msdn® training

Module 7: Creating a Microsoft ASP.NET Web Application

Contents

Overview	1
Requirements of a Web Application	2
What Is New in ASP.NET?	3
Sharing Information Between Pages	13
Securing an ASP.NET Application	24
Lab 7: Creating an ASP.NET Web Application	38
Review	50

Microsoft

Overview

- Requirements of a Web Application
- What Is New in ASP.NET?
- Sharing Information Between Pages
- Securing an ASP.NET Application

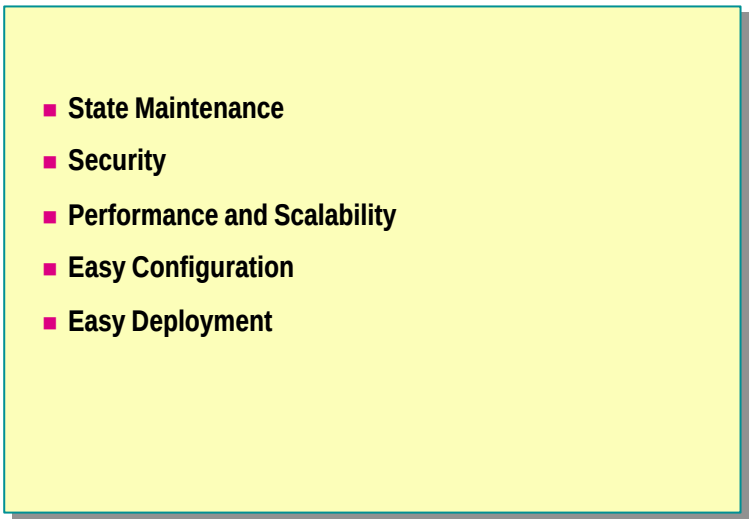
Just as in Active Server Pages (ASP), Microsoft® ASP.NET supports the concept of a Web application with application-specific settings and services. An ASP.NET application is defined as all the files, pages, handlers, modules, and executable code that can be invoked from a virtual directory and its sub-directories on a Web application server.

In this module, you will learn about some of the files that are used to build Web applications and some of the capabilities of an ASP.NET application, such as maintaining state and authentication. You will also learn how to configure and deploy an ASP.NET application.

After completing this module, you will be able to:

- Describe cookie-less sessions.
- Set up cookie-less sessions in the web.config file.
- Use event procedures in global.asax.
- Set up page output caching for ASP.NET pages.
- Share information between pages of an ASP.NET application by using ASP.NET cache, web.config, session variables, and a database.
- Describe how authentication works.
- Set up authentication for an application in web.config.

Requirements of a Web Application



A good Web application meets the following requirements:

- State maintenance

  Web applications should be able to maintain state across pages. If state is maintained between pages, information supplied by users can be reused and users do not need to enter the same information multiple times.

- Security

  A good Web application should have security features. Most importantly, it should be able to authenticate and authorize users who can access the application.

- Performance and scalability

  All Web applications should be built with high performance and scalability in mind. Caching is an extremely important technique for building high-performance, scalable Web server applications.

- Easy configuration

  Configuration is an important aspect of any application. A central requirement of any Web application server is a rich and flexible configuration system. The objective is to enable developers to associate settings easily with an installable application without needing to embed values into code. This allows administrators to adjust or customize these values easily after the Web application has been deployed.

- Easy deployment

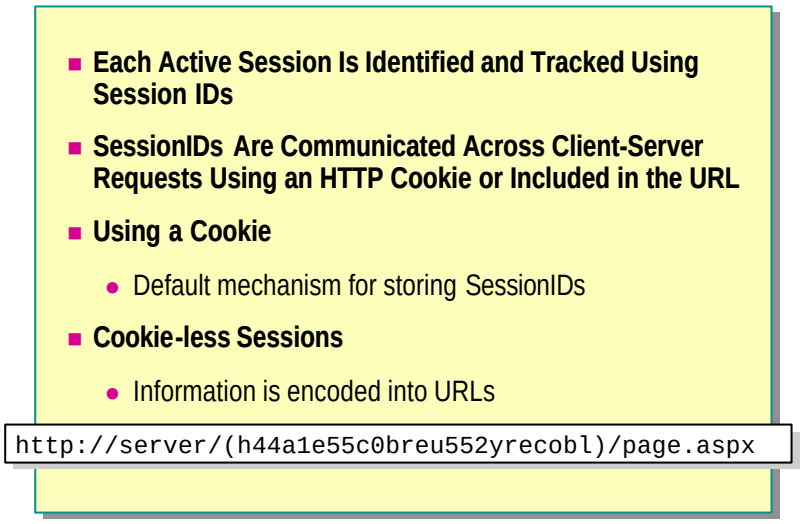
  Historically, one of the biggest problems associated with Web applications is deployment. A well-designed Web application is easy to deploy.

What Is New in ASP.NET?

- Cookie-less Session IDs
- Configuration File (web.config)
- Setting Up Cookie-less Sessions
- Global Application File (Global.asax)
- Demonstration: Using Event Procedures in Global.asax
- Page Caching

In addition to all of the features provided by ASP, ASP.NET provides several new features designed to enhance the functionality of an application. This section focuses on the new features in ASP.NET.

Cookie-less Session IDs



Each active session in ASP.NET is identified and tracked by using a 120-bit SessionID string containing Uniform Resource Locator (URL)-legal ASCII characters. SessionID values are generated by using an algorithm that guarantees both uniqueness and randomness. SessionIDs are communicated across client-server requests by using either a Hypertext Transfer Protocol (HTTP) cookie or included in the URL.

Using Cookies

Cookies are a mechanism by which data can be maintained in a file on the user's computer. By default, SessionIDs are stored in cookies.

However, users can turn off cookies through a setting in their browsers. This creates the risk that your Web application will not work if it requires session information conveyed by cookies and a user has disabled cookies in his or her browser.

Using Cookie-less Sessions

The ability to use cookie-less sessions in ASP.NET is a new concept that was not available with earlier technologies, including ASP.

This method uses URLs, as opposed to cookies, to pass the SessionID to an ASP.NET page. It involves encoding data into a URL, which is done automatically by the browser. This enables you to use session state even with browsers that have cookie support disabled.

For example, the browser generates the following URL for a request to the ShoppingCart.aspx page on the http://localhost/conference web site:

http://localhost/conference/(h44a1e55c0breu552yrecobl)/ShoppingCart.aspx

To enable cookie-less sessions, add the following to the web.config configuration file:

```
<sessionState cookieless="true"/>
```

Configuration File (web.config)

- **All Configuration Information for an ASP.NET Application Is Contained in web.config**
- **Web.config Can Be Placed in the Same Folder as the Application Files**
- **Web.config Contains Sections for Each Major Category of ASP.NET Functionality**

```
<configuration>
  <system.web>
      <trace enabled="true"
      requestlimit="40" pageoutput="true"/>
  </system.web>
</configuration>
```

ASP.NET configuration uses hierarchical configuration architecture. All configuration information for an ASP.NET application is contained in configuration files named web.config that can be placed in the same directories as the application files. Child directories inherit the settings of the parent directories unless overridden by a web.config file in the child directory.

