Using Web Forms

Yesterday’s lesson introduced ASP.NET programming. You saw how to implement a basic ASP.NET page using a simple Web form and how to use code behind with Web forms. Today’s lesson will introduce more techniques that you can use with Web forms, including basic state management and some details about the life cycle of Web forms. Today you will learn about the following:

- The way Web forms are processed
- An introduction to Web form state management
- The Session and Application objects
- The way global objects are stored

Understanding How Web Forms Are Processed

As we discussed yesterday, a Web form is another name for an ASP.NET page. You learned yesterday that a Web form can be made up of a single file with an .aspx extension. You also saw how an .aspx file and a code behind file can be
combined to make a Web form. As we’ll show in the following few days, Web forms can also contain your own custom controls, defined similarly to .aspx pages, called *user controls*.

Because a Web form can include HTML, ASP.NET Web controls, custom (user) controls, and code behind, creating them can get complicated pretty quickly. However, you need to remember only a few key ideas about ASP.NET page processing so that you can develop and debug effectively.

The first key idea is that much of the ASP.NET infrastructure is set up so that a single ASP.NET page can post form data back to itself repeatedly. You saw this “postback” technique in yesterday’s examples with the Login page. This technique is most useful when you divide your Web site’s functionality into a few main pages. We’ll explore this kind of design in the first bonus programming project in this book, after Day 12.

The second key idea to remember is that all ASP.NET pages are eventually compiled into executable files by the ASP.NET infrastructure. This means that every time a page is processed and rendered, a small program corresponding to each Web form is executed by the ASP.NET infrastructure.

Let’s explore in more detail how Web forms are processed. Listing 3.1 shows a sample Web form that performs an English unit to metric unit conversion, and Figure 3.1 shows the resulting page.

**Listing 3.1**  EnglishToMetric.aspx: Converting Feet and Inches to Meters

```
<%@ Page Language='VB' %>
<html>
<body>
<form runat='server'>
<h2>English to Metric conversion</h2>
<h3>Please enter your height, and then click the Convert button</h3>
<asp:Textbox id='Feet' runat='server'/>Feet
<br/>
<asp:Textbox id='Inches' runat='server'/>Inches
<br/>
<asp:Button OnClick='OnConvert' Text='Convert' runat='server'/>
<br/>
<asp:Label id='lblMeters' runat='server'/>
</form>
</body>
</html>
<script runat='server'>
Sub Page_Load(Sender As object, e As EventArgs)
```

20: Sub Page_Load(Sender As object, e As EventArgs)
Let’s examine how Web forms work using Listing 3.1 as our reference. Web forms like Listing 3.1 are processed in two different ways:

- A Web form is rendered when the user initially browses to the .aspx file. In this case, the Web form doesn’t process any events because there has been no user interaction.
- A Web form may be processed after the user interacts with one of the page controls. For instance, the user might click a button on the page or select an item from a drop-down list. When the user does so, the same Web form gets hit by the user’s browser, this time with information about what the user has done.

Through both cases, a Web form goes through five distinct stages when it’s rendered, as shown in Figure 3.2.
In Figure 3.2, the first stage deals with Web control initialization. We’ll learn more details about how controls save and then remember their state in tomorrow’s lesson. You don’t need to be too concerned about this stage in your own programming tasks.

Your code is frequently involved in the second stage of Web forms processing, when the ASP.NET infrastructure fires the Load event in the Page class. As Listing 3.1 shows, you can use the Page_Load event (Lines 20-26) to change the contents of controls or to perform any other kinds of processing before a page is rendered. If the page posts back to itself (that is, if the user clicks the Convert button), the Page_Load event will fill in zeros if the user leaves any field blank (Lines 22–25).

The third stage of Web forms processing is for control event handling, which happens only if the user manipulates one of the Web controls on the page. This stage won’t happen if this is the first time the user is browsing to this particular page. In Listing 3.1, the OnConvert method is called during this stage (Lines 28–35). Note that event handling happens after the Page_Load event. You will learn about more complex Web controls that contain many events on Day 5, “Using Advanced ASP.NET Web Controls.”

