State Management

Objectives

- Learn what state is and how you use it.
- Learn different methods of state management.
- Learn the advantages and disadvantages of each method.

Investigating State Management

Web pages are, by definition, stateless. This means that as you move from page to page, the data from each page is automatically discarded. Because you might need data from one of these previous pages, you need to store that data (or state) as you move from one page to another. There are many techniques you can use to manage maintaining state.

In this chapter, you will learn how to use the various state management techniques in your .NET Web applications. You will see how to use the Session object, StateBag objects, and the .NET Framework to help you manage state across a Web farm. You will also learn the advantages and disadvantages of each of these techniques.

State Management Techniques

You can manage state in your .NET Web applications using tools available in Internet Information Server and within the .NET Framework itself. Many of the IIS tools have not changed from Active Server Pages, but have been updated to be more scalable. The .NET Framework has also added some additional tools that you will learn about in this chapter. Some of the ways you can use to manage state include the following:

- Using Session and Application objects to cache information.
- Using Memory and Disk Cookies to preserve information.
- Using hidden input fields or the URL-embedded information to pass information from one page to another.
- Using the ViewState property of the page to set and retrieve information stored in a StateBag object.
- Using SQL Server to store state information.

In each case, your goal is to take data from the current page, and have that data available, on demand, either when you redisplay the current page, or as reference when displaying a different page.

Session and Application Objects

The Session and Application objects allow you to store name/value pairs of values. The Session object stores values between Web pages, maintained for each user. Use the Application object to store data that you want to make available across the whole site, for all users. You might use a Session variable to keep track of user identity, as the user navigates the pages of your application. You might use an Application variable to keep track of the number of times a page has been hit. Both of these objects, Session and Application, store the state information on the Web server.

Cookies

Some developers use memory cookies to reduce the amount of resources stored on the server. Memory cookies pass back and forth from the browser to the server, where they're maintained as the user moves from page to page on a site. When the user closes the browser, the cookie is released from memory.

Permanent (or “hard”) cookies allow you to save data on the users’ local computers. If you know that a user will visit your site multiple times, it makes sense to save state information locally, if the user has allowed this option. The Web server sends permanent cookies to the user’s browser, and the browser stores this data on the user’s hard disk. The browser can retrieve this data from disk when the user revisits the Web site. The cookie is again passed from the browser to the server for each page.

Hidden Input Fields

Hidden input fields can be used to pass data from one page to another. When the user clicks on a Submit button, the form posts the data the user filled in, along with any hidden input fields as well. Create a hidden input field using a normal HTML tag, like this:

```
<input type="hidden" value="10" name="txtRate">
```

You can store data in a hidden input field to help maintain state from one page to another. In the example above, the value 10 is stored in the hidden field named *txtRate*.

Embedded URL Parameters

You may pass values on the URL by using value/data pairs. For example, you may call an ASP.NET page like this

```
Main.aspx?CMD=1&ID=29398
```

When you navigate to Main.aspx, you pass two variables in the URL: CMD and ID. You can pass quite a bit of information on this URL, so this is a reasonably effective method of maintaining state.

> WARNING! Passing parameter information in the URL displays the values for the user to see—that is, if it's in the URL, it gets displayed in the browser. Don't plan on passing sensitive data in the URL.

StateBag Class

The .NET Framework provides a StateBag class that allows you to preserve view state while you are working within one page. If the user will be posting data back to the server while staying on the same page, you can use a StateBag object to hold multiple intermediate values for this page. You might use this technique when the user must choose a value from one combo box on a page, and then, you want to fill the items in another combo box based on the selected item in the first combo box.

SQL Server

In addition to all the other techniques available, you can also store state information in a SQL Server database. If you need to maintain a lot of data between pages, this may be your best bet. If you're gathering a large amount of data about the user over several Web pages, you might consider storing this data in SQL Server. There are two techniques for using SQL Server: you can construct the session data and insert the data into the database yourself, or you can let the .NET Framework handle the job for you automatically.

Web Forms Manage the View State

Without some means of preserving the state between round trips to the Web server, data that you've entered onto a Web page would disappear as you post back a page. If you click a button, or take any other action that causes a post back to the server, without some help from ASP.NET, your data would be lost as the server resends the current page. ASP.NET takes care of managing this round-trip postback state for you with no extra coding on your part.