If a web.config file is present at the root directory of a Web server—for example, Inetpub\wwwroot—the configuration settings will apply to every application in that server.

The presence of a web.config file within a given directory or application root is optional. If a web.config file is not present, all configuration settings for the directory are automatically inherited from the parent directory. The root configuration file for all Web applications is named machine.config, and is found in the c:\winnt\Microsoft.NET\Framework\v<*version_number*>\Config folder.

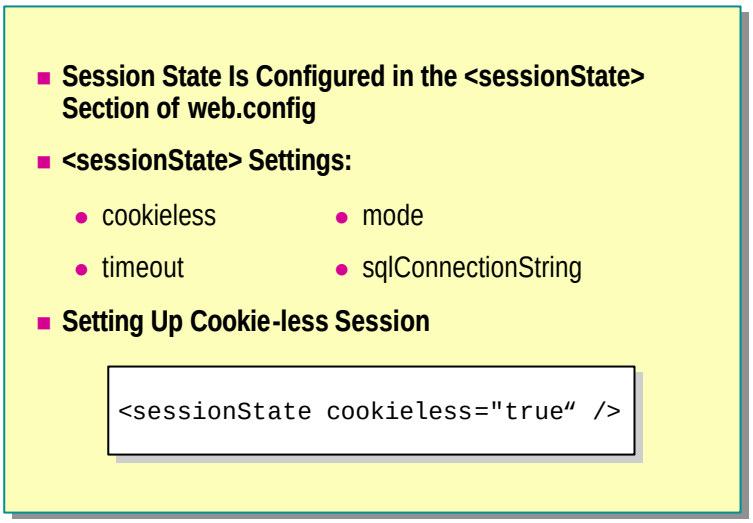
In a web.config file, there are sections for each major category of ASP.NET functionality, as shown in the following table.

Section name	Description
<browsercaps>	Responsible for controlling the settings of the browser capabilities component.
<compilation>	Responsible for all compilation settings used by ASP.NET.
<globalization>	Responsible for configuring the globalization settings of an application.
<httpmodules>	Responsible for configuring HTTP modules within an application. HTTP modules participate in the processing of every request into an application. Common uses include security and logging.
<httphandlers>	Responsible for mapping incoming URLs to **IHttpHandler** classes. Sub-directories do not inherit these settings. Also responsible for mapping incoming URLs to **IHttpHandlerFactory** classes. Data represented in <**httphandlerfactories**> sections are hierarchically inherited by sub-directories.
<iisprocessmodel>	Responsible for configuring the ASP.NET process model settings on Internet Information Services (IIS) Web Server Systems.
<authentication> **<identity>** **<authorization>**	Responsible for all security settings used by the ASP.NET security **HttpModule**.
<sessionState>	Responsible for configuring the session state **HttpModule**.
<trace>	Responsible for configuring the ASP.NET trace service.

ASP.NET configuration settings are represented within these configuration sections. For example, as you saw in Module 5, "Using Trace in Microsoft ASP.NET Pages," in Course 2063B, *Introduction to Microsoft ASP.NET*, you can turn the trace feature on for an entire application in the <trace> configuration section as follows:

```
<configuration>
  <system.web>
      <trace enabled="true"
      requestlimit="40" pageoutput="true"/>
  </system.web>
</configuration>
```

Setting Up Cookie-less Sessions



Session state features can be configured by the <sessionState> section in the web.config file. The <sessionState> section sets the behavior of the session state throughout the application. The following table lists the settings of the <sessionState> section and their descriptions.

Setting	Description
cookieless="[true/false]"	Indicates whether cookies should be used to store Session IDs. The default value is **False**. When the value is **False**, cookies are used.
mode="[off/InProc/ SQLServer/StateServer]"	Specifies where the session information is kept: in memory on the Web server; in a Microsoft SQL Server™ database; or in a separate process on the Web server or remote computer.
sqlConnectionString	Specifies the connection string for a SQL Server. For example, "data source=127.0.0.1;user id=sa; password=". This attribute is required when **mode** is set to **sqlserver**.
timeout="[number of minutes]"	Specifies the number of minutes a session can be idle before it is abandoned. The default value is 20 minutes.

For example, to double the default timeout of 20 minutes, the following can be added to the web.config of an application:

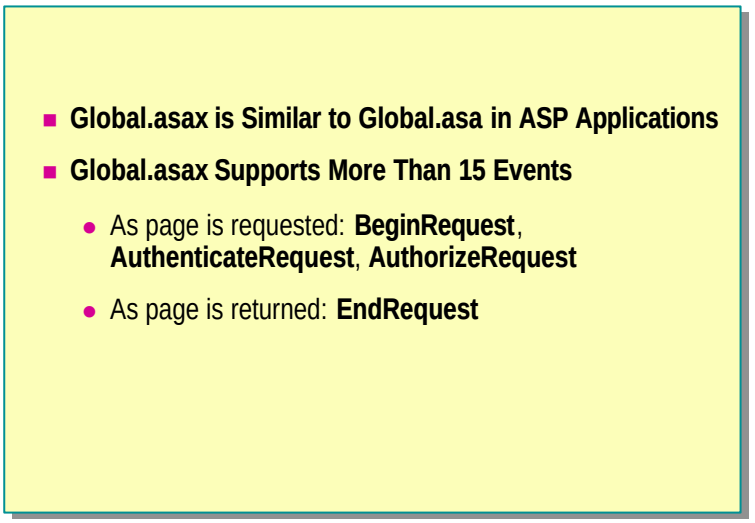
```
<sessionstate timeout="40" />
```

Setting Up a Cookie-less Session

By default, ASP.NET uses cookies to identify requests within a single session. If cookies are not available, or have been disabled by the user, a session can be tracked by adding a session identifier to the URL. You can enable cookie-less sessions as follows:

```
<sessionstate cookieless="true" />
```

Global Application File (Global.asax)



Similar to ASP, ASP.NET supports one global declarative file per application for application events and state called global.asax. The global.asax file is similar to ASP's global.asa, but has numerous new events that are not supported by global.asa.

Note In ASP.NET, global.asax can be used as an .asax file or as a component that can be deployed in the application's /bin directory.

Events

The event model provided by ASP.NET supports more than 15 events. This is different from global.asa, which had only Application and Session **OnStart** and **OnEnd** events.