In the fourth stage, the HTML for the page is rendered. This process involves calling the Render method for the page, which will call the Render method for every control, among other things.

**Note**

Listing 3.1 doesn’t contain an explicit Render method, because the page class and Web control classes have default implementations for it already. You will need to override the Render method only if you want to customize the exact HTML that each class produces.
In the last stage, the Page_Unload event gets called. You can use the Unload event to clean up database connections and other objects that have to be closed explicitly, for example. (Listing 3.1 also doesn’t define the Page_Unload event, because it doesn’t need to clean up any objects.)

**Performing Basic State Management in Web Applications**

Most Web applications are customized for each individual site user. Even if the site behaves the same way for every user, a user will often perform an operation on the site that spans two or more pages. For instance, if your Web application has a search feature, you might use one page to gather a search string. You might use a second page to display the results and a third page to hold a history of search queries. Designing your ASP.NET code to keep track of search queries, search results, and search history entries involves state management. In the broadest terms, state management refers to the ways an ASP.NET page “remembers” what a user has done previously.

If the searching feature was part of a standalone program, such as a Visual Basic Windows application, storing the user’s state, including search queries and results, isn’t an issue—just store the search strings and results on the client computer. With Web applications, you can’t store much of anything on the client computer because the client is just a Web browser.

Web programmers have invented a number of methods for storing state in Web applications. The most basic methods include cookies, query strings, and hidden fields. These methods allow you to store a small bit of information, such as a key to a database table.

By using ASP.NET, you have these and several other more powerful state options at your disposal. These methods include the Session and Application objects and the ViewState property. ASP.NET also adds a new class for page caching, Response.Cache. You can use this feature to achieve significant performance improvements in your Web applications, usually after the bulk of coding is complete. Page caching will be explained in more detail on Day 7.

**Using Cookies**

Cookies allow you to store small bits of data on the user’s computer. They take up a small amount of space on the user’s hard drive and are often useful for storing nonessential information, such as user preferences.
ASP.NET lets you manipulate cookies quite easily with the Cookies collection on the Request and Response objects. Listings 3.2 and 3.3 are two companion pages that will read and write cookies that you enter. Figures 3.3 and 3.4 show the two pages.

**LISTING 3.2  WriteCookies.aspx: Writing Arbitrary Cookies**

```vbnet
<%@ language="VB" %>
<script runat="server">
Sub WriteClicked(Sender As Object, e As EventArgs)
 'Create a new cookie, passing the name into the constructor
 Dim cookie as HttpCookie = new HttpCookie(NameField.Text)
 'Set the cookies value
 cookie.Value = ValueField.Text
 'Set the cookie to expire in 1 minute
 Dim dtNow as DateTime = DateTime.Now
 Dim tsMinute as New TimeSpan(0, 0, 1, 0)
 cookie.Expires = dtNow.Add(tsMinute)
 'Add the cookie
 Response.Cookies.Add(cookie)
 Response.Write("Cookie written. <br><hr>")
 End Sub
</script>
<html>
<body>
<h3>Use the button below to write cookies to your browser </h3>
The cookies will expire in one minute.
<form runat="server">
 Cookie Name <asp:textbox id="NameField" runat="server"/>
 Cookie Value <asp:textbox id="ValueField" runat="server"/>
 <asp:button text="WriteCookie" onclick="WriteClicked" runat="server" />
</form>
<a href="readcookies.aspx">Read the cookies</a>
</body>
</html>
```