ASP.NET keeps track of this data by adding a hidden input control on each page. This control (always named __VIEWSTATE) maintains all the information from all the controls on the page that have their EnableViewState property set to True. You can see this hidden input variable if you view the source for a Web page from your browser. ASP.NET compresses and encrypts the state information, so you won't be able to discern any of its contents.

TIP To follow along with the code in this chapter, you can load the **StateMgmt.sln** solution from the **Ch11-ASPNET-StateMgmt** or **Ch11-ASPNET-StateMgmt-CS** folder.

Using the Session Object

The ViewState information is great for maintaining state across postbacks on a single page, but as you develop Web applications in .NET, you will find that there is a definite need to keep track of data from one page to another, not just on a single page. That's when you need a Session Object.

Each time a new user comes into your site, IIS creates a new Session object. IIS automatically assigns a unique number to this session and places it into the Session object's SessionID property. You can use this Session ID to uniquely identify a particular user and create your own session variables to hold state as long as that user's session is active. IIS sends this Session ID to the browser as a dynamic cookie; each time the browser navigates to any page on the site, this cookie is sent to the server via the HTTP header.

NOTE In order to make use of the Session object, your users must accept cookies. Although most users don't turn off this capability in their browsers, some do. You'll see, later in this chapter, how you can eliminate cookies when using Session variables.

SessionID Longevity

The Session object for a particular user does not live indefinitely. The ID only lives until one of the following conditions becomes true:

- Your code calls the Session.Abandon method .

- The Session object times out. The default timeout value is 20 minutes, and so if the user doesn't submit a request back to the site in that time, theSession object will be released. You can change this timeout value.
- The IIS Service shuts down

Creating Session Variables

You can create your own session variables and assign values to these variables, using code like this:

VB.NET

```
Session("Email") = "JohnDoe@yahoo.com"
```

C#

```
Session["Email"] = "JohnDoe@yahoo.com";
```

This code creates a new Session variable, named Email, which is unique for this user. Once you have created Session variables, the values stay around until you explicitly set them equal to **Nothing/null**, or until the session is destroyed, as explained in the previous section.

To retrieve the value of a Session variable, use code like this:

VB.NET

```
txtPassword.Text = Session("Email").ToString()
```

C#

```
txtPassword.Text = Session["Email"].ToString()
```

TIP	The value returned when you retrieve a Session variable is an Object type—you'll need to convert the value as required by your application, as in the previous code example.

Issues with Session Objects

There are several issues that you'll need to keep in mind when using the Session object to maintain your application's state.

Memory on the Server is Limited

Each Session variable you create consumes memory on the server. Although the memory needed for one variable may not seem like much, when you multiply all of the data in all your Session variables by the number of users that might hit your site at one time, your Session variables could be eating up quite a large amount of memory.

You'll Need Some Security

Once a session begins, the user works with the same SessionID value until the session times out. If a hacker was able to retrieve this number, the hacker could potentially take over the user's session. If you were storing credit card information into a Session variable, it's possible that the hacker could see this information, causing a security leak. Although this is unlikely, it isn't impossible. To get around this problem, you might wish to employ the Secure Sockets Layer (SSL) when working with sensitive information.

Session Variables Don't Scale

A Web farm is a group of Web servers working together to service a particular Web site. Each time the user hits a page on your Web site, an IP router determines which machine is not being used too heavily and routes the request for the page to that machine. If the user is routed to a different machine than the one on which the Session object was created, the new server has no way to retrieve that state, and all that user's data is lost.

To alleviate the problem of different machines serving different pages, you can set values in the application's Web.Config file that specify the name of the machine that stores all session variables. This way, you can dedicate one machine that will do nothing but manage session variables for your site.

Although this helps with the problem of a Web farm, it introduces problems of its own. For example, this "session state" machine is one more machine that you need to keep up and running twenty-four hours a day. You might also need some redundancy for this machine, to provide continuity while you perform maintenance tasks. In addition, the one main machine could cause performance problems, as one machine could be a bottleneck when many machines have to cross the network to get at the data. In any case, we'll discuss Web.Config settings later in the chapter.