For example, in ASP, if you wanted some code to run at the beginning of every page, you would have needed to use an Include file at the top of every ASP page. Using the ASP.NET global.asax file, you can simply declare the code in the **Application_BeginRequest** event procedure, which is called at the beginning of every request for the application.

```
Sub Application_BeginRequest(s As Object, e As EventArgs)
...
End Sub
```

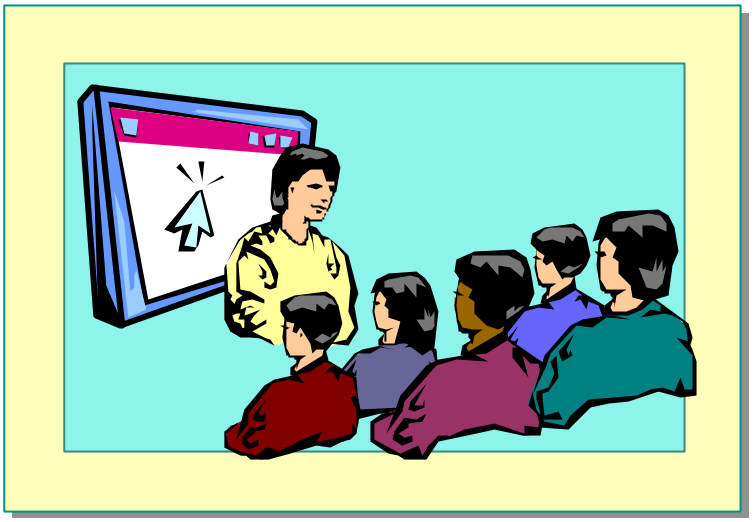
ASP.NET still supports the Application and Session **OnStart** and **OnEnd** event procedures, but Global.asax also includes events that are fired when a client requests a page. The following table lists events that can be used when a page is requested.

Event Name	Description
Application_Error	This event is fired when an un-handled error occurs within an application.
Application_BeginRequest	This event is fired whenever a new request is received.
Application_AuthenticateRequest	This event indicates that the request is ready to be authenticated.
Application_AuthorizeRequest	This event signals that the request is ready to be authorized.
Application_ResolveRequestCache	This event is used by the output cache module to stop the processing of requests that have been cached.
Application_AcquireRequestState	This event signals that per-request state should be obtained.
Application_PreRequestHandlerExecute	This event signals that the request handler is about to execute.

Global.asax also includes events that are fired when the requested page is sent back to the client. The following table lists these events.

Event Name	Description
Application_PostRequestHandlerExecute	This event is first available after the handler (such as an ASP.NET page), or Web service has completed its work.
Application_ReleaseRequestState	This event is called when the request state should be stored, because the application is finished with it.
Application_UpdateRequestCache	This event signals that code processing is complete and the file is ready to be added to the ASP.NET cache.
Application_EndRequest	This event is the last event called when the application ends.
Application_PreRequestHeaderSent	This event provides the opportunity to add, remove, or update headers and the response body.

Demonstration: Using Event Procedures in Global.asax



In this demonstration, you will see how to display events fired in the global.asax file.

↙ To run this demonstration

1. Copy the file *<install folder>*\Democode\Mod07\global.asax to the folder *<install folder>*.

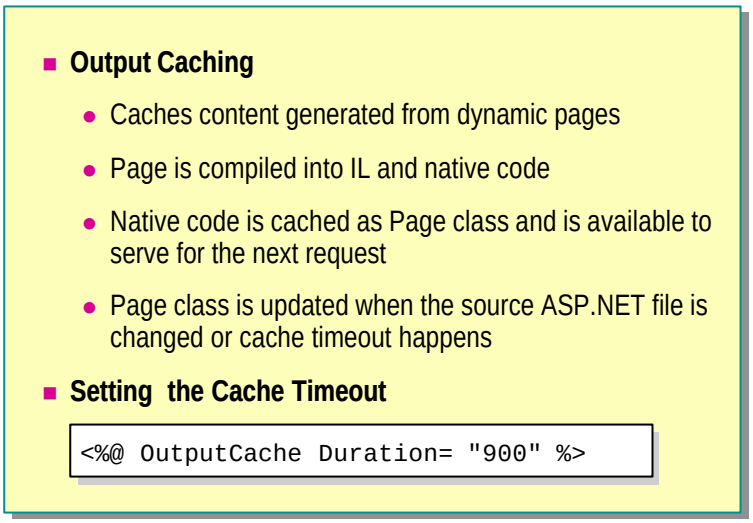
   This is the root of the 2063 virtual directory.

2. Open the file global.asax. There are **Response.Write** statements in the event procedures.

3. View http://localhost/2063/DemoCode/Mod07/GlobalEvents.aspx in Microsoft Internet Explorer.

4. Delete the global.asax file from the root of the 2063 virtual directory.

Page Caching



Page caching allows you to cache dynamic content. When an ASP.NET page is accessed for the first time, the page is compiled into Intermediate Language (IL) and to native code. This native code is cached as **Page class** and is available to serve the next request. This cached **Page class** is updated and rebuilt when the source ASP.NET file is changed or the cache timeout is reached.

Setting the Cache Timeout Value

You can specify the cache timeout value by setting the output cache page directive. For example, to cache an ASP.NET page for 15 minutes, add the following @OutputCache directive to the .aspx page:

```
<%@OutputCache Duration="900" %>
```

The unit for time for the **Duration** attribute is seconds.

This cache option is very useful for pages that do not change often (or do not change within a known time period).

Note Creating an output cache for an application should be your final task in application development. Otherwise, when you debug your pages, instead of getting new and modified pages, you may get old pages that are stored in the output cache.

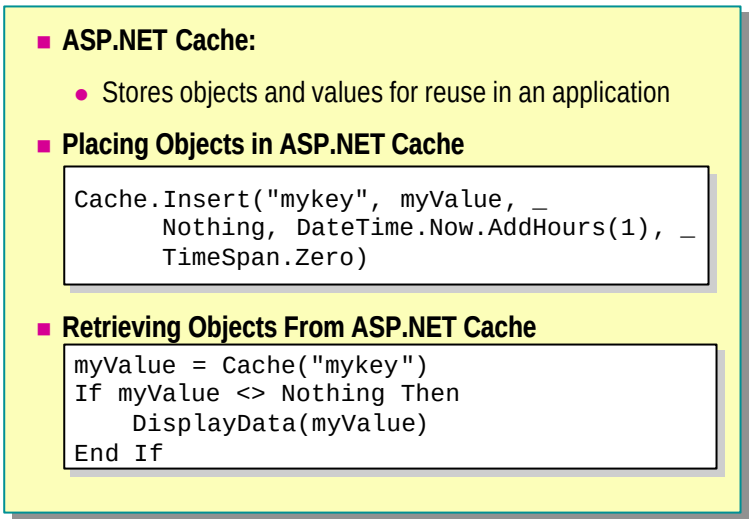
◆ Sharing Information Between Pages

- Using ASP.NET Cache
- Using web.config Variables
- Demonstration: Using web.config Variables
- Using Session and Application Variables
- Demonstration: Using Session Variables
- Saving Session and Application Variables in a Database
- Discussion: Different Ways of Sharing Information

It is often necessary to share information between the pages of a Web application. The information being shared can be either static or dynamic.

In this section, you will learn how to share both static and dynamic data between pages of a Web application.