Tip

Store information that you are willing to lose in cookies. Users can delete cookies at any time, and some users disable them altogether.
LISTING 3.3  ReadCookies.aspx: Reading Cookies Written from the WriteCookies Example

```vbnet
<%@ language="VB" %>
<script runat="server">
Sub ReadClicked(Sender As Object, e As EventArgs)
  'Get the cookie name the user entered
  Dim strCookieName as string = NameField.Text

  'Grab the cookie
  Dim cookie as HttpCookie = Request.Cookies(strCookieName)

  'Check to make sure the cookie exists
  if cookie Is Nothing then
    Response.Write("Cookie not found. <br><hr>")
  else
    'Write the cookie value
    Dim strCookieValue as string = cookie.Value.ToString()
    Response.Write("The " & strCookieName & " cookie contains: <b>" & strCookieValue & "</b><br><hr>")
  end if
End Sub
</script>

<html>
<body>
Use the button below to read a cookie<br>
<form runat="server">
  Cookie Name <asp:textbox id="NameField" runat="server" />
  <asp:button text="ReadCookie" onclick="ReadClicked" runat="server" />
</form>
<a href="writecookies.aspx">Write Cookies</a>
</body>
</html>
```

FIGURE 3.3
The WriteCookies.aspx page writes cookies to the user’s browser.
To write a cookie, create a new `HttpCookie` object (Line 6 of Listing 3.2), assign a string to its `Value` property (Line 9), and then call the `Add()` method on the `Response.Cookies` object (Line 17). You can also set the time of expiration for a cookie by setting the `Expires` property to a `DateTime` value (Line 14). 

ReadCookies.aspx in Listing 3.3 shows that it’s equally easy to read cookies back, using the `Request.Cookies` collection (Line 9), which is indexed by cookie name.

Cookies can store only strings, so if you need to store a more complex data type, it must be converted into a string. One possibility for storing complicated data structures is to write the structure out as an XML string and convert it back when reading the cookie.

You can store multiple strings in a cookie by treating each cookie as a collection object. For example, the following would work fine:

```vbnet
Dim cookie As HttpCookie = New HttpCookie("UserFavorites")
cookie("FavoriteColor") = "blue"
cookie("FavoriteFlavor") = "chocolate"
cookie("FavoriteDrink") = "coffee";
```

The `HttpCookie` class contains some advanced properties, listed in Table 3.1.

**Table 3.1 Advaned Properties of the `HttpCookie` Class**

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain</td>
<td>Gets/sets the domain name that this cookie belongs to. If set, it restricts access to this cookie from Web servers in the specified domain, such as <code>mycompany.com</code>.</td>
</tr>
<tr>
<td>Path</td>
<td>Gets/sets the path that this cookie belongs to. If set, it restricts access to this cookie from Web pages in the specified path.</td>
</tr>
<tr>
<td>Secure</td>
<td>Gets/sets a flag that tells whether the cookie should be transmitted securely to the client browser using the HTTPS protocol. You must have HTTPS set up on your Web server for this option to work.</td>
</tr>
<tr>
<td>HasKeys</td>
<td>Tells whether the cookie is made up of a collection of strings.</td>
</tr>
</tbody>
</table>
Using Hidden Fields and Query Strings

You also can store small amounts of information on the client by using hidden fields. Hidden fields are HTML elements, similar to text boxes, where you can store strings. Web browsers don’t display hidden fields in page output. However, when you use a hidden field within an HTML form, the contents are submitted back to your program.

To use hidden fields, use a code line that looks similar to the following within a <form> area:

```html
<input type="hidden" value="Value That You Need to Store" id="KeyName">
```

You can extract the hidden field’s value by using the Params collection in the Request object, using a code line like this:

```vbnet
Dim strValue As HttpCookie = Request.Params("KeyName")
```

As you will see in the following lessons, hidden fields play a much smaller role in ASP.NET Web programming than in other technologies, such as the older ASP. This is true because ASP.NET Web controls keep track of their own state and because the Page class contains a ViewState collection in which you can store string data easily.

You can also use query strings to pass parameters from page to page. Query strings are bits of information appended to an URL, as in the following line:

```html
<a href='ShowSales?Dept=Mfg&Year=2001'>Manufacturing Sales</a>
```

Listings 3.4 and 3.5 show examples of one page that passes a parameter to another page using a query string.