Not All Users Accept Memory Cookies

If you have users that will not accept memory cookies in their browsers, you need to set a specific configuration option in .NET to make the session variables work correctly. In the past, memory cookies were *required* on the client browser to make Active Server Pages hold state. The .NET Framework can preserve session state without memory cookies. See the section "Cookieless Sessions" for information on setting this up.

Turning Off Cookies Page By Page

Whenever an ASPX page is hit, the .NET runtime will, by default, attempt to generate a cookie and send it to the browser. If you know that a page will not use any session state, you can set the **EnableSessionState** page directive to False to turn off this automatic generation, on a page-by-page basis. At the top of every ASPX page, you will find a Page Directive that looks like the following:

```
<%@ Page Language="*" AutoEventWireup="false"
 Codebehind="SessionTestError.vb"
 Inherits="DotNetStateMgmt.SessionTestError"
 EnableSessionState="False"
%>
```

Add the EnableSessionState directive and no cookie will be generated for this page.

WARNING:	If you attempt to use a Session object on a page that has the EnableSessionState Page Directive set to False, you will receive a runtime error.

Using Cookies

If you do not wish to store the state of your application on the Web server, you can send the state out to the client's browser. You do this by using **cookies**.

Memory Cookies

You will most likely use a memory cookie, as these cookies are destroyed when the user closes their browser. To create a memory cookie, you use the Cookies property of the Response object, as shown in the following code. In this example, the code creates a memory cookie named "Email", and assigns the email address into the cookie.

```
VB.NET
Response.Cookies("Email").Value = "JohnDoe@yahoo.com"

C#
Response.Cookies["Email"].Value = "JohnDoe@yahoo.com";
```

Another method of creating a new cookie is to use the Add method of the Cookies collection, which is a property of the Response object. You can create a new System.Web.HttpCookie object, and pass in the name and the value to the constructor for that object.

```
VB.NET
Response.Cookies.Add(New _
 System.Web.HttpCookie("Email", "JohnDoe@yahoo.com"))

C#
Response.Cookies.Add(new _
 System.Web.HttpCookie("Email", "JohnDoe@yahoo.com"));
```

Either method shown above will set a memory cookie into the user's browser for you. This assumes that the user allows memory cookies into their browser.

To retrieve a memory cookie on any subsequent page, or even on the same page, use the Request object. You should first check to see if that memory cookie has been created yet. If you try to access the Cookies collection and pass in the name of a variable that has not yet been created, you will receive a runtime error.

```
VB.NET
If Not Request.Cookies("Email") Is Nothing Then
   txtEmail.Text = Request.Cookies("Email").Value
End If

C#
if (Request.Cookies["Email"] != null)
   txtEmail.Text = Request.Cookies["Email"].Value;
```

The sample code checks to see if the `Request.Cookies(“Email”)` object is Nothing. If so, you won't be able to do anything with that cookie. If the object has something in it, you can retrieve the value and assign it to a text box on the form.

Permanent Cookies

If you wish to store a cookie to a user's hard disk, you need to set the Expires property on that cookie. The code below shows an example of how you create a cookie called EmailPerm and set the expiration date for 30 days in the future.

```
VB.NET
Response.Cookies("EmailPerm").Value = txtEmail.Text
Response.Cookies("EmailPerm").Expires = _
   DateTime.Today.AddDays(30)

C#
Response.Cookies["EmailPerm"].Value = txtEmail.Text;
Response.Cookies["EmailPerm"].Expires = _
  DateTime.Today.AddDays(30);
```

By setting the Expires property to a date in the future, the browser stores this cookie into a folder on the user's hard disk. Users may set their browsers to only allow memory cookies and not allow permanent cookies. If the active browser won't allow permanent cookies, the data can't be stored. You will not receive an error that the cookie could not be stored, but you won't get the data back when you request the cookie. If you wish to remove a permanent cookie, set the Expires property to a date in the past.

Issues with Cookies

Using cookies gives you excellent state management capabilities, as they are simple to implement, and they help you move resources off the server. Like almost any particular technique, cookies have some limitations.