Using ASP.NET Cache



The ASP.NET cache can be used to store objects and values that you reuse in your application. ASP.NET provides a full-featured cache engine that can be used by pages to store and retrieve arbitrary objects across HTTP requests. The ASP.NET cache is private to each application and stores objects in memory. The lifetime of the cache is equivalent to the lifetime of the application. This means that, when the application is restarted, the cache is recreated.

The ASP.NET cache also provides a way to pass values between pages within the same application. The ASP.NET cache methods implement automatic locking; therefore, it is safe for objects to be accessed concurrently from more than one page. The only drawback is that another page may change the values that you place in the cache. It is not a per-user cache.

The cache provides a simple dictionary interface that allows you to insert objects easily and retrieve them later. In the simplest case, placing an item in the cache is exactly like adding an item to a dictionary:

```
Cache("mykey") = myValue
```

Retrieving this data is equally simple:

```
myValue = Cache("mykey")
If myValue <> Nothing Then
    DisplayData(myValue)
End If
```

You can also supply parameters when inserting an item into the cache by using the **Insert** method.

Cache.Insert Syntax

```
Cache.Insert ( key As String, _
   value As Object, _
   dependencies As CacheDependency, _
   absoluteExpiration As DateTime, _
   slidingExpiration As TimeSpan )
```

- **key** is the cache key used to reference the object.
- **value** is the object to be cached.
- **dependencies** should be set to **Nothing**.
- **absoluteExpiration** is the time at which the cached object expires and is removed from the cache.
- **slidingExpiration** is the interval between the time the cached object was last accessed and when that object is scheduled to expire. For example, if **slidingExpiration** is set to 20 minutes, the object will expire and be removed from the cache 20 minutes after the object was last accessed.

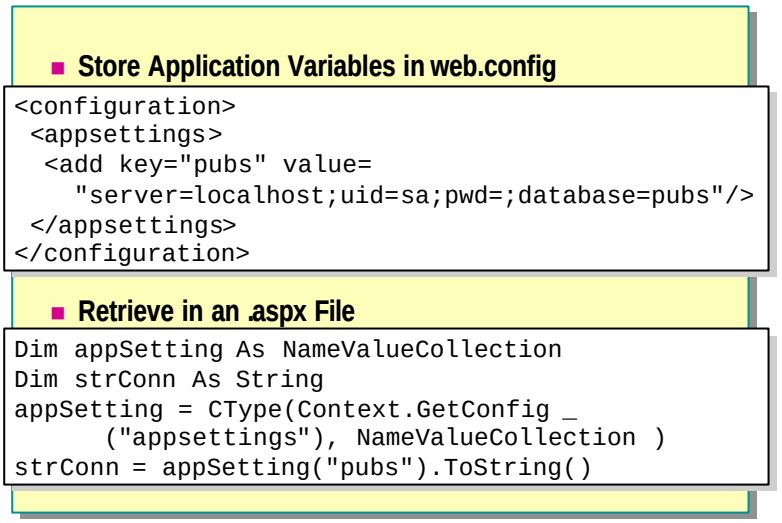
The following code shows how to use the ASP.NET cache to store a **DataView** that is used in an ASPX page. The first time the page is requested, it reads data from the database and then stores it in the ASP.NET cache. At every subsequent request for the page, the **DataView** is retrieved from the cache, eliminating a call to the database. This example uses absolute expiration to set the cache to expire one hour from the first time it was accessed.

```
Dim dvMenuItems As DataView
dvMenuItems = Cache("dvMenuItems")

If (dvMenuItems = Nothing) Then
  Dim products As New Conference.ProductsDB
  dvMenuItems = products.GetProductCategories(). _
      Tables(0).DefaultView
  Cache.Insert("dvMenuItems", dvMenuItems, Nothing, _
      DateTime.Now.AddHours(1), TimeSpan.Zero)
End If

MyList.DataSource = dvMenuItems
MyList.DataBind()
```

To set the cache to use sliding expiration and expire one hour after the last time it was accessed, apply the following parameters to the **Insert** method:

```
Cache.Insert("dvMenuItems", dvMenuItems, Nothing, _
  DateTime.Now, TimeSpan.FromHours(1))
```

Using web.config Variables



You can use the <appsettings> section of the web.config file as a repository for application settings. In the <appsettings> section, you can create key/value pairs for data that is commonly used throughout your application. This is very useful because you can define all application configuration data in a central location. For example, you can store a database connection string for an application in a central location, instead of having it in each ASP.NET page.

The following web.config file creates two key/value pairs for the connection strings for the databases used in the application:

```
<configuration>
  <appsettings>
      <add key="pubs"
      value="server=localhost;uid=sa;pwd=;database=pubs" />
      <add key="northwind"
      value="server=localhost;uid=sa;pwd=;database=northwind"
          />
  </appsettings>
</configuration>
```

Use the **GetConfig** method of the **Context** object to read the data from the web.config file. You need to supply the name of the section and the name of the key to retrieve. The **GetConfig** method returns a **NameValueCollection** variable containing the key/value pairs defined in the requested section.

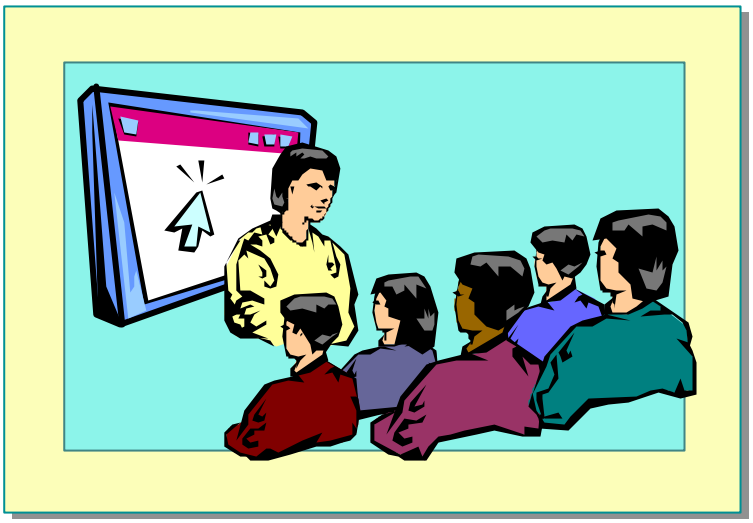
For example, the following sample code reads the value of the pubs key from the <appSettings> section.

```
Dim appSetting As NameValueCollection
Dim strConn As String
appSetting = CType(Context.GetConfig("appsettings"), _
  NameValueCollection)
strConn = appSetting("pubs").ToString()
```

You can also use the **GetConfig** method of the **HttpContext** object to access configuration information from a component. The syntax for this is:

```
appSetting = _
  CType(HttpContext.Current.GetConfig("appsettings"), _
  NameValueCollection)
```

Demonstration: Using web.config Variables



In this demonstration, you will see how to create a constant in web.config and use the constant from different ASP.NET pages.