**Listing 3.4** ShowSales.htm: Passing a Query String to SalesFigures.aspx

```html
<html>
<body>
<h2>Please select one of the links below for Widgets Inc quarterly sales figures</h2>
<a href='salesfigures.aspx?Department=Manufacturing'>Manufacturing</a>
<br/>
<a href='salesfigures.aspx?Department=RD'>Research & Development</a>
</body>
</html>
```

**Listing 3.5** SalesFigures.aspx: Showing Different Sales Depending on a Query String

```vbnet
<%@ Page Language="VB" %>
<html>
<body>
<h2>Sales figures for Widgets Inc</h2>
```
Listing 3.5  continued

```vbscript
Sub WriteSales()

    if Request.Params("Department") = "Manufacturing" Then
        Response.Write("<b>Manufacturing Quarterly Sales</b><br>
        Response.Write("Quarter 1 Sales: $24M<br>
        Response.Write("Quarter 2 Sales: $34M<br>
        Response.Write("Quarter 3 Sales: $12M<br>
        End If
    if Request.Params("Department") = "RD" Then

        Response.Write("<b>Research and Development Quarterly Sales</b><br>
        Response.Write("Quarter 1 Sales: $2M<br>
        Response.Write("Quarter 2 Sales: $3M<br>
        Response.Write("Quarter 3 Sales: $1M<br>
        End If

End Sub
</script>
```

Working with the Session Object

Programming with query strings and hidden fields is cumbersome if you need to manipulate more than a trivial amount of state data. Luckily, ASP.NET gives you a better way to store state for each user on the server with the Session object. Every time a new browser hits your ASP.NET application, a new Session object is created for that Web browser. You can store data in the Session object, and it will be available from hit to hit for the same Web browser. Sessions expire after 20 minutes of inactivity by default, although you can change this behavior, as we’ll show shortly.

Sessions aren’t magic. By default, ASP.NET uses a cookie containing a unique and random ID that it uses to look up the Session object in the ASP.NET server process.

Tip

If needed, you can turn off the cookies to instantiate cookieless sessions. You can also offload the storage of session state onto a different server or into a SQL Server database, for “Web farm” scenarios. We’ll explain these different session configurations on Day 18, “Configuring Internet Applications.”
Adding Data to the Session Object

You can easily add data to the Session object. The line

```csharp
Session("ValidUser") = true
```

automatically creates a new variable called ValidUser (if it doesn’t exist already) and sets it to true. By default, every variable added to the Session object is of the .NET Object type.

Because variables in the Session object are of type Object, proper programming practice argues that you should cast them to the appropriate type when accessing them:

```csharp
Dim strUserName As String = CStr(Session("UserName"))
```

Because Visual basic is very good at automatic data conversions, however, you can usually leave out the explicit cast as shown here:

```csharp
Dim strUserName As String = Session("UserName")
```

You might be wondering whether it’s appropriate to store large custom objects in Session. The answer is that you should avoid storing large amounts of data in session if possible. You can easily overburden your Web server by storing large amounts of data in Session, especially if your Web site has many users. Databases are a better choice for storing large amounts of state data.

The Session object is of type HttpSessionState. Its default property is the Item collection, which allows you to access the stored items by using the () notation.

Lifetime of the Session Object

A new session is created once for each new browser that hits your ASP.NET Web site. If a user stops hitting your Web site, his Session will time out after 20 minutes of inactivity, by default.

You can find out how long the Session timeout setting is by using the Timeout method. The following code line prints "20" by default:

```csharp
<% Response.Write(Session.Timeout.ToString()) %>
```

You can change the timeout for the Session object by assigning the Timeout property to a certain value, in minutes, such as

```csharp
Session.Timeout = 5
```

Removing Objects from the Session Object

Because sessions time out, you don’t really need to remove objects from them. However, you can remove objects by using the Session.Remove() method. You also can remove
everything in the Session object by using the RemoveAll() method. You might want to use these two methods to conserve Web server resources, especially if you store large objects in Session.

Listing 3.6 shows a page that lets you add and remove strings to the current Session.

