# MSIT 537 ASP Module Number 2

## QUERYSTRINGS, COOKIES, AND SESSIONS

### Passing Information Through the Querystring

When you only need to maintain state for the duration of a user's visit to your site, you have a couple of options. If you only need to save simple data types, a series of cookies will suffice. If more complex data types need to be used, you can store this information through the Session object. (The Session object uses cookies to uniquely identify each visitor to your site. We discuss the inner workings of the Session object later in "The Session Object.") The drawback to both these methods is that if the user has cookies disabled, your Web site will appear to be stateless.

Although the vast majority of Web surfers today have cookies enabled, if it is essential that your site maintain state for all your visitors, cookies and session variables just won't do. If this is ever the case, and all you need to persist is simple data types, you can store your state information in the querystring.

The Request object can process information from the querystring if it is formatted in name/value pairs, with each name and value separated by an equals sign (=), and each name/value pair separated from one another by an ampersand (&). For example, a querystring that contained my name and age might look like:

```
?Name=Scott&Age=21
```

Imagine that when users first come to your Web site, you ask them to enter their name. You would need to create a form that contained a text box where users could enter their names, and the form also would need a submit button. Listing 11.1 shows the HTML needed to create such a form.


**Listing 11.1 - What Is Your Name?**

```
1:  <FORM METHOD=GET ACTION="Welcome.asp">
2:    What is your name?<BR>
3:    <INPUT TYPE=TEXT NAME=Name>
4:    <P>
5:    <INPUT TYPE=SUBMIT VALUE="Login">
6:  </FORM>
```

Line 1 in Listing 11.1 creates the form with the METHOD property set to GET. This sends the results of the form through the querystring. The code in Listing 11.1 also creates a text box for the

users to enter their names (line 3) and a submit button (line 5). Lines 1-6 shows the code for Listing 11.1 when viewed through a browser

When the form created in Listing 11.1 is submitted, Welcome.asp, the form processing script, is called. Welcome.asp is passed the name of the user through the querystring. The information you want to persist is the user's name, which is in the querystring. It's obvious that Welcome.asp can ascertain the user's name (via Request.QueryString("Name")), but how can you ensure that other Web pages on your site will have access to this information?

The secret is to make sure that each and every ASP page contains the same querystring that Welcome.asp contains. If you can ensure uniformity of the querystring across your Web site, then each ASP page that needed to obtain the user's name could do so by using Request.QueryString("Name"). The question now is how can each Web page on your site have the same querystring as Welcome.asp.

A hyperlink is created with the following syntax:

```
<A HREF="URL">The title of the link</A>
```

To pass the querystring to the URL, you need to append a question mark (?) followed by the querystring. The following code would create a hyperlink titled Click Me that would pass to ClickMe.asp the querystring Name=Scott:

```
<A HREF="ClickMe.asp?Name=Scott">Click Me</A>
```

Creating hyperlinks with a static querystring, however, will not suffice. You need the hyperlink URL's querystring to be equal to the current querystring in Welcome.asp. This way, you will maintain state for each user. Recall that for an ASP page, the entire, current querystring can be read using Request.QueryString. You can create all your hyperlinks so that they pass the current querystring by using the following syntax:

```
<A HREF="somePage.asp?<%=Request.QueryString%>">Click Me</A>
```

The preceding syntax will create a hyperlink that, when clicked, will pass the ASP page somePage.asp the current querystring. Recall that when an ASP page is requested from a Web server, all the ASP code is processed on the server and the client is sent pure HTML. The client does not receive:

```
<A HREF="somePage.asp?<%=Request.QueryString%>">Click Me</A>
```

Rather, the value of Request.QueryString is inserted after somePage.asp?. For example, say that Welcome.asp's querystring is Name=James. If you create a hyperlink in Welcome.asp using the following syntax:

```
<A HREF="somePage.asp?<%=Request.QueryString%>">Click Me</A>
```

the Web browser, when visiting Welcome.asp, would receive the HTML as follows:

```
<A HREF="somePage.asp?Name=James">Click Me</A>
```

Now that you have sent the querystring Welcome.asp you received to somePage.asp, somePage.asp can access the user's name with Request.QueryString("Name"). Listing 11.1 created a form where the user could enter her name. This form, when submitted, sent the user's name to Welcome.asp through the querystring. All the hyperlinks in Welcome.asp need to pass the current querystring to their respective URLs. The code for Welcome.asp can be seen in Listing 11.2.

**Listing 11.2 - Inserting the Current Querystring into all the Hyperlinks**

```
1:  <%@ Language=VBScript %>
2:  <% Option Explicit %>
3:  <%
4:      'Read the Name
5:      Dim strName
6:      strName = Request.QueryString("Name")
7:  %>
8:  <HTML>
9:  <BODY>
10:   Hello <%=strName%>!
11:   <P>
12:   What interests you?<BR>
13:   <LI><A HREF="sports.asp?<%=Request.QueryString%>">Sports</A><BR>
14:   <LI><A HREF="politics.asp?<%=Request.QueryString%>">Politics</A><BR>
15:   <LI><A HREF="fashion.asp?<%=Request.QueryString%>">Fashion</A><BR>
16:   <LI><A HREF="events.asp?<%=Request.QueryString%>">Current Events</A><BR>
17: </BODY>
18: </HTML>
```


Welcome.asp starts by reading in the user's name (line 6). When using the querystring to maintain state, all the ASP pages on your Web site should start out by reading the persistent information; in this case, the user's name. Line 10 displays a personalized greeting, and lines 13 through 16 create a series of hyperlinks. Notice that each hyperlink passes its URL the current querystring using Request.QueryString. Figure 11.3 shows Welcome.asp when viewed through a browser.

You might be wondering what the HTML the browser received from Welcome.asp looked like. Listing 11.3 reveals the exact HTML received by the browser when Welcome.asp was sent Name=Scott in the querystring.

**Listing 11.3 - The HTML Received by the Browser When Visiting Welcome.asp**

```
1:  <HTML>
2:  <BODY>
3:    Hello Scott!
4:    <P>
5:    What interests you?<BR>
6:    <LI><A HREF="sports.asp?Name=Scott">Sports</A><BR>
7:    <LI><A HREF="politics.asp?Name=Scott">Politics</A><BR>
```

```
8:    <LI><A HREF="fashion.asp?Name=Scott">Fashion</A><BR>
9:    <LI><A HREF="events.asp?Name=Scott">Current Events</A><BR>
10: </BODY>
11: </HTML>
```

Notice in lines 6 through 9 that the querystring, Name=Scott, was appended to the URL in the hyperlink. This ensures that when any of these hyperlinks are clicked, it will be sent the current URL. This ensures that the user's name will persist on the next ASP page on your site that the user reaches via a hyperlink.