↙ To run the demonstration

1. Create a new web.config file in the folder *<install folder>*\DemoCode\Mod07.
2. Add the <configuration> tag with an <appsettings> section.
3. Add a new key to the <appsettings> section:

```
<configuration>
   <appSettings>
      <add key="intNumberInConfig" value="9" />
   </appSettings>
</configuration>
```

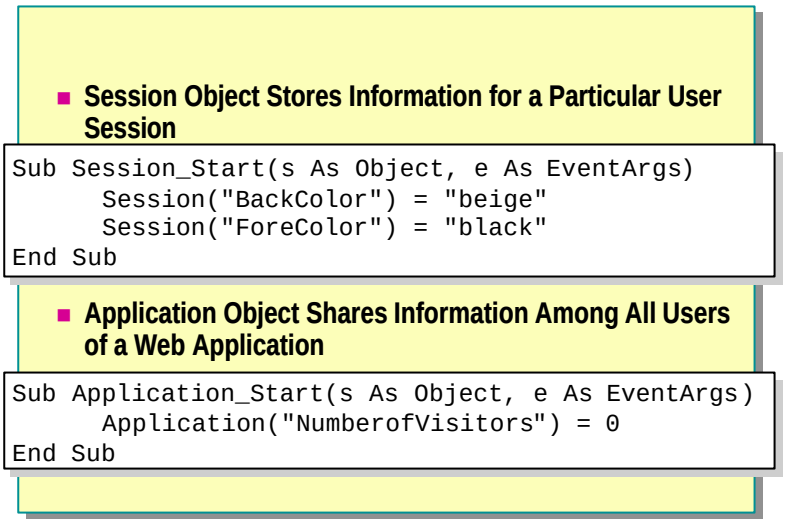
4. Open the file *<install folder>*\DemoCode\Mod07\UsingConfigVar1.aspx.

   The **intNumberInConfig** value is retrieved and displayed in the **Page_Load** event procedure.
5. View the UsingConfigVar1.aspx page in Internet Explorer.

   The value of intNumberInConfig, 9, is displayed.
6. Click **Next**.

   The UsingConfigVar2.aspx page opens and displays the same value.

Using Session and Application Variables



You can use session and application variables to share information between pages of an ASP.NET application.

You generally initialize session and application variables in the **Start** event procedures of the **Session** and **Application** objects in the global.asax file.

Session Variables

You use the **Session** object to store information that is needed for a particular user session. Variables stored in the **Session** object will not be discarded when the user goes between pages in the Web application. Instead, these variables will persist for the entire user session. The following example illustrates how session variables are used to store information about a particular user session.

```
<script language="VB" runat="server">
Sub Session_Start(S As Object, E As EventArgs)
   Session("BackColor") = "beige"
   Session("ForeColor") = "black"
   Session("FontName") = "verdana"
End Sub
</script>
```

Application Variables

You can use the **Application** object to share information among all users of a Web application. An **Application** object is created when the first user of the application requests an .aspx file. It is destroyed when all users have exited the application and the application is unloaded.

For example, you might store the total number of visitors to a Web site in an application-level variable.

```
Sub Application_Start(s As Object, e As EventArgs)
  Application("NumberofVisitors") = 0
End Sub
```

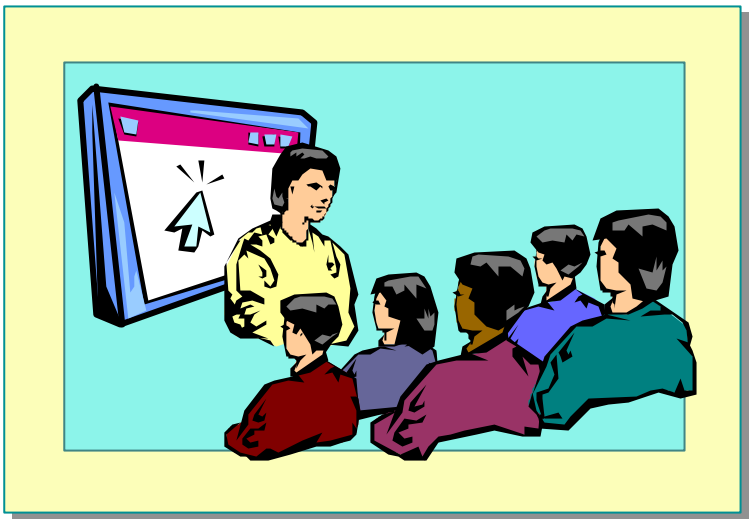
Note A disadvantage of using session and application variables is that the memory occupied by these variables will not be released until the value is either removed or replaced. For example, keeping seldom-used 10-megabyte (MB) recordsets in application-state permanently is not the best use of system resources.

Reading Session and Application Variables

To use a session or application variable in an ASP.NET page, simply read the value from the **Session** or **Application** object.

```
Session("BackColor")
Session("ForeColor")
Session("FontName")
Application("NumberofVisitors")
```

Demonstration: Using Session Variables



In this demonstration, you will learn how to share the same session variable through two different ASP.NET pages.

↙ To run the demonstration

1. Copy the file <*install folder*>\Democode\Mod07\global.session to the root of the 2063 virtual folder and rename it global.asax.
2. Open the file global.asax and initialize a session variable named **intNumber** to **3** by adding the following code to the **Session_Start** event procedure:

   ```
   Session("intNumber")=3
   ```

3. Open the file <*install folder*>\Democode\Mod07\UsingSessionVar1.aspx.

   The session variable is retrieved and displayed in the **Page_Load** event procedure.

4. View the page http://localhost/2063/democode/Mod07/UsingSessionVar1.aspx in Internet Explorer.

   The value of the session variable, 3, is displayed.

5. Click **Next**.

   The UsingSessionVar2.aspx page opens, increments the session variable by 4, and displays the new value, 7.

6. Click **Back** to return to the UsingSessionVar1.aspx, which displays the new value of the session variable.

Saving Session and Application Variables in a Database

- **Session State Can Be Stored in a Separate SQL Server Database**
- **A Temporary Table Is Used for the Serialized Session Data**
- **Advantages:**
  - State can be recovered even if an application crashes
  - An application can be partitioned across multiple Web farm machines

---

In ASP.NET, an application can store session state in a separate SQL Server database. A temporary table is used for the serialized session data. The table can be accessed by a combination of stored procedures and managed data access components for SQL Server.