**Listing 3.6  SessionPopulate.aspx: Adding and Removing Strings from Session State**

```vbnet
<%@ language="VB" %>
<script runat="server">
Sub AddClicked(Sender As Object, e As EventArgs)
    Session(Key.Text) = Value.Text
End Sub

Sub RemoveClicked(Sender As Object, e As EventArgs)
    Session.Remove(Key.Text)
End Sub
</script>

<html>
<body>
<h3>Current items in Session</h3>
<form method="post" runat="server">
<table border="1">
<tr>
    <td><b>Item Name</b></td>
    <td><b>Value</b></td>
</tr>
<% Dim strSesKeyName As String
    Dim strSesItem As String
    Dim i As Integer
    For i = 0 To Session.Count - 1
        strSesKeyName = Session.Keys(i)
        strSesItem = Session(i)
        Response.Write("<tr><td>" & strSesKeyName & "</td><td>" & strSesItem & "</td></tr>"
    Next
%>
</table>
<br>
Key <asp:textbox id="Key" runat="server"/>
Value <asp:textbox id="Value" runat="server" /><br>
<asp:button text="Add/Modify Key/Value pair" onclick="AddClicked" runat="server" />
<asp:button text="Remove Key" onclick="RemoveClicked" runat="server" />
</form>
</body>
</html>
```
You also can kill off a session immediately by using the Abandon() method. As soon as Abandon is called, a new session is created automatically.

**Tip**

---

**Working with the Application Object**

Not all data in your Web applications is specific to individual users of your Web site. For instance, if your application tracks stock prices, this data should be available to all users. Data like this is called *global data*, and using global data in session doesn’t make sense. Fortunately, ASP.NET provides the Application object just for this purpose.

The Application object is stored in memory and unlike Session, objects can’t be offloaded to another server or a SQL database. This limits the usefulness of the Application object for Web farm scenarios. However, if your Web application contains a large amount of read-only data that takes awhile to load, the Application object is a good place to store it.

**Adding Objects to the Application**

You can add objects to the Application object in the same way that you can in Session objects. For instance,

```
Application("WebStoreName") = "Stanley Sprockets Online"
```

stores a string into the Application object. Conversely,

```
Dim strStoreName As String = Application("WebStoreName")
```

fetches the value.

**Synchronizing Code in the Application Object**

If more than one Web page tries to store or update a value in the Application object at the same time, the data can become corrupt unless the updates are synchronized. This situation could easily happen with a site that experiences even a moderate amount of traffic, so using the Lock() and Unlock() methods of the Application object is important when you’re storing data. Listing 3.7 shows how to synchronize code properly.
Listing 3.7 HitCounter.aspx: Updating a Hit Counter for the Web Site

```vbnet
<%@ language="VB" %>
<
' We need to check if the "Hits" object
' has never been assigned and is null.
' If it is null, then we set it equal to 0
if Application("Hits") Is Nothing Then
  Application.Lock()
  Application("Hits") = 0
  Application.UnLock()
End If
Dim nHits As Integer = Application("Hits")
nHits = nHits + 1
Application.Lock()
Application("Hits") = nHits
Application.UnLock()
%
<html>
<body>
<h2>This site has been hit
  <% Response.Write(Application("Hits").ToString()) %>
times</h2>
</body>
</html>
```

Caution

Locking the Application object blocks other pages that use the object from running until it is unlocked. For best performance, don't make many updates to the Application object. The best use for the object is for infrequently updated read-only data. An example might be an application that tracks sporting events and stores the latest scores from current games every five minutes.

Removing Objects from the Application

You can remove objects from the Application object by using the Remove() and RemoveAll() methods, just like with the Session object.

Lifetime of an Application

The Application object is created on the first hit from the first user to your ASP.NET application. The Application object remains in memory until the Web server is shut down, or the web.config site configuration file has changed. (We introduce the web.config file shortly.) Unlike the Session object, the Application object doesn't have an Abandon() method.
Setting Up Global Objects with the global.asax File

ASP.NET uses a special file, called global.asax, to establish any global objects that your Web application uses. The .asax extension denotes an application file rather than .aspx for a page file.