> ***Note***
>
> To have the user's name be maintained throughout your Web pages, every page must be passed the user's name through the querystring. To ensure that every page is passed the user's name through the querystring, every hyperlink on every ASP page must have ?<%=Request.QueryString%> appended to it! This may seem like a burden and a headache - it is.

The querystring solution for maintaining state is not without pitfalls. Imagine that a user entered her information and surfed through a couple of pages on your site by clicking the hyperlinks, so far, so good. Now, imagine that the user wants to visit a specific URL on your site, so they type it in the Address bar. The user will reach that URL without having passed the persistent information. At this point, state has been lost. The querystring method also is a development headache. If you forget to append the querystring to a hyperlink's URL, when that hyperlink is clicked, state will be lost because the querystring won't contain the maintained information. Also, the querystring method cannot persist objects because it would be impossible to express an object via the querystring. Finally, keep in mind that the querystring method can only persist data while the user is on your Web site. The second the user leaves your site, state is lost.

The querystring approach, despite its disadvantages, will always work with any browser, whether or not the user has disabled cookies. Also, the querystring approach is free of the performance concerns that plague you when dealing with the Session object, which is discussed in length later today in "The Session Object." If, for the duration of a user's visit, you must have information persisted, regardless of whether the user has cookies enabled, the querystring approach is the way to go.

## Using Cookies

Cookies are small text files written to the client's computer. Cookies can be written to the client's computer using the Response.Cookies collection and can be read using the Request.Cookies collection.

Cookies can persist on the client's computer for a variable amount of time. When writing a cookie to the client's computer, you can set when the cookie expires. This can be in a day, a week, or even a year. Cookies allow a user's state to be maintained beyond the current visit. The section "Passing Information Through the Querystring" earlier looked at persisting a user's name throughout all the ASP pages on your Web site. Let's examine how you can use cookies to maintain this information.

Listing 11.1 created a simple form in which the user was prompted to enter his or her name. This form, when submitted, passed the user's name through the querystring to Welcome.asp, the form processing script. Because you are going to use cookies to maintain state in this example, you don't need to worry about having all the hyperlinks in Welcome.asp passing along the querystring

information. All you need to do in Welcome.asp is write the name of the user to a cookie. Listing 11.4 shows Welcome.asp, modified to use cookies to maintain the user's name.

**Listing 11.4 - Using Cookies to Maintain State**

```
1:  <%@ Language=VBScript %>
2:  <% Option Explicit %>
3:  <%
4:      'Read in the Name from the form
5:      Dim strName
6:      strName = Request("Name")
7:
8:      'Write the user's name to a cookie
9:      Response. Cookies("UserName") = strName
10:
11:     'Set the cookie to expire in a week
12:     Response.Cookies("UserName").Expires = Date() + 7
13: %>
14:
15: <HTML>
16: <BODY>
17:    Hello <%=strName%>!
18:    <P>
19:    What interests you?<BR>
20:    <LI><A HREF="sports.asp">Sports</A><BR>
21:    <LI><A HREF="politics.asp">Politics</A><BR>
22:    <LI><A HREF="fashion.asp">Fashion</A><BR>
23:    <LI><A HREF="events.asp">Current Events</A><BR>10: </BODY>
24: </HTML>
```

To maintain state using cookies, you only need to write the cookie to the client's computer once: when the user enters the information that you want to persist. After the user enters his or her name into the form created by Listing 11.1, Welcome.asp, shown in Listing 11.4, is called. Line 6 starts off by reading the user's name into the variable strName. Line 9 then creates a cookie named UserName and writes to it the user's name. Line 12 sets the cookie to expire in seven days. Assuming that the user accepts cookies, the user's name will be persisted for a week.

Line 17 simply print outs the personalized welcome message. Lines 20 through 23 create a series of hyperlinks. Notice that with the cookie method you don't need to bother with appending the current querystring to the hyperlinks. The output of Listing 11.4, when viewed through a browser, is no different than that of Listing 11.2. The output can be seen in Figure 11.3.

You can read the user's name from any other ASP page on your site by using the Request.Cookies collection to access the UserName cookie. On each page that you wanted to display your personalized greeting, you could simply add the following ASP code:

```
<%
  Dim strName
  strName = Request.Cookies("UserName")
  Response.Write "Hello " & strName & "!"
%>
```

***Caution***

Recall that some people set their browsers to not accept cookies. If such

a person were to visit your site, his name would not be persisted. The only way to persist information for these types of users is to use the querystring method discussed in the earlier section "Passing Information Through the Querystring."

## Using the Session Object

Active Server Pages comes with a built-in object to help developers maintain state on a user-by-user basis. This object is called the Session object and can be accessed through any ASP page on your Web site. The Session object can store any kind of data type, from numbers and strings to arrays and objects!

The Session is used to maintain state only for the duration of a user's visit to your Web site. When each new user comes to your site, memory on the Web server is allocated to store the Session object for that user. This memory is released if the user does not visit your Web site for a certain length of time. This time period is 10 minutes, by default but can be set to a shorter or lengthier period. We will discuss the finer details of the Session object in "The Session Object."

Each variable stored in the Session object is referred to as a session variable. You can create session variables with the following syntax:

```
Session(sessionVariableName) = value
```

where sessionVariableName is a string. The following lines of code create a number of session variables:

```
1:  Session("Today") = Date()
2:  Session("WelcomeMessage") = "Hello, world!"
3:  Session("Age") = 21
```

Line 1 creates a session variable named Today, which stores the current date. Line 2 creates a session variable named WelcomeMessage, which contains a string. Finally, line 3 stores a numeric value in the session variable named Age.

Each time you create a new variable in the Session object, that bit of memory for each unique user increases, and the new variable is stored in that memory space. These variables are persisted for each user as long as the user keeps making page requests of the Web site. Therefore, the session variables can be used to maintain state.