By cleanly separating the storage of session data from the application usage of it, ASP.NET provides advantages that did not exist in ASP:

- Ability to recover from application crashes

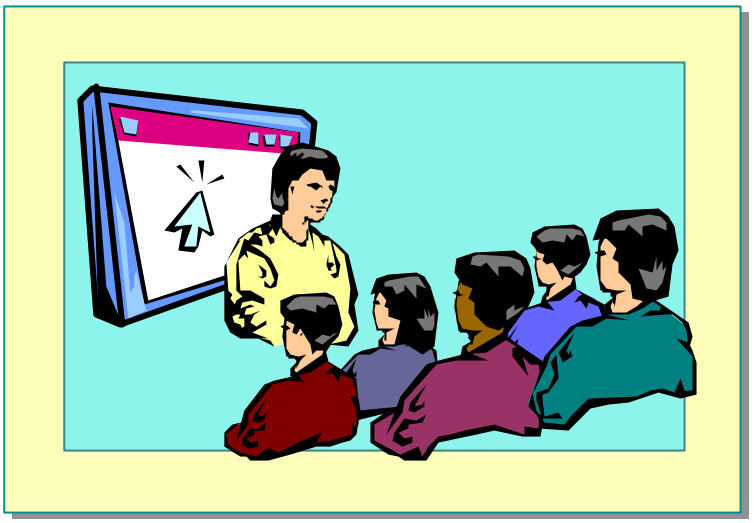
  Because all state is stored externally from an individual worker process, it will not be lost if the process crashes or is forcibly restarted.
- Ability to partition an application across multiple Web farm servers

  Because all state is stored externally from worker processes, you can partition an application across multiple worker processes running on multiple servers. The model for communicating state between a worker process and a state service running on the same server is almost identical to models in which the state service is located on a completely different machine.

To configure your application to store session state in a separate database, set the **mode** attribute of the **sessionState** section to **SQLServer** and set the **sqlConnectionString** attribute to the connection string for the server.

For more information on saving session information in a database, see “Session State” in the Microsoft .NET Framework software development kit (SDK) documentation.

Discussion: Different Ways of Sharing Information



Securing an ASP.NET Application

- What Is Authentication?
- Forms Authentication Architecture
- Setting Up Security in web.config
- Creating a Login Form
- Demonstration: Setting Up Security in web.config

Securing Web sites is a critical issue for Web developers. A secure system requires careful planning, and Web site administrators and programmers must have a clear understanding of the options for securing a site. Security, in the context of an ASP.NET application, involves performing three fundamental functions for all requests: authentication, authorization, and impersonation.

Authentication is the process verifying that the user requesting a page is actually that user. To authenticate a request, you accept credentials from a user and validate those credentials against some authority.

After you have authenticated the user, the authorization process determines whether that user should be granted access to a given resource.

Another important feature of server applications is the ability to control the identity under which server application code is executed. When a server application executes code with the identity of the requesting entity, this is known as impersonation.

In this section, you will learn about securing an ASP.NET application by implementing authentication and authorization in web.config.

What Is Authentication?

- **Authentication**
  - Accept credentials from a user
  - Validate the credentials
- **Types of ASP.NET Authentication**
  - Windows authentication
  - Passport authentication
  - Forms authentication

Authentication is the process of accepting credentials from a user and validating those credentials against some authority. If the credentials are valid, you have an authenticated identity.

Authentication in ASP.NET is implemented through authentication providers. ASP.NET authentication providers are the code modules that contain the code necessary to authenticate the requestor's credentials.

The first version of ASP.NET will ship with support for the following authentication providers:

- Windows authentication

  This method is used in conjunction with the IIS authentication. Authentication is performed by IIS in one of three ways: Basic, Digest, or Integrated Windows Authentication. The advantage of using this authentication type is that it requires minimum coding. Also, because it is Microsoft Windows NT® authentication, there is no additional cost to implement this type of security. However, because the resources for authentication are not embedded in the ASP.NET application, using this method involves additional effort when deploying the application.

  For more information on Windows authentication, see the Microsoft .NET Framework SDK documentation.

- Passport authentication

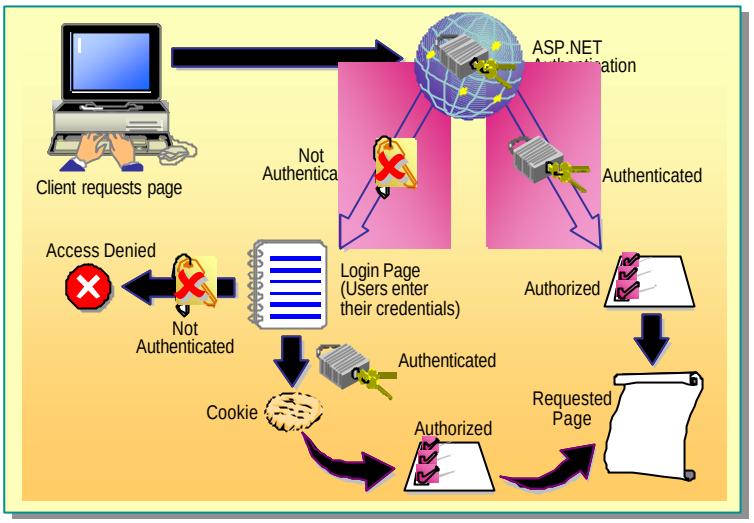
  Passport authentication is a centralized authentication service provided by Microsoft that offers a single sign-in and core profile services for member sites. It is a Web service and an integral part of the Microsoft .NET Framework.

  For more information on Passport authentication, go to: http://www.passport.com/Business

- Forms authentication

  Forms authentication refers to a system where unauthenticated requests are redirected to a Hypertext Markup Language (HTML) form (by using HTTP client-side redirection). The user provides credentials and submits the form. If the application validates credentials on the form, the system issues a cookie to the user. Subsequent requests are issued with the cookie in the request headers; therefore, they are authenticated.

Forms Authentication Architecture



The most commonly used authentication type in ASP.NET applications is forms authentication.

A request for a page protected by forms authentication must still go through IIS first. Therefore, you must set IIS authentication to Anonymous Access. This allows all requests to get to ASP.NET before being authenticated.

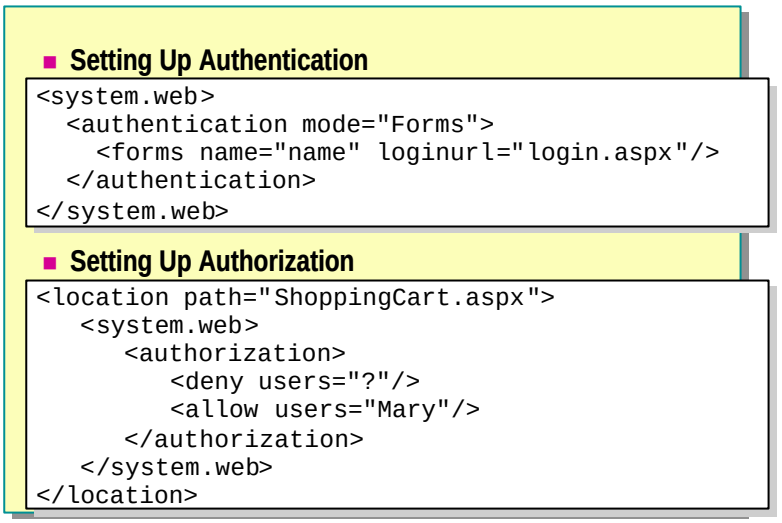
Tip For more information on setting up Anonymous Access, see the Security section in the IIS documentation.

