NL ...
NL </HTML>
```



Whether the browser actually obeys this directive is beyond your control as a web designer.

## 27.4 Discussion: Cookies and sessions

The `session` object is an easy way to achieve persistence of variables within a session. Although the feature may seem complicated, the implementation of the `session` object is not that different from the query string/hash value approach used in [Section 26.3.5](#). The main difference is that the Session ID (a hash value that uniquely identifies a session) is created automatically by the server and passed back and forth between the server and client as an HTTP **cookie** instead of a query string.

When you send a request to an ASP page for the first time, a number of things occur:

- The server creates a Session ID value that is unique to the session (assume “123” for simplicity’s sake).
- Any session variables created during the user’s interaction with the web application are stored in the server’s memory and tagged as belonging to Session ID = 123 (remember that there may be thousands of concurrent users, each with a unique Session ID and set of session variables).

- The server appends the SessionID to an HTTP header variable called **HTTP\_COOKIE**. The header is included in the response sent back to the client.
- The browser extracts the cookie from the server’s HTTP response . The data in the cookie is saved to a small text file on the client computer along with sender information that allows the browser to associate the cookie with the name of the server that sent it.
- Whenever the client sends an HTTP request to the server, it checks to see whether it has saved a cookie associated with that particular server. If so, the browser sends the data in the cookie (e.g., Session ID = “123”) back to the server.

The server can use the cookie value to “recognize” the session and retrieve any server-side variables assigned to it. In this way, ASP creates the illusion of a contiguous session.



Cookies are site-specific. That is, your browser is responsible for ensuring that a cookie deposited by Server X is not sent to Server Y, and vice-versa.

### 27.4.1 Session example

To illustrate the use of cookies, do the following:



- ➡ Close your browser. If you have multiple browser windows open, make sure they are all closed. This eliminates any session-level (temporary) cookies you may have open.
- ➡ Re-open a browser windows and send an HTTP request to `EchoRequest.asp` (recall [Section 25.3](#)).

At first, the “Server Variables” section (near the bottom of the page) should not contain any information about an ASP session. However, when you hit the **Refresh** button, you should see that a Session ID has been deposited on your computer and sent back to the server. The contents of the cookie should look something like the following:

```
HTTP_COOKIE =  
ASPSESSIONIDQQGGQQFB=DIHJOFOANKGNILQAFON  
GJIDL
```

### 27.4.1.1 Cookie expiration

How do the two computers know when the session is over? Each cookie has an expiry date. The Session ID cookies, which are created automatically by the ASP-enabled web server, expire immediately at the end of the session (e.g., when the browser is closed). On the server-side, a session time-out can be set by the server administrator. When there is no activity from the client for 20 minutes (the default), the

session is closed and all the server-side session variables are released.

### 27.4.1.2 Longer-term cookies

Of course, not all cookies are created automatically. In ASP for example, the `Response.Cookies` collection can be used to create cookie key/value pairs and set cookie-level properties such as expiry date. If a cookie's expiry date is set to a date in the future, the cookie file saved on the client's hard drive is not deleted at the end of the session. In this way, the cookie values are available for many sessions, even if the browser and/or computer are shut down.

Persistent cookies are used by many web sites to automatically identify you whenever you visit the site. From a user's perspective, cookies are convenient if they are used to personalize the site or eliminate the need to login manually with each visit. From the site owner's perspective, cookies are valuable because they permit the behavior of users within the site to be tracked, both within a particular visit and across multiple visits.

For example, if you visit `www.volvo.com` ten times in one week to look at the same car, you may be a promising sales lead for VOLVO. However, without a persistent cookie, there is no reliable way to know that the ten visits came



from the same computer. Naturally, the use of cookies in this manner leads to concerns about privacy. And at a more basic level, some people simply do not like the idea of a remote computer writing something to their hard drive.



It is important to realize that a cookie's potential to do harm is constrained by the HTTP protocol on one hand and your browser (which is solely responsible for the creation and management of cookie files) on the other. Cookies are just key/value pairs—nothing more. A cookie cannot erase your hard drive, implant a virus, or spy on you.

### 27.4.1.3 Cookie design issues

Given the number of times a cookie's contents are sent to the server, it is best to minimize the amount of data stored in the cookie. As such, most cookies contain nothing but an identification value. The identification value (or primary key, if you prefer) is used to access more extensive visitor information that is stored on a server-side database. Where does the more extensive server-side visitor information come from? Typically, you provide it when you register for a site, enter a contest, or fill in any type of form.



If you provide the site with certain key information (such as your postal code or social insurance number), it may be possible for the site to combine data from multiple sources to get a very clear picture of who you are. From a privacy point of view, the problem is not the cookie. Rather, it is the widespread practice of reselling and merging data that you have willingly provided about yourself.

If you are curious about the contents of the cookies on your computer's hard drive, you can download one of the many free "cookie viewers" available over the Internet. For example, [Figure 27.2](#) shows the contents of a cookie left by a MICROSOFT web site.

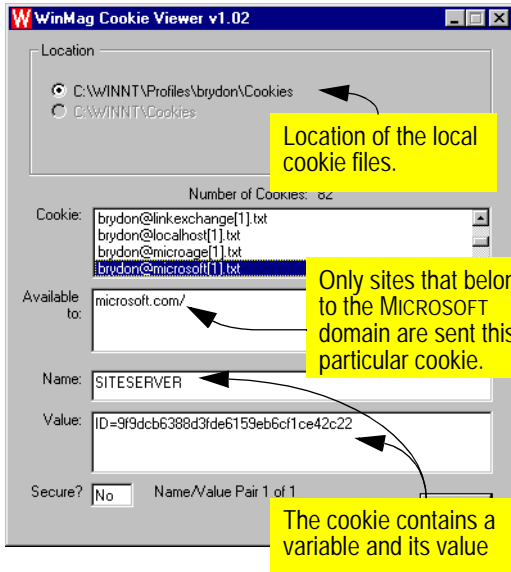
## 27.5 Application to the project

It is important that each page of your extranet require authentication. As such, you should include a conditional redirect at the top of each content page to ensure that the only way the page can be accessed is if a session variable is set to some value first. The only way to set the value of the session variable is to login successfully.

- ➔ Add a conditional redirect (similar to that shown on the right in [Figure 27.1](#)) to all the



FIGURE 27.2: The contents of a cookie created by a MICROSOFT web site.



content pages of your web-based order entry system. Of course, the login page should not include this code.



Since you are adding script to the pages, the file names must be changed to end with the .asp extension.