In both the sections "Passing Information through the Querystring" and "Using Cookies," you saw an example where the user would enter his or name. The name would then be shown in a personalized greeting on each Web page. You already know how to accomplish this with cookies and the querystring method - let's examine how you would use the Session object.

You only need to slightly modify the code in Listing 11.4 to use the Session object to maintain state. Listing 11.5 contains the new code for Welcome.asp.

**Listing 11.5 - Using the Session Object to Maintain State**

```
1:  <%@ Language=VBScript %>
2:  <% Option Explicit %>
3:  <%
4:      'Read in the Name from the form
```

```
5:      Dim strName
6:      strName = Request("Name")
7:
8:      'Write the user's name to a session variable
9:      Session("UserName") = strName
10: %>
11:
12: <HTML>
13: <BODY>
14:   Hello <%=strName%>!
15:   <P>
16:   What interests you?<BR>
17:   <LI><A HREF="sports.asp">Sports</A><BR>
18:   <LI><A HREF="politics.asp">Politics</A><BR>
19:   <LI><A HREF="fashion.asp">Fashion</A><BR>
20:   <LI><A HREF="events.asp">Current Events</A><BR>
21: </BODY>
22: </HTML>
```

To maintain state using the Session object, you need to create a session variable for each bit of information that needs to be persisted. Because you only need to save the user's name, you can use just one session variable. Listing 11.5 starts off by reading in the name the user entered in the previous form (line 6). Next, on line 9, a single session variable, named UserName, is created. This session variable is then assigned the value of strName.

Lines 12 through 21 have not changed from Listing 11.4. Line 14 simply prints out a personalized welcome message, whereas lines 17 through 20 create a series of hyperlinks. Notice that when using session variables, you don't need to bother with appending the current querystring to the hyperlinks. Again, the output of Listing 11.5, when viewed through a browser, is no different from that of Listing 11.2 or Listing 11.4. The output can be seen in Figure 11.3.

To obtain the user's name in any other ASP page on your Web site, all you have to do is read the value of your session variable. Session variables are read using the following syntax:

```
SomeVariable = Session(sessionVariableName)
```

You can display your personalized greeting on any ASP page with the following code:

```
<%
  Dim strName
  strName = Session("UserName")
  Response.Write "Hello " & strName & "!"
%>
```

The Session object uniquely identifies visitors via cookies. This means that session variables will not persist across ASP pages if the user has cookies disabled. Once again, the only sure-fire way to guarantee that state will be maintained for all your visitors is to use the querystring method. However, because the vast majority of users accept cookies, most Active Server Pages developers feel comfortable using cookies or the Session object to maintain state. A more detailed discussion is dedicated to this matter later today in the section "The Session Object."

## Using the Application Object

The Application object is another intrinsic ASP object to help maintain state. You may wonder

how the Application and Session objects differ, if they both are designed to maintain state on your Web site. Whereas the Session object is designed to maintain state on a user-by-user basis, the Application object is designed to maintain state globally, across the entire Web site. The Application object, like the Session object, can store an assortment of variables of any type. Each variable stored in the Application is referred to as an application variable.

Application variables are sometimes referred to as global variables because any user can access any application variable from any ASP page. It is important to keep in mind that there is only one instance of the Application object for your Web site. Every single user to your Web site has access to the exact same set of application variables.

Imagine that two users visit your site and that there exists an application variable named DefaultMessage, which contains the string Welcome to my Web site!. Imagine that the first user visits an ASP page that changes the application variable DefaultMessage to Hello, there!. After this change has been made, the second user visits another ASP page that displays the application variable DefaultMessage. The second user will see Hello, there! through his browser. Because there is only one instance of the Application object and because any user on any ASP page can alter application variables, when the first user changes the value of an application variable, the results are immediately noticed by the second user.

You may be wondering how the Application object can be used to maintain state. Because the Application object is global among all users, you should not try to use application variables to maintain state on a user-by-user basis. However, if you have some global piece of information that you want to save for the entire Web site, application variables are often the way to go, especially if the information changes often. We will discuss when and how to use application variables in greater detail later today in the section "The Application Object."

***Do/Don't Box***

**DON'T** store user-specific information in the Application object. If you need to store information specific to each user, use the Session object instead.

## Choosing the Approach that Works for You

So far we've examined the four methods of maintaining state. The first three approaches examined - using the querystring, using cookies, and using session variables - can be used to maintain state on a user-by-user basis. The final method discussed - using the Application object - can be used to maintain Web site-wide state.

Now that you've studied the four methods of persisting state, which method should you choose? Because each approach has strong points and weak points, the method to use depends largely on the situation. Table 11.1 presents all four methods of maintaining state and lists under what conditions to use what method.

**Table 11.1 - Approaches to Maintaining State**

| Methods | When to Use |
|---|---|
| Querystring method | When using the querystring method to persist state, make sure that each hyperlink passes on the current querystring to the next ASP page. This leads to a maintenance headache. The nice thing about the querystring method, though, is that it doesn't require the user to have cookies enabled. If you need to persist simple data types for only the duration of the user's visit to the site and it is imperative that even users who have |

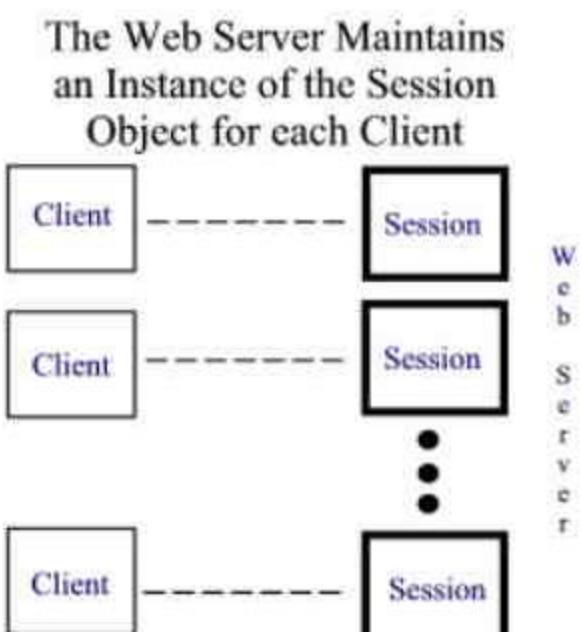
| | |
|---|---|
| | cookies disabled have their state maintained, then the querystring method is the way to go. |
| Cookies | If you need to maintain state for periods longer than the duration of the user's visit, then cookies are your only option. Cookies can only save simple data types (strings, numbers, dates) and can be rejected by the client's computer. Cookies, though, require no overhead on the Web server because they are stored on the client's machine. If performance is a big concern, or if you need to persist information for days, weeks, or months, then cookies are the best choice! |
| Session variables | If you only need to maintain state for the duration of your users' visits, using session variables may be the way to go. The Session object, unlike cookies, can store any variable type. However, the Session object resides on the Web server. If you receive many concurrent users or place large objects in the Session object, your Web server's performance will degrade. Session variables are the best choice when you need to maintain state only for the user's visit to your site. |
| Application variables | Application variables can be used to maintain information that is global to the entire Web site. The Application object should not be used to maintain state on a user-by-user basis. Application variables are a wise choice when you have some piece of information that changes often and that is global to the entire Web site. For example, if you ran a Web site that had a message board where users could post their questions, you might want to display on all your Web pages the time and date the last post was made. This information should be stored in an application variable because it changes often and is global to the Web site. |