The following is the set of events that take place during forms authentication:

1. A client generates a request for a protected page.
2. IIS receives the request, and passes it to the ASP.NET application. Because the authentication mode is set to Anonymous Access, IIS authentication is not used.
3. ASP.NET checks to see if a valid authentication cookie is attached to the request. If it is, this means that the user's credentials have already been confirmed, and the request is tested for authorization. The authorization test is performed by ASP.NET and is accomplished by comparing the credentials contained in the request's authorization cookie to the authorization settings in the application's configuration file (web.config). If the user is authorized, access is granted to the protected page.
4. If there is no cookie attached to the request, ASP.NET redirects the request to a login page (the path of which resides in the application's configuration file), where the user enters the required credentials, usually a name and password.

5. The application code on the login page checks the credentials to confirm their authenticity and, if authenticated, attaches a cookie containing the credentials to the request. If authentication fails, the request is returned with an "Access Denied" message.
6. If the user is authenticated, ASP.NET checks authorization as in step 3, and can either allow access to the originally requested, protected page or redirect the request to some other page, depending on the design of the application. Alternatively, it can direct the request to some custom form of authorization where the credentials are tested for authorization to the protected page. Usually if authorization fails, the request is returned with an "Access denied" message.

Setting Up Security in web.config



Security for ASP.NET applications is set up in the application's web.config file. The security settings in web.config are included in the <authentication>, <authorization>, and <identity> sections.

Note You can set up authentication only in the web.config file in the root of your Web application.

Setting Up Authentication

Begin by setting the authentication method for the application in an <authentication> subsection of the <system.web> section, as shown in the following example.

```
<configuration>
  <system.web>
      <authentication mode="authmode">
      </authentication>
  </system.web>
</configuration>
```

As mentioned earlier in this module, ASP.NET supports three types of authentication: Windows authentication, Forms authentication, and Passport authentication. You can set the authentication mode for an application by setting the **mode** attribute in the <authentication> tag to "None", "Windows", "Passport", or "Forms".

If you set the authentication mode to “Forms”, you need to add a <forms> element to the <authentication> section, as shown in the following example:

```
<system.web>
  <authentication mode="Forms">
      <forms name=".namesuffix" loginurl="login.aspx" />
  </authentication>
</system.web>
```

In the <forms> section, you configure settings of the cookie. Set the **name** attribute to the suffix to be used for the cookies and the **loginUrl** attribute to the URL of the page to which un-authenticated requests are redirected.

Setting Up Authorization

After specifying the authentication mode, you need to either mark the entire Web application as needing authorization or specify which pages are secure and require authorization.

Securing an Entire Application

To mark the entire application as secure, create an <authorization> section in the <authentication> section, as illustrated in the following code example:

```
<system.web>
  <authentication mode="Forms">
      <forms name=".aspxauth"
          loginurl="login.aspx" />
      <authorization>
      </authorization>
  </authentication>
</system.web >
```

Securing Specific Pages

To mark only specific pages as secure, create a <location> section with <system.web> and <authorization> sub-sections for each secure page in your Web application:

```
<location path="ShoppingCart.aspx">
  <system.web>
      <authorization>
      </authorization>
  </system.web>
</location>
```

Any configuration settings contained in the <location> section will be directed at the file or directory indicated in the **path** attribute. There can be multiple <location> sections in the <configuration> section.

In the <system.web> section, you create an <authorization> subsection to specify what type of authorization will be enforced. Create <allow> or <deny> tags to allow or deny users access to a page. Within these tags, "?" indicates anonymous users, whereas "*" means all users. For example, the following code denies access to all anonymous users.

```
<authorization>
  <deny users="?"/>
</authorization>
```

The following code allows the user "Mary" access to a page:

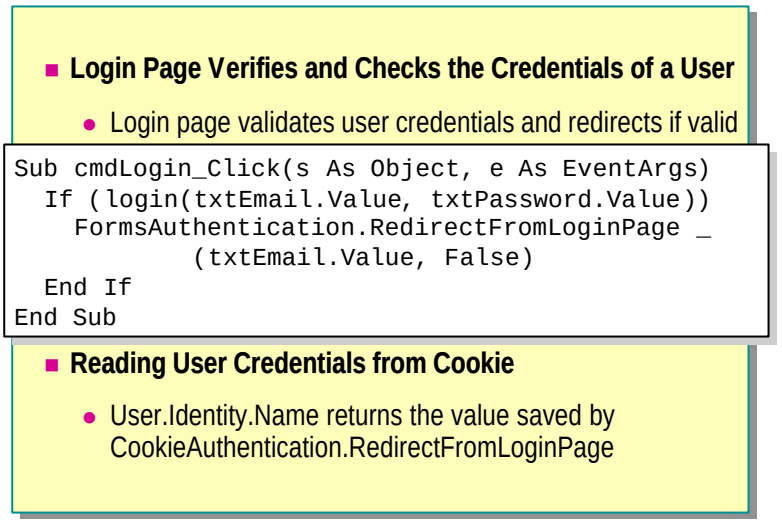
```
<authorization>
  <allow users="Mary"/>
</authorization>
```

The following example denies all anonymous users access to the ShoppingCart.aspx page.

```
<location path="ShoppingCart.aspx">
  <system.web>
      <authorization>
          <deny users="?"/>
      </authorization>
  </system.web>
</location>
```

For more information about setting up authorization in web.config, see "ASP.NET Authorization" in the Microsoft .NET Framework SDK documentation.

Creating a Login Form



During authentication, all requests are redirected to the login page specified in the **loginurl** attribute of the <cookie> tag. The login page verifies and checks the credentials of a user.

How Does a Login Page Work?

If the authentication mode is set to "Forms", ASP.NET looks for a cookie attached to a request for a secure page. If it does not find one, it redirects the request to a specified login page.

On the login page, the user enters the required credentials. The page checks the entered credentials either through application-specific code or by calling **FormsAuthentication.Authenticate**. If the credentials are valid, a cookie is generated and the user is redirected to the originally requested page by calling **FormsAuthentication.RedirectFromLoginPage**. However, if the credentials are not valid, the user stays on the login page and is given a message that indicates that the login credentials are invalid.

The **RedirectFromLoginPage** method takes two parameters, **userName**, which specifies the name of the user for forms authentication purposes, and **createPersistentCookie**. If the value of **createPersistentCookie** is **True**, a cookie is created on the user's machine.

The following table lists all the methods of the **FormsAuthentication** object, which can be used in the authentication process.

Method	Function
Authenticate	Given the supplied credentials, this method attempts to validate the credentials against those contained in the configured credential store.
GetAuthCookie	Creates an authentication cookie for a given user name. This does not set the cookie as part of the outgoing response, so that an application can have more control over how the cookie is issued.
GetRedirectUrl	Returns the redirect URL for the original request that caused the redirect to the login page.
RedirectFromLoginPage	Redirects authenticated users back to the original URL they requested.
SetAuthCookie	Creates an authentication ticket for the given userName and attaches it to the cookies collection of the outgoing response. It does not perform a redirect.
SignOut	Given an authenticated user, calling **SignOut** removes the authentication ticket by doing a **SetCookie** with an empty value. This removes either durable or session cookies.