Each ASP.NET application can contain at most one global.asax file. The file is compiled on the first page hit to your Web application. ASP.NET is also configured so that any attempts to browse to the global.asax page directly are rejected.

Listing 3.8 shows a global.asax file that you can use to make a more complete hit counter.

**Listing 3.8  global.asax: Event Handlers for the Application and Session Objects**

```vbnet
<%@ language="VB" %>
<script runat="server">
Sub Application_Start(Sender As Object, e As EventArgs)
  Application("Hits") = 0
  Application("Sessions") = 0
  Application("TerminatedSessions") = 0
End Sub

'The BeginRequest event is fired for every hit
to every page in the site.
Sub Application_BeginRequest(Sender As Object, e As EventArgs)
  Application.Lock()
  Application("Hits") = Application("Hits") + 1
  Application.UnLock()
End Sub

Sub Session_Start(Sender As Object, e As EventArgs)
  Application.Lock()
  Application("Sessions") = Application("Sessions") + 1
  Application.Unlock()
End Sub

Sub Session_End(Sender As Object, e As EventArgs)
  Application.Lock()
  Application("TerminatedSessions") = Application("TerminatedSessions") + 1
  Application.Unlock()
End Sub
```
LISTING 3.8  continued

33:    Application("TerminatedSessions") + 1
34:    Application.UnLock()
35:
36:    End Sub
37:
38:    Sub Application_End(Sender As Object, e As EventArgs)
39:
40:         'Write out our statistics to a log file
41:         '...code omitted...
42:
43:    End Sub
44: 

ANALYSIS  The global.asax file in Listing 3.8 contains event handlers for the Session and Application objects. Each event handler has the same signature as the Page_Load event handler.

The code in Listing 3.8 handles three Application object-related events: Start (Lines 3–9), End (Lines 38–43), and BeginRequest (Lines 13–19). Start and End are called when the Web application starts and ends, respectively. BeginRequest is called for every page hit that the site receives. Listing 3.8 updates the total number of hits in this event.

The Session Start (Lines 21–27) and End (Lines 29–36) events are handled in the middle of the listing. These two events count how many different Web users have accessed the site.

You can write a simple page to utilize the statistics that Listing 3.8 tracks. Listing 3.9 shows a page that writes out the results of the hit-counting code in the global.asax file. Figure 3.5 shows the statistics page after a few hits.

LISTING 3.9  Statistics.aspx: The Results of the Tracking in the global.asax File

```html
<%@ page language='VB' %>
<html>
<body>
<h2>Statistics for the Test Web Application</h2>
Total hits: <% Response.Write(Application("Hits").ToString()) %> <br>
Total sessions: <% Response.Write(Application("Sessions")) %> <br>
Expired sessions: <% Response.Write(Application("TerminatedSessions")) %> <br>
</body>
</html>
```
Adding Objects to the global.asax File

To use global objects in your ASP.NET application, add the <object> tag in the global.asax file for each one. The <object> tag has an optional attribute called scope, which determines if the added object will be created on-the-fly, associated with the Application object, or associated with the Session object.

To explore the <object> tag, let's create a simple class that stores and retrieves strings. The sample is going to associate an object of this class with the Application object in the global.asax file, so the class must be thread-safe. The term thread-safe means that many client threads can access the class at the same time without any data corruption. Because ASP.NET uses one thread per page, ensuring that the class is thread-safe is critical if multiple users browse the site at the same time.

Understanding Threads

What's a thread? To answer, let's review processes first. All Windows applications are processes that run on your computer. Processes contain their own code and memory space, and can interact with computer peripherals, such as the screen or the network card. ASP.NET runs as a process, and it executes your code, of course.