Some Users Don't Allow Cookies

Some users believe that viruses can be sent in a cookie and will not allow them onto their computers. Although there have never been any documented cases of this happening, and no one could realistically send a virus through a cookie, a lot of users still turn off the ability to accept cookies. When this happens, the user will not be able to use your site if you use cookies for managing state.

Performance Can Deteriorate

Imagine that a user walks through a wizard on your site, as you gather 100 pieces of data from that user over several pages. Each page needs to post gathered data to the server. If you wait until all 100 pieces of data are gathered, you need to store that data somewhere in the meantime. If you keep putting data into a cookie, there is a lot of data being sent back and forth between the browser and the server. This will eat up a lot of bandwidth and could slow your whole site down. Remember, the data has to go both ways for each page the users hit on your site.

Cookies Take Up Memory

Some browsers impose a limit on the size of the cookie data they can accept, or the number of cookies they can accept at one time. In addition, the amount of memory that you may chew up on the user's machine may cause their operating system to swap some memory to disk. Under this circumstance, the cookie has slowed down your user's machine as well as the server.

Using the ViewState Property

In some cases, you do not need to maintain state across pages, only between calls to the same page. If you need to do this, you can use the ViewState property of an ASP.NET page. (The ViewState property is an instance of a StateBag class. That is, the property is defined "As StateBag". In the **ViewStateTest.aspx** page, you input three values. You can submit them to the server and display some data in the Result label. At the same time, you create three variables in a "state bag" using the ViewState object.

In the Click event procedure of the **Create State** button, write the following code to display the data and create the ViewState object:

VB.NET

```
Public Sub btnSubmit_Click(ByVal sender As Object, _
 ByVal e As System.EventArgs) Handles btnSubmit.Click
    ViewState("First") = txtFirst.Text
    ViewState("Last") = txtLast.Text
    ViewState("Password") = txtPassword.Text

    lblResult.Text = txtLast.Text & ", " & _
     txtFirst.Text & " (" & txtPassword.Text & ")"
    lblStateResult.Text = ViewState.Count.ToString()
End Sub
```

C#

```
private void btnSubmit_Click(object sender, System.EventArgs e)
{
     ViewState["First"] = txtFirst.Text;
     ViewState["Last"] = txtLast.Text;
     ViewState["Password"] = txtPassword.Text;

     lblResult.Text = txtLast.Text + ", " +
        txtFirst.Text + " (" + txtPassword.Text + ")";
     lblStateResult.Text = ViewState.Count.ToString();
}
```

In the code listed above, you use the ViewState object just like the Session object. Create a new variable just by supplying the name of the variable in parentheses and quotes and assigning a new value to it. Each of the values comes from the text boxes on the Web page.

While you are still on the same page, you can test to see that the values in the state bag are really preserved during the round trip. You can click the Check State button to redisplay the values from the StateBag. (When you click the button, you're submitting the page to the server, forcing a round trip.)

VB.NET

```
Private Sub btnStateCheck_Click(ByVal sender As Object, _
  ByVal e As System.EventArgs) Handles btnStateCheck.Click
     If ViewState("First") Is Nothing Then
        lblResult.Text = "NO STATE BAG SETUP"
     Else
        lblResult.Text = _
         "First=" & ViewState("First").ToString() & " - " & _
         "Last=" & ViewState("Last").ToString() & " - " & _
         "Password=" & ViewState("Password").ToString()

        lblStateResult.Text = ViewState.Count.ToString()
     End If
End Sub
```

C#

```
private void btnStateCheck_Click(object sender,
  System.EventArgs e)
{
     if (ViewState["First"] == null)
     {
        lblResult.Text = "NO STATE BAG SETUP";
     }
     else
     {
        lblResult.Text =
           "First=" + ViewState["First"].ToString() + " - " +
           "Last=" + ViewState["Last"].ToString() + " - " +
           "Password=" + ViewState["Password"].ToString();

        lblStateResult.Text = ViewState.Count.ToString();
     }
}
```

StateBag objects are only valid while the page is active. As soon as you navigate to another page, the StateBag object is discarded. If you really wish to give the ViewState property a test, you could click on the Create State button, and then restart the IIS service. After restarting the IIS service, click the Check

State button and see that all of your values have been preserved. The ViewState property, with all the data, has been preserved in a hidden field in your form. This is a nice feature–it means that restarting the Web server will not affect the data you were storing for that one page.