Creating a Login Page

A login page is simply an ASP.NET page with an HTML form, a **Submit** button, and a **Click** event procedure for the **Submit** button.

The following is an example of a form on a login page:

```
<form runat=server>
  Email:<input id="txtEmail" type="text" runat=server/>
  <BR>
  Password<input id="txtPassword" type="password"
      runat="server"/>
  <BR>
  <asp:button text="Login" OnClick="Login_Click"
      runat="server" />
  <asp:label id="lblMsg" runat="server"/>
</form>
```

In the **Click** event procedure of the **Submit** button, you validate the information entered in the form, then call **FormsAuthentication.RedirectFromLoginPage** if it is valid. The **RedirectFromLoginPage** method issues the cookie and then redirects the user to the originally requested page. The following sample code uses a custom function named **Login** to validate the username and password, and then calls **RedirectFromLoginPage** if they are valid:

```
<script language="VB" runat=server>
Sub cmdLogin_Click(Sender As Object, E As EventArgs)
  Dim strCustomerId As String
  'Validate User Credentials
  strCustomerId = Login(txtEmail.Value, txtPassword.Value)

  If (strCustomerId <> "") Then
      FormsAuthentication.RedirectFromLoginPage _
          (strCustomerId, False)
  Else
      lblMsg.Text = "Invalid Credentials: Please try again"
  End If
End Sub
</script>
```

Reading Credentials from Cookie

After a user has been authenticated, you can obtain the user name of the authenticated user programmatically by using the **User.Identity.Name** property. This is useful to build an application that uses the user's name as a key to save information in a database table or directory resource.

Using Credentials from web.config

If you do not want to write your own validation function, you can create a list of users in the web.config file and use the **FormsAuthentication.Authenticate** method to validate a username and password pair.

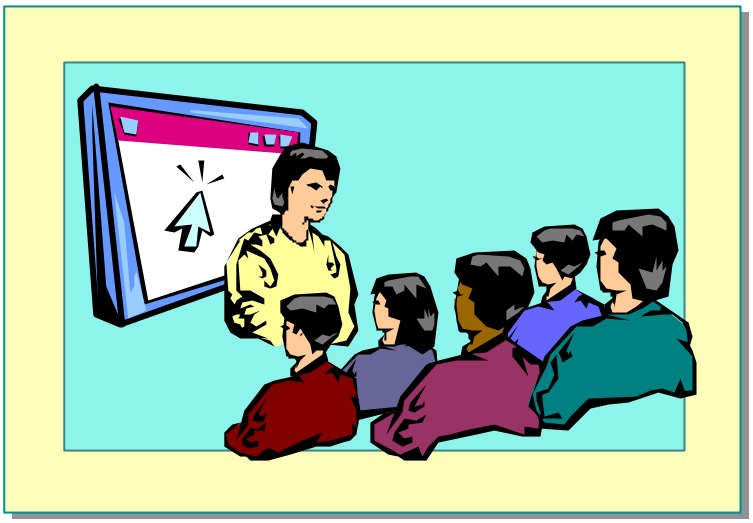
The following web.config file creates two valid users for an application:

```
<system.web>
  <authentication mode="Forms">
      <forms loginurl="democode/Mod07/LoginDemo.aspx">
          <credentials passwordFormat="Clear">
              <user name="ruth" password="password"/>
              <user name="bob" password="password"/>
          </credentials>
      </forms>
  </authentication>
</system.web>
```

The following event procedure on the login page calls the **Authenticate** method to authenticate only those users from the web.config file:

```
Sub cmdLogin_Click(S As Object, E As EventArgs)
  If (FormsAuthentication.Authenticate _
      (txtEmail.value, txtPassword.Value))
      FormsAuthentication.RedirectFromLoginPage _
          (txtEmail.Value, False)
  Else
      lblMsg.Text = "Invalid Credentials: Please try again"
  End If
End Sub
```

Demonstration: Setting Up Security in web.config



In this demonstration, you will learn how to set up authentication in web.config and how to create an authentication cookie in a login file.

↙ To run the demonstration

1. Copy the files <*install folder*>\Democode\Mod07\config.demo, LoginDemo.aspx, SecurePageDemo1.aspx, SecurePageDemo2.aspx, UsingSessionVar1.aspx, and UsingSessionVar2.aspx to the root of the 2063 virtual directory.
2. Rename the file config.demo to web.config.
3. Edit the file web.config.

   There is an <authentication> section that redirects all unauthenticated requests to the LoginDemo.aspx page.

   Two pages have been set up as secure pages.
4. Edit the file <*install folder*>\LoginDemo.aspx.

   The **cmdLogin_Click** event procedure validates the user name and password and calls **RedirectFromLoginPage** if they are valid.
5. View the unsecured page http://localhost/2063/UsingSessionVar1.aspx page in Internet Explorer.
6. View the secure page http://localhost/2063/SecurePageDemo1.aspx page in Internet Explorer.

   You are redirected to the LoginDemo.aspx page.

7. Type an e-mail name and an invalid password (“password” is the only valid password) and click **Sign In Now**.

   It will fail because the password is invalid.

8. Type **password** for the password field and click **Sign In Now** again. You will be redirected to the SecurePageDemo1.aspx page.

9. View the secure page http://localhost/2063/SecurePageDemo2.aspx in Internet Explorer.

Lab 7: Creating an ASP.NET Web Application

Objectives

After completing this lab, you will be able to:

- Use forms authentication to authenticate and authorize users when accessing secured pages.
- Use the ASP.NET cache to save data for the _menu.aspx page.
- Maintain state of an application by saving data in a database.

Prerequisite

Before working on this lab, you must know how to use a component.

Lab Setup

There are starter and solution files associated with this lab. The starter files are in the folder <*install folder*>\Labs\Lab07\Starter and the solution files for this lab are in the folder <*install folder*>\Labs\Lab07\Solution.

Scenario

In Exercise 1, you will use the ASP.NET cache to store a **DataSet** that is used by the _menu.aspx page.

When users want to sign into the conference system, they need to enter a valid e-mail name and corresponding password. This is done on the login.aspx page or through register.aspx in the case of a new user. In this lab, you will first complete the login.aspx and register.aspx pages to attempt to authenticate a user through the use of a cookie (forms authentication). Exercises 2 and 4 involve variations on forms authentication of this user. Exercise 3 illustrates how to retrieve the customer ID of the authenticated user from any of the pages of your application. You can use this ID to store customer information or products selected by the customer in the database.

Estimated time to complete this lab: 60 minutes

Exercise 1
Storing Data

In this exercise, you will programmatically cache the **DataView** used by the **DataList** control