Now that we've examined four ways to maintain state, we are going to delve into the Session and Application objects. Both of these built-in objects are useful but often misused. Because the Session and Application objects reside on the Web server, misusing these two objects can lead to a performance hit. The next two sections, "The Session Object" and "The Application Object," discuss not only how to use these objects but also how not to use these objects.

## The Session Object

The Session object is an intrinsic ASP object designed to maintain state on a user-by-user basis. Each user is assigned his own Session object. Because each user has his/her own Session object, each user's unique data can be saved. Figure 11.4 graphically shows that each user is assigned his/her own Session.

It is helpful to think of the Session object as a warehouse. When each new user arrives at the site, she receives her own warehouse. Throughout the site, any ASP page can deposit or retrieve information into a user's warehouse. Such a collection of user-specific information can prove useful.




For example, many of today's eCommerce sites have a shopping cart system, where, as you browse through the site, if you see an item you want to purchase, you can simply click it to add it to your shopping cart. When you are ready to "check out," you visit a page that summarizes your purchases, presents a total

charge, and asks for your billing and shipping information. The shopping cart is your personal warehouse, holding the information on your specific items.

When a visitor reaches your site, his "personal warehouse" is, technically, a new instance of the Session object. This object is created specifically for this particular user, serving as a vault of user-specific information. A user's Session object instance is often referred to simply as the user's Session.

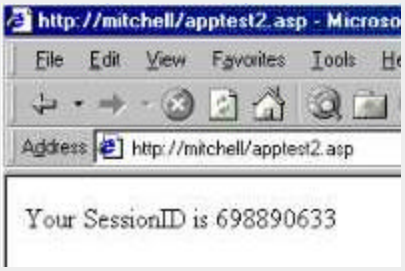
Because each user is assigned his/her own Session, each instance needs to be uniquely identifiable. A numeric ID, referred to as the SessionID, is used to identify that a particular Session belongs to a particular user. To list the SessionID for a user's Session, you can use the following syntax:

```
Session.SessionID
```

The SessionID is a numeric value, uniquely identifying each Session from other another. The following line of code would display, to each visitor, his or her unique SessionID:

```
<% Response.Write "Your SessionID is " & Session.SessionID %>
```

**Figure 11.5 shows the output of the preceding line of code.**




***Caution***

The SessionID is guaranteed to be unique for each separate session as long as the Web server is running. However, if the Web server is restarted, new SessionIDs might be duplicates of older SessionIDs. For this reason, it is unwise to use the SessionID as a unique identifier in a database.

The SessionID is stored in two locations: the Web server and the client. Each Session that is managed by the Web server contains its own SessionID. This SessionID is also stored on the client's computer, in the form of a cookie. Because the SessionID is saved on both the client and the Web server, the Web server can establish what Sessions belong to what clients.

Imagine that you have an ASP page that contains the following simple line of code:

```
<% Response.Write "Your name is " & Session("Name") %>
```

The preceding line would display Your name is, followed by the value in the session variable Name. What, exactly, happens when a user visits this page? Because each visitor can have a different value for the session variable Name, how is the correct value selected? Recall from yesterday's lesson that each time a Web page is requested from the Web server, a number of HTTP headers are sent. One of the HTTP headers is the Cookie header, which contains all the cookies on the client's computer that were created by the Web site. If session variables are being

used on your Web site, one of these cookies contains the SessionID associated with a particular Session on the Web server. This cookie is matched up with the correct Session, and the Name variable is displayed.

Using cookies to associate a particular client with a particular Session has its drawbacks. What happens if the user has set up his browser to not accept cookies? If this is the case, this user cannot have his own "personal warehouse," and state will not be persisted for this user. Although the majority of Web surfers today have cookies enabled, there is no guarantee that all your visitors accept cookies. There is, however, a way to use session variables with users who do not accept cookies. This workaround is discussed later today in the section "Session Variables Without Cookies."

Because each user's Session is created and stored in the Web server's memory, it is important to have this memory freed up when the visitor is no longer at your site. Due to the client-server model, you cannot determine when someone leaves your site. However, you can keep track of the last time a specific user accessed your site. If a specific user has not accessed your site for a particular amount of time, her Session is freed from the Web server's memory. The amount of time that passes before a user's Session is freed is referred to as the session timeout.

***Caution***

IIS 5.0 is scheduled to have the default session timeout to 10 minutes. However, with Windows 2000 Release Candidate 2, the default session timeout is still at 20 minutes. With IIS 4.0, the default session timeout is, and has always been, 20 minutes.

You can set the session timeout by using the Timeout property of the Session object. You can assign the Timeout property a numeric value, representing the number of minutes before a user's Session times out and destroys itself. For example, if you want to set the session timeout to 5 minutes, use the following line of code:

```
'The Session object is set to timeout in five minutes!
Session.Timeout = 5
```

If you want to destroy the user's Session explicitly, before the session timeout occurs, use the Abandon method. Some personalized sites have a LogOut button available that, when clicked, removes any saved information, such as cookies and session variables. The LogOut button, when clicked, should display a LogOut message and call the Session object's Abandon method. Listing 11.6 shows the code for LogOut.asp, which simply displays a short message informing the user that she has been logged out and destroys the user's Session through a call to Session.Abandon.