State Bag Issues

The StateBag object provides a nice mechanism for storing view state infomation. It gives you the ability to store state for an individual page, across round trips to the server. But, like most techniques, it comes with its own set of issues:

- The ViewState property is only valid for a single page. For multiple pages, you need to use another structure.
- The more information you store in your StateBag object, the more data you have to exchange with your client's browser. This can cause performance problems across your whole site. Remember that the data must go back and forth with each post of the page back to the server.
- Memory could become an issue: Because pages could grow quite large, with many controls, the ViewState information can take up a large amount of memory on the user's machine. This may cause the user's operating system to swap some memory to disk and slow down the user's machine even more.

Cookieless Sessions

If you know you will be running in a Web farm (described in the next section), or you do not want to take a chance on using cookies, you can configure ASP.NET to run without using cookies. You will still be able to use the Session object, but a memory cookie will not be generated. You can accomplish this by changing the Cookieless attribute in the SessionState setting in the Web.Config file from "false" to "true". Once you've set this option to true, the SessionID value is added to the URL as shown in the following example.

```
http://localhost/StateMgmt/(pe5t5r55ay2cqrfpu4tvgm45)/StateBagTes
t.aspx
```

The additional text (pe5t…) added to the URL contains the session ID value. IIS uses this number as the session ID when this page is resubmitted back to the server. If you are using any hyperlink elements or <Form Action=""> tags in your HTML, .NET will add this session ID to each of these HTML tags as well. This works for all pages in your Web site, regardless of whether they are HTML or ASPX pages.

To enable cookieless sessions, you need to make one change in the Web.Config file in your ASP.NET application. Follow the steps below to enable a cookieless session.

1. Open your application’s Web.Config file in the Visual Studio.NET editor.
2. Locate the <sessionState> XML element.
3. Change the Cookieless attribute from “false” to “true”.

```
<sessionState
    mode="InProc"
    stateConnectionString="tcpip=127.0.0.1:42424"
    sqlConnectionString="data source=127.0.0.1;user
id=sa;password="
    cookieless="true"
    timeout="20"
/>
```

WARNING: The cookieless attribute is case-sensitive. You must use “true” and “false” explicitly. If you use “True” or “False” it will not work!

Once you have done this, run the application again and watch the URL. You will see a session ID appear with each page that is displayed. This ID is appended to each and every call to any page in this Web site. The ASP.NET engine handles all of the details of sending this ID back into the Session object.

ASP.NET State Service

One of the problems that you learned about earlier was session state management across a Web farm. When users come into a Web site with several servers that can serve a particular user at any time, the session state does not automatically carry over from machine to machine. Another problem with session state is that it typically runs in the same process space as the IIS service. As a result, if your IIS service goes down, you also lose all session states.

If you wish to solve both of these problems, you need to move the session state management to an out-of-process component that is separate from the IIS service. When the .NET Framework is installed, a new service called ASP.NET State is installed on your server. This service manages the Session object in a separate process. This separate process can be located on the same machine as IIS, or on a separate machine.

If you choose to use a separate server to be the state management machine, all the servers in your Web farm use this machine to store and retrieve state for a user. No matter which machine serves the user, the state is maintained on a separate machine, one from which each of these Web servers can retrieve data.

You are probably thinking that this is going to be very difficult to set up and maintain. Nothing could be further from the truth! In fact, all it takes is to change one setting in each Web server's Web.Config file.

Follow the steps below to enable an out-of-process session state manager.

1. Using the Services applet, start the ASP.NET State Service.
2. Open your application's Web.Config file in the Visual Studio.NET editor.
3. Locate the <sessionState> XML element.
4. Change the Mode attribute from InProc to StateServer.
5. Make sure the Cookieless attribute is set to true.

```
<sessionState
    mode="StateServer"
    stateConnectionString="tcpip=127.0.0.1:42424"
    sqlConnectionString="data source=127.0.0.1;user
id=sa;password="
    cookieless="true"
    timeout="20"
/>
```

After setting this attribute, you can test your changes:

1. Run some code that creates a session variable.
2. Stop and re-start the IIS Admin and Web Publishing Services.
3. Go back to a page that retrieves the session variable that you previously set: you'll find that the saved variable still exists.