Each process contains one or more threads. A thread is like a process or an individual program, because it also executes a certain set of code. However, a thread is a "lightweight" version of a process. Threads live inside processes, and use a process's memory. The Windows operating system gives each thread a small amount of time to execute and quickly switches between threads so that it seems
like more than one thread is executing at the same time. For all practical intents, the threads are running at the same time. Because threads use their parent process's memory, they can potentially change the same object (in memory) at the same time. For two threads, A and B, thread A might add 10 to the counter object. Thread B might subtract 10 from the counter. If the two threads are switched on and off by the operating system in an "unlucky" way, the counter object could contain a scrambled result.

When each ASP.NET page is being processed, it gets its own thread. If more than one user accesses the Web site at the same time, many threads will appear even if both users are accessing the same ASP.NET page.

To prevent threads (and ASP.NET pages) from interfering with each other when accessing the same object, use the technique in the example that follows.

To make the class thread-safe, use the Synchronized method of the base collection class, Hashtable. This class is shown in Listing 3.10.

**Listing 3.10  MyClass.vb: Implementing a Class to Store and Retrieve Strings in a Thread-Safe Way**

```vbnet
Imports System
Imports System.Collections

Namespace TestApplication

Public class MyClass1
    Private _col As Hashtable
    ' _colSync will be a thread-safe container for _col
    Private _colSync As Hashtable

    Public Sub New()
        _col = New Hashtable()
        _colSync = Hashtable.Synchronized(_col)
    End Sub

    Public Sub AddItem(Name As String, Value As String)
        _colSync(Name) = Value
    End Sub

    Public Function GetItem(Name As String) As String
        Return _colSync(Name)
    End Function

End Class
```


Using Web Forms

Note that the DLL file is placed in the bin directory off the project directory. The next step is to add this object to the global.asax file with the <object> tag. A short global.asax file that does this follows:

```xml
<object id="MyStringCollection" runat="server"
    class="TestApplication.MyClass1" scope="Application" />
```

The `id` attribute tells ASP.NET what to call our object when it’s used later. The `class` attribute can be used to specify COM objects in addition to .NET components, but by using their `ProgID`.

If the listing omitted the `scope` attribute, a new object is created on-the-fly for every page that uses the `StringCollection` object.

Let’s write a sample page that uses `StringCollection`. Listing 3.11 shows just such a page.

**LISTING 3.11** UseObject.aspx: Using the Global Object

```html
<%@ language="VB" %>
<script runat="server">
Sub Page_Load(Sender As Object, e As EventArgs)
    MyStringCollection.AddItem("FirstUser", "Joe Smith")
End Sub
</script>
<html>
<body>
The name of the first user is
    <% Response.Write(MyStringCollection.GetItem("FirstUser")) %>
</body>
</html>
```

**Putting Code Behind the global.asax File**

If you use Visual Studio.NET to create your Web project, it will use the code behind feature of ASP.NET for global.asax. The code behind file that is generated is named
global.asax.vb, when using the Visual Basic compiler. To use code behind in global.asax manually, use the Application directive instead of the Page directive, like this:

```vbscript
<@ Application Inherits="MyApplication.GlobalClass" %>
```

Listing 3.12 shows a code behind example for the global.asax file.

**Listing 3.12  GlobalClass.vb: Implementing the Code for the global.asax File**

```vbscript
Namespace MyApplication
    Imports System
    Imports System.Web
    Imports System.Web.SessionState

    Public class GlobalClass
        Inherits System.Web.HttpApplication

        protected Sub Session_Start(Sender As Object, e As EventArgs)
            Response.Write("Session started <br>
        End Sub

    End Class

'note: use "vbc /out:bin\globalclass.dll /t:library globalclass.cs
/r:system.dll"
' to compile with the command line utility

End Namespace
```

**Tip**

You can mix code behind and the `<object>` tag in the global.asax file.

**Configuring the Application**

Configuring Web sites is much easier using ASP.NET than it is with many other solutions. ASP.NET allows you to configure almost every aspect of your site using an XML file called web.config.