**Listing 11.6 - Using Session.Abandon to Destroy the User's Session Object**

```
1:  <%@ Language=VBScript %>
2:  <% Option Explicit %>
3:  <%
4:      'Destroy the user's session
5:      Session.Abandon
6:  %>
7:
8:  <HTML>
9:  <BODY>
10:   You have been logged out.  Your Session variables have
```

```
11:   been destroyed!
12: </BODY>
13: </HTML>
```

Because each user has their own instance of a Session object, the memory requirements on your Web server increase as the number of concurrent users on your Web site increases. Therefore, it helps to free the memory associated with a user's Session as soon as possible. Listing 11.6 demonstrates how to remove a user's Session object by using the Abandon method of the Session object (line 5).

When each new user visits your site, he is given a "personal warehouse," into which your ASP pages can store and retrieve user-specific information. The Session object acts as the warehouse itself. Each warehouse contains its own unique, numeric ID, called the SessionID. After a warehouse has not been accessed for a set length of time, the warehouse is demolished, freeing up real estate for another warehouse. This length of time is the session timeout and can be accessed via the Timeout property of the Session object. Finally, use the Abandon method if you need to explicitly destroy a Session prematurely.

Now that you've examined the role of the Session object, it's time to discuss how to write and read the variables stored inside the Session object. These session variables are responsible for saving the user's information, thereby maintaining state across the Web application. The next section, "Using Session Variables," discusses the intricacies of session variables.

## Using Session Variables

As discussed in the previous section, an instance of the Session object serves as a warehouse for a specific user's information. What use is a warehouse, though, if you put nothing inside it? If you have a user-specific piece of information that you need to persist, you need to save this information to a variable and put that variable into the user's Session. Such variables are referred to as session variables.

To write a value to a session variable, use the following syntax:

```
Session(sessionVariableName) = Value
```

This, essentially, stores a variable into a user's personal warehouse. A session variable can be read by using:

```
Value = Session(sessionVariableName)
```

Imagine that you were creating a site, and at the top of each Web page you wanted to put a quote of the day. It would be nice to present the user with an option to hide the quote. Listing 11.7 shows the source code for an ASP page that shows both the quote of the day and an option to hide the quote.

**Listing 11.7 - Hiding the Quote of the Day**

```
1:  <%@ Language=VBScript %>
2:  <% Option Explicit %>
3:  <%
4:      'Do we want to show the quote of the day?
5:      If Session("ShowQuote") = False then
6:          'The user doesn't want to see the quote
```


```
7:        'Show them an option to see the quote again
8:  %>
9:        <A HREF="/scripts/ShowQuote.asp">Show Quote of the Day</A>
10: <%
11:     Else
12:       'The user wants to see the quote, so show it!
13: %>
14:       A stitch in time saves nine.<BR>
15:       <A HREF="/scripts/HideQuote.asp">Hide Quote</A>
16: <%
17:     End If
18: %>
19:
20:   <P><HR><P>
21:   <I>Blah blah blah blah...</I>
```

The code in Listing 11.7 first determines whether the user is interested in seeing the quote of the day. In line 5, an If statement determines whether the session variable ShowQuote is False. If ShowQuote is False, you don't want to show the quote of the day. Line 9 provides a hyperlink that, when clicked, allows users to see the quotes of the day. If, on the other hand, ShowQuote is not False, the code after the Else on line 11 is executed. Line 14 displays a famous quote from Ben Franklin, and line 15 provides a link that, when clicked, turns off the quote of the day. Figure 11.6 shows the output of Listing 11.7 when a user visits the site for the first time.

In Figure 11.6, the quote of the day isn't shown on the user's first visit. This is because the session variable ShowQuote has not yet been created in the user's Session. Because ShowQuote doesn't yet exist, when you ask for ShowQuote in line 5 of Listing 11.7, you are returned an empty string, which evaluates to False. There's nothing wrong with this unless you want to have the quote of the day shown by default. Later in the section, "Initializing Application and Session Variables," we will discuss how to have session variables created automatically upon a new user's visit.

Note Figure 11.6 and line 9 in Listing 11.7. If the ShowQuote session variable is False, you need to provide the user with a link to start showing the quotes of the day again. This link, if clicked, will take the user to ShowQuote.asp. This ASP page needs to "turn on" the quote of the day. This is done by setting the ShowQuote session variable to True. After ShowQuote has been set to True, ShowQuote.asp needs to redirect the user back to the page he came from. Listing 11.8 presents the code for ShowQuote.asp.

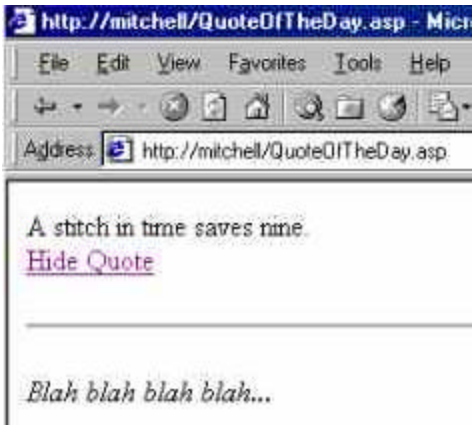
Listing 11.8 - ShowQuote.asp "Turning On" the Quote of the Day

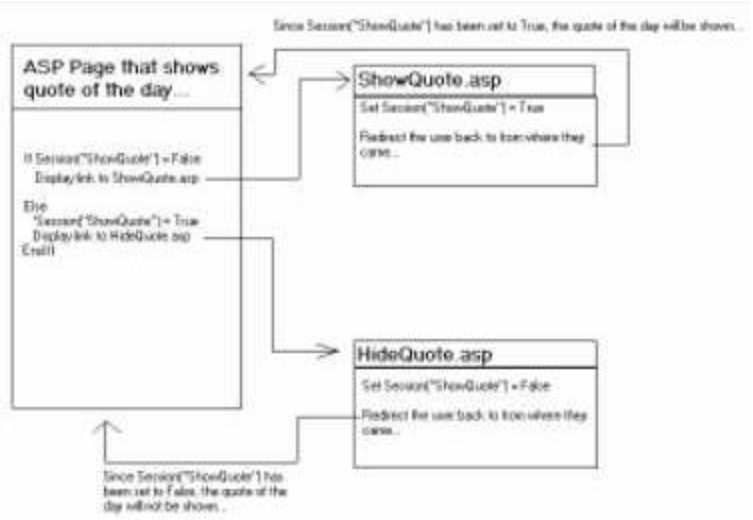
```
1:  <%@ Language=VBScript %>
2:  <% Option Explicit %>
3:  <%
4:      'Set ShowQuote to True
5:      Session("ShowQuote") = True
6:
7:      'Send the user back to the page from which they came
```

```
8:      Response.Redirect Request.ServerVariables("HTTP_REFERER")
9:  %>
```

ShowQuote.asp needs to do two things: set the ShowQuote session variable to True, thereby turning on the quote of the day; and redirect the user back to the page from which he came. The first task is accomplished on line 5. The second task is completed on line 8, using the Redirect method of the Response object. The user is sent to the URL specified by the Referer HTTP header, which is the URL of the page he came from. The Request.ServerVariables collection and the HTTP headers were discussed on Day 10.

By clicking the Show Quote of the Day hyperlink in Figure 11.6 (created on line 9 in Listing 11.7), ShowQuote.asp was loaded. ShowQuote.asp then set the ShowQuote session variable to True and sent the user back to the Web page he came from. When the user arrives back at the page he started on, the quote is now showing. Figure 11.7 shows the results of the code in Listing 11.7 after the user clicks the Show Quote of the Day hyperlink. Figure 11.8 presents a diagram of the steps taken to show and hide the quote of the day.




When the quote of the day is shown, not only do you need to display the quote of the day, you also need to provide the user with an option to turn off the quote of the day. Line 15 in Listing 11.7 creates a link to HideQuote.asp, which needs to do nearly the exact same thing as ShowQuote.asp. HideQuote.asp should set the ShowQuote session variable to False and then redirect the user back to the page he came from. The code for HideQuote.asp can be found in

Listing 11.8. After turning off the quote of the day, the output is the same as when you initially visited. You no longer see a quote but instead see the Show Quote of the Day hyperlink. This output was shown previously in Figure 11.6.

The Session object can be used to store any type of variable. With cookies, you are restricted to only saving simple data types on the client's computer. The Session object, however, can be used to store arrays. To show this, create an ASP page named CreateSessionArray.asp and enter the code shown in Listing 11.9.

**Listing 11.9 - Using the Session Object to Store Arrays**

```
1:  <%@ Language=VBScript %>
2:  <% Option Explicit %>
3:  <%
4:      'Create an array
5:      Dim aSentence(4)
6:      aSentence(0) = "I "
7:      aSentence(1) = "like "
8:      aSentence(2) = "Active "
9:      aSentence(3) = "Server "
10:     aSentence(4) = "Pages!"
11:
12:     Dim iLoop
13:     For iLoop = LBound(aSentence) to UBound(aSentence)
14:        Response.Write aSentence(iLoop)
15:     Next
16:
17:     'Store the array in the Session object
18:     Session("Sentence") = aSentence
19: %>
```

Listing 11.9 starts out by creating an array, aSentence (line 5). This array contains five elements, 0 through 4. Lines 6 through 10 set the values of each of these five array elements. Next, the contents of the array are printed out. Line 13 uses the LBound and UBound functions to iterate through the array aSentence one element at a time. Recall from Day 5, "VBScript's Built-in Functions," that the LBound function returns the starting index of an array, while UBound returns the ending index, or upper bound, of an array. UBound and LBound can be used in conjunction with a For loop to iterate through an array. Refer to Appendix B for a more thorough description of LBound and UBound.

Next, line 14 prints out the current element on each iteration of the loop. Finally, line 18 creates a session variable named Sentence and sets it equal to the array aSentence. Note how simple it is to set a session variable equal to an array. The output of Listing 11.9 is I like Active Server Pages!

Now that you have your array in a session variable, you can access the contents of that array on another ASP page. Listing 11.10 shows the code for PrintSessionArray.asp, which, as the name suggests, prints out the contents of the Sentence session variable array.

**Listing 11.10 - Displaying the Contents of the Session Variable Array**

```
1:  <%@ Language=VBScript %>
2:  <% Option Explicit %>
```

```
3:  <%
4:    Dim iLoop
5:
6:    'First, make sure that the Session variable
7:    'Sentence is a valid array
8:    If IsArray(Session("Sentence")) then
9:      'Print out each element in the array
10:     For iLoop = LBound(Session("Sentence")) to UBound(Session("Sentence"))
11:       Response.Write Session("Sentence")(iLoop)
12:     Next
13:   Else
14:     'If Session("Sentence") is not a valid array,
15:     'display an error message
16:     Response.Write "No sentence to process!"
17:   End If
18: %>
```

The code in Listing 11.10 displays the contents of the Sentence session variable array, if it exists. Before attempting to read from Session("Sentence"), line 8 checks to make sure that Sentence is a valid array by using the IsArray function. If the user had not visited CreateSessionArray.asp prior to visiting PrintSessionArray.asp, the session variable Sentence would not have existed and would have returned an empty string. Passing LBound an empty string, as opposed to an array, would have generated an error. Therefore, it is essential that you first check to make sure that Sentence is a valid array.

If Sentence is an array, line 10 loops through each element of the array, displaying the contents of each element (line 11), similar to lines 13 through 15 in Listing 11.9. The syntax for referring to a specific element in a session variable array may seem a bit confusing (line 11). With an array, you refer to a specific variable with ArrayName(index). When you store an array in the Session object, the ArrayName is Session(sessionVariableName). Therefore, to read an element from a session variable array, the syntax is Session(sessionVariableName) (index).

If you were to leave out the If statement starting on line 8, an error would occur when you asked to compute LBound(Session("Sentence")). The error message displayed would read:

```
Microsoft VBScript runtime error '800a000d'

Type mismatch: 'LBound'

/PrintSessionArray.asp, line 9
```

Because the If statement is on line 8, if the Sentence session variable array hasn't been created, IsArray(Session("Sentence")) will return False, and the code following the Else statement on line 13 will execute. Line 16 simply displays an error message, indicating to the user that there was no sentence to display. This type of error message is preferred to the VBScript runtime error message shown previously.

The Session object can also contain session variable objects, although it is vital that, as a developer, you use prudence when placing objects into the Session object. Because each user has his own Session, if your Web site has many concurrent users, Session scoped objects can quickly eat the Web server's memory. Objects should only be placed in the Session on small Internet or intranet sites, where the number of concurrent users is guaranteed to be low. The ramifications of using session variable objects are discussed in detail later today in the section "Pitfalls of Session Variables."

When storing an object in the Session, the Set keyword is also used, as in the following example:

```
Set Session(sessionVariableName) = ObjectInstance
```

The following lines of code show how to set a session variable to an instance of an object:

```
1: Set Session("MyDict") = Server.CreateObject("Scripting.Dictionary")
2: Set Session("Connection") = Server.CreateObject("ADODB.Connection")
3: Set Session("CustomObject") = Server.CreateObject("My.Object")
```

Line 1 creates a session variable named MyDict that is an instance of the Scripting.Dictionary object. Line 2 creates an instance of the ADODB.Connection object, storing it in the session variable Connection. Similarly, line 3 creates an instance of an object named My.Object and assigns it to the session variable CustomObject. Although these three examples create instances of objects as session variables, it is not wise to place such objects in the Session, due to performance and memory concerns. This topic is addressed later in the section "Pitfalls of Session Variables."

The Session object provides two collections containing the session variables.

- Contents contains the non-object session variables.
- StaticObjects contains session variable objects.

The code in Listing 11.11 displays all the non-object session variables in a user's Session.

**Listing 11.11 - The Contents Collection Contains All Non-Object Session Variables**

```
1:  <%@ Language=VBScript %>
2:  <% Option Explicit %>
3:  <%
4:    'How many session variables are there?
5:    Response.Write "There are " & Session.Contents.Count & _
6:              " Session variables<P>"
7:
8:    Dim strName, iLoop
9:    'Use a For Each ... Next to loop through the entire collection
10:   For Each strName in Session.Contents
11:     'Is this session variable an array?
12:     If IsArray(Session(strName)) then
13:       'If it is an array, loop through each element one at a time
14:       For iLoop = LBound(Session(strName)) to UBound(Session(strName))
15:         Response.Write strName & "(" & iLoop & ") - " & _
16:             Session(strName)(iLoop) & "<BR>"
17:       Next
18:     Else
19:       'We aren't dealing with an array, so just display the variable
20:       Response.Write strName & " - " & Session.Contents(strName) & "<BR>"
21:     End If
22:   Next
23: %>
```


Listing 11.11 displays all the user's session variables and values. Line 5 uses the Count property of the Contents collection to display the number of session variables. Line 10 starts the For Each ... Next loop through the Contents collection. Because the Contents collection contains all non-object session variables, it also contains session variable arrays. For this reason, before you display a session variable, you need to first determine whether it is an array. Line 12 accomplishes this, using the IsArray function. This is identical to the syntax used on line 8 in Listing 11.10 to determine whether a session variable is an array.

If the session variable is an array, lines 14 through 17 are executed. Line 14 is a For loop, which iterates through each element in the session variable array. The contents of each session variable array element are displayed (line 16). If, however, the session variable is not an array, the Else block starting on line 18 is executed. The session variable's name and value are then displayed (line 20). Figure 11.9 shows an example of the output of Listing 11.11.

Before running this page, another session variable array was created, Message. As you can see, there are four session variables, two of which are arrays, and two of which are not. These session variables, except for Message, were all created with earlier examples in today's lesson. Specifically, Name was created in Listing 11.5, Age in an example in the section "Using the Session Object," and Sentence in Listing 11.9.

The other session variable collection, StaticObjects, cannot be used to list session variable objects created in an ASP page. Objects created in this manner are referred to as dynamic objects because they are created on-the-fly, only when a user visits an ASP page that contains code similar to the following:

```
Set Session(sessionVariableName) = ObjectInstance
```

Rather, the StaticObjects collection, as its name implies, can only be used to iterate through static objects in the Session object. Static objects are objects that are created for each user when she first visits the site. We will discuss how to create such objects later in today's lesson, in the section "Initializing Application and Session Variables." The syntax for iterating through the StaticObjects collection is straightforward. You can access each static object in a user's Session by issuing a For Each[el]Next construct. The following example lists all the session variable names that are holding a static object instance:

```
Dim strName
For Each strName in Session.StaticObjects
   Response.Write strName & "<BR>"
Next
```

Using these two collections, Contents and StaticObjects, you can loop through the session variables stored in a user's Session (except for session variable objects created on an ASP page). When using the Session object to maintain state on your Web site, it helps to think of the Session object itself as a warehouse and session variables as the goods inside the warehouse. Because each user has their own Session, each user can have unique session variables values. This allows for user-specific state to be maintained across a Web site.

## Pitfalls of Session Variables

When using session variables in your Web application, try to avoid a few common pitfalls:

- Pitfall 1 - Placing objects in a user's Session
- Pitfall 2 - Setting the Timeout property to a non-optimal value
- Pitfall 3 - Creating unnecessary session variables

Because each user is assigned their own instance of the Session object, the more concurrent users visiting your site, the more Session instances are needed. If you start placing large objects into each user's Session, each Session object will grow, requiring more of the Web server's memory. As more and larger Session objects are instantiated, the slower your Web server will become. For this reason, it is wise to keep objects out of the Session.

Sometimes an object needs to be used on nearly every ASP page. Some developers place large objects into the user's Session, reasoning that creating the object once will lead to a performance boost over creating the object every time a page is loaded. Don't fall into this trap! Microsoft recommends that you wait to create objects as late as possible and destroy them as soon as possible. This technique leads to the best performance. Again, resist the urge to place objects in your users' Sessions.

Another common pitfall that developers experience when using the Session object is setting the Timeout property to an optimal value. Recall that after a particular user has not accessed your Web site for a specific duration, the user's Session will be freed from memory. This length of time before the Session terminates itself is referred to as the session timeout and can be set using the Session object's Timeout property. Do not set this value too high. Imagine, for a moment, that you set the Session's Timeout property to 120 minutes. When a new visitor comes, a Session is created for that user. If the user browses your site for a while and then leaves, the memory resources allocated for that user will remain present for another 2 hours.

Imagine that, on average, every half-hour, 100 users come to your site as 100 users leave. The net effect is an average of 100 users on your site at any given time. You would expect to have decent performance, but a poorly set Timeout would thrash your Web server's memory. When the first 100 users come, 100 Sessions are created. When they leave half an hour later and a new 100 users come, another 100 Sessions are created. However, the first 100 Sessions aren't freed from the Web server's memory and won't be for another 2 hours. Your Web server will soon have 500 Session object instances in memory, even though there are only 100 concurrent users on your site. Table 11.2 displays the growth of user Sessions on a Web site with its session timeout set to 120 minutes.

**Table 11.2 - A High Timeout Property Wastes Memory**

| Time | Concurrent Users | Total Sessions | Explanation |
|---|---|---|---|
| 0:00 | 100 | 100 | At time 0, 100 new users come to your site. Hence, |

| | | | |
|---|---|---|---|
| | | | 100 Sessions are created. |
| 0:30 | 100 | 200 | Your 100 initial visitors have left (although their Sessions won't be removed until the start of time 2:30. 100 new visitors have come to the site, adding another 100 Sessions. |
| 1:00 | 100 | 300 | Another 100 new visitors come, adding a new 100 Sessions. 200 Sessions exist from the first two groups of 100 visitors. |
| 1:30 | 100 | 400 | Another 100 new visitors replaced the 100 from the half-hour period before. |
| 2:00 | 100 | 500 | Yet another 100 visitors arrived. You now have 500 Sessions for only 100 concurrent users! What a waste of memory! |
| 2:30 | 100 | 500 | A new 100 users come, but the first 100 visitors from time 0 have their 100 Sessions removed. As long as you keep receiving 100 new users every half hour, you will remain at 500 total Sessions. |
| 3:00 | 100 | 500 | ... |

Be careful not to set your Timeout property too low, either. Imagine that you decided to set the session timeout to 1 minute. Although this will surely keep the Web server's performance up to par, your visitors will find your site annoying to use because, if they don't visit a page within 1 minute, their Session will have expired. When the Session is lost, state is lost because the Session is used to maintain state.

If you were creating an eCommerce site that used session variables to maintain the state of a shopping cart, setting the Timeout property to 1 minute would surely aggravate your users. If a user placed a few items in his cart and then, perhaps, sent an email or performed some other activity that kept him from browsing your Web site for longer than 1 minute, when the user went back to continue shopping, his cart would be empty.

For these reasons, choosing an appropriate session timeout value is important. IIS 5.0 should set the default session timeout to 10 minutes, which is a good value for most Web applications. (Note that the beta version of IIS 5.0 shipped with Windows 2000 Release Candidate 2 has the session timeout set to 20 minutes.) If, however, you know that your users will spend long periods of time viewing your Web pages, it would make more sense to up the session timeout. If you expected several hundred concurrent users, it might be beneficial to lower your session timeout from 10 minutes. The best way to determine what timeout value works best for your particular Web site is to simply try various timeout values during high loads and note the performance of your Web server.

The Session is easy to use. It's simple to use session variables anytime you need to maintain state. They are easy to program and work across all ASP pages. This leads us to the third pitfall - creating unneeded session variables. It is easy to become a bit overzealous when using session variables. If you are using session variables to store information that is not user specific, use the Application object instead. For example, many Web sites have a standard navigation footer at the bottom of all of their Web pages. This navigation footer displays a list of links to the site's various sections. When you use the Session object to store non-user-specific values, you are wasting your Web server's resources. Why should each user have the text navigation footer stored in his Session, when every user sees the same footer?

***Note***

If you want to have a navigation footer on your Web site, or any other kind of static information that needs to appear on all your ASP pages, optimal performance can be achieved through the use of include files.

Before deciding to use the Session object, carefully consider a few issues. Do you need to maintain state via the Session object? Could cookies be used instead? Because the Session object carries with it potentially grave performance consequences, it is important to ask yourself the following questions before deciding to use the Session object:

- How many concurrent users do I expect on my Web site at any given time?
- How many session variables do I plan on using?
- Can I maintain state using an alternative approach?

If you expect many concurrent users, you should not use any session variables. The number of concurrent users that your Web site can handle before showing signs of performance degradation depends on the hardware you use to run your Web site. When I develop ASP applications, if the number of concurrent users is expected to be more than 50, I try my best to refrain from using session variables.

## Session Variables Without Cookies

Because each user has his own Session, the Web server needs some way to tie a particular user to a particular Session object instance. This is done through a unique SessionID, which links together the Session and the user. The SessionID is found both on the particular Session object instance, and on the client's computer, in the form of a cookie. When the user requests an ASP page that uses session variables, the user's session cookie can be read and matched up to an existing Session in memory.

What happens, though, if the user has configured her browser not to accept cookies? Because the Web server cannot associate the user with a particular Session, all session variables accessed by this user will return empty strings. You will not be able to maintain state for this user, or any other user who does not accept cookies.

To remedy this problem, Microsoft has created an ISAPI filter that will simulate cookies for users who have configured their browsers not to accept cookies.

An *ISAPI filter* is a small, low-level program that enhances a Web server by providing additional functionality. ISAPI is an acronym for Internet Server Application Programming Interface.

This ISAPI filter, named Cookie Munger, can be downloaded for free from Microsoft's Web site (http://www.microsoft.com). By using this ISAPI filter on your Web server, those who do not accept cookies can still utilize session variables.

When a user who accepts cookies visits your site, Cookie Munger does nothing. However, when a user who does not accept cookies visits your site, the following transactions occur:

1. When a page is requested, Cookie Munger generates a SessionID for that particular client.
2. The ASP page takes that SessionID and can map it to a particular Session object instance.
3. When an ASP page is sent to a client that does not accept cookies, Cookie Munger parses the HTML for hyperlinks. At the end of the hyperlink URL, Cookie Munger adds the synthesized SessionID for the client.
4. When a client requests a URL that has the SessionID appended to the end, Cookie

Munger removes the appended SessionID, calls the ASP page requested, and passes the SessionID to the ASP page.

Through these four steps, Cookie Munger allows state to be maintained via session variables for those who do not accept cookies.

If Cookie Munger can be used to allow all visitors to use session variables, you may be wondering why it isn't installed with IIS by default. Because Cookie Munger has to process every ASP page being requested and process every outgoing ASP response, this can place a tremendous burden on your Web server, especially for high volume sites. Cookie Munger should only be used for low volume sites where it is necessary for all users to be able to store and retrieve session variables. For more information on Cookie Munger, visit Microsoft's Web site.