If you will be using a separate machine for state management, make sure that the ASP.NET State service is running on this other machine. Next, set the stateConnectionString attribute in the <sessionstate> XML element to the name or IP address of the machine that will manage the state.

> TIP: By default, the stateConnectionString attribute is set to 127.0.0.1, which corresponds to the current machine. If you want to manage state on a separate machine, you'll need to modify this value.

Issues With the ASP.NET State Service

There are some issues with using this ASP.NET State service:

- **Performance.** The performance of retrieving state from an out-of-process service will be slower than from an in-process service. If you are retrieving data across the network to a state server, you also have the network traffic to contend with. This can slow your retrieval of state significantly.
- **Redundancy.** If you use another machine to manage state, you will need to set up some redundancy for this machine in case it crashes. Of course this redundancy will not help you if the original machine dies, because all of the session data is stored in memory.

SQL Server State Management

If you have an installation of SQL Server available, you might want to consider moving your session state management to SQL Server, especially if session state is of critical importance to your application and you can't afford to lose the session state for a user.

Follow the steps below to use SQL Server to manage your state.

1. Open the Web.Config file in the Visual Studio.NET editor.
2. Locate the <sessionState> XML element.
3. Change the **Mode** attribute to **SQLServer**.
4. Make sure the **Cookieless** attribute is set to **true**.
5. Change the **sqlConnectionString** attribute so the Data Source expression refers to your server. Add valid User Id and Password values, as well.You do not need to specify the name of a database, as the tables that manage state are located in Tempdb.

```
<sessionState
    mode="SQLServer"
    stateConnectionString="tcpip=127.0.0.1:42424"
    sqlConnectionString="data source=(local);user
id=sa;password="
    cookieless="true"
    timeout="20"
/>
```

After setting these attributes, you need to create the ASPState database with some stored procedures that the .NET Framework will use to manage state. Follow these steps to complete the installation:

1. Find the file named **InstallSqlState.sql** located in your <systemdrive>\Winnt\Microsoft.NET\Framework\<Version> folder.
2. Load the InstallSqlState.sql file into SQL Query Analyzer and execute the statements. This creates the ASPState database and all of the appropriate stored procedures.

After you have done these things, try running your sample that creates a session variable. You can stop and start the IIS and Web Publishing Services, and once again, your state will remain. If you open SQL Server Enterprise Manager and navigate to the Tempdb database, you will find a table named ASPStateTempSessions. Open this table and you will find a record with your session ID, the time this session was created, and when this session will expire. You will also see several binary fields. These fields contain the data for the session. You won't be able to look at this data, but then you don't really need to because the .NET Framework takes care of all of this for you automatically.

Issues with Automatic SQL Server State Management

Although using SQL Server to store your session state relieves you of many difficult development issues, you'll still need to consider some important limitations:

- **You're limited to SQL Server.** This technique can only use SQL Server, no other server database. If you do not have a SQL Server installation available, you will be unable to use this solution.
- **Performance may suffer**. Like any of the state management techniques, using SQL Server to manage your application's state can cause your performance degrade a little. Because it takes a little bit of time to make a connection and read and write state information in the database, there's no avoiding a small bit of overhead.

What's Changed from ASP?

State management capabilities were very primitive in ASP, compared to ASP.NET. The Session object could only run in-process with the IIS server and did not have any Web farm capabilities. If you wanted to use a Web farm, you were forced to create some sort of COM component to store state into a SQL Server database. You were able to use hidden input fields, but you didn't get the automatic state management provided by the ViewState object, and you had to manage it all manually. There were no built-in StateBag objects, so any state you needed to keep while on a page had to be managed in hidden input fields, and you had to handle the encryption yourself.

Summary

In this chapter you learned about the many ways you can maintain state on a Web site. In even the smallest of Web applications, you will most likely need to use one or more of these techniques. Storing state information in SQL Server offers flexibility and performance, although sometimes you might need to use a combination of the techniques presented. Being aware of all the possibilities is important, and you might consider each of these options as you plan for state management in your own applications.