The web.config file is a boon for developers who have their Web applications hosted by another company. Rather than have to deal with surly support technicians from your hosting company, you can simply upload new versions of a web.config file at will.
The web.config file is similar to the global.asax file, in that it can be stored at the root of an ASP.NET application. Also, whenever you modify the file, the ASP.NET application is automatically restarted.

Look at a sample web.config file in Listing 3.13.

**Listing 3.13  Sample web.config File to Specify a Session Timeout of Five Minutes**

```xml
<?xml version="1.0" ?>
<configuration>
  <system.web>
    <sessionState
      timeout="5"
    />
  </system.web>
</configuration>
```

Visual Studio.NET creates a more detailed web.config file by default when you create a new Web application. On Day 18, we’ll describe how to construct web.config files in detail.

As a prelude to Day 18’s lesson, here is a list of some changes you can make with a web.config file:

- Set default language for ASP.NET pages
- Turn debugging and tracing on and off
- Specify error-handling pages
- Specify where session state is stored
- Set the authentication mode
- Set globalization parameters
- Prevent pages ending in a specific extension from being downloaded
- Set the browser capabilities of your site
- Add assemblies to your Web site
- Configure Web service

Although this book won’t go into the details of each feature until Day 18, this list should give you a starting point for figuring out the default web.config file that Visual Studio generates.
Summary

Today’s lesson covered the state management features of ASP.NET, including cookies, the Application and Session objects, ViewState, hidden fields, and query strings. You saw how to use cookies to store items such as user preferences on the client computer. You also saw how to use strings and objects in Session and Application state on the Web server.

Today’s lesson didn’t give you enough information for you to decide when to use session state, ViewState, and application state. Day 12 will contain instructions for making an intelligent choice in your own state management approach.

You learned about the global.asax file and how to use it to specify which objects are stored in Session and Application state.

Today’s lesson explained the order of events in a typical ASP.NET Web form as it’s rendered to the client computer. You saw an example that read and modified the values of some simple Web controls on a Web form.

Last, today’s lesson introduced the web.config file and explained how it’s used to configure your ASP.NET Web site. You saw a simple example of a web.config file that set the timeout value for Session objects.

Tomorrow you will learn about Web controls in detail and how to use them in your Web forms to produce powerful Web applications.

Q&A

Q I want to make my ASP.NET Web site extremely fast. What strategies should I take?

A For ultimate performance, don’t use ASP.NET sessions, and don’t use Web Control ViewState (which will be explained in full tomorrow). As you absorb tomorrow’s lesson, this strategy will become more clear.

Q I have heard that ASP.NET contains special features for so called Web Farms, where multiple Web servers are used to host one site. How does this work?

A ASP.NET has features that allow you to store sessions on another computer (the session server) or in a Microsoft SQL Server database. These techniques are explained on Day 18. You can’t use the Application object this way.
Workshop

The workshop provides quiz questions to help you solidify your understanding of the material covered today as well as exercises to give you experience using what you have learned. Try to understand the quiz and exercise before continuing to tomorrow’s lesson. Answers are provided in Appendix A, “Answers to Quizzes and Exercises.”

Quiz

1. What is a postback, and when is it used in ASP.NET Web forms?
2. What are the five different ways of handling state in ASP.NET Web forms?
3. Where are cookies stored, and what kinds of data can they contain?
4. Why would you store an object in Application state instead of Session state?
5. How do you change the timeout for a Session object?
6. What events does the HttpSession class support?
7. What file do you use to handle Session and Application events?
8. What tag do you use to declare an object in the global.asax file?
9. What file do you use to configure a Web application?
10. What method do the .NET collection classes use to return thread-safe versions of themselves?

Exercises

1. Create a new Web service using Visual Studio. What files did Visual Studio create that we didn’t in today’s examples?
2. Add a method called Clear to the MyClass.vb application. How can you ensure that the method is thread-safe?
3. Create a two-page Web application that allows a user to log on and then select her favorite football team. Save the user’s response in a cookie so that subsequent uses of the application will remember the user’s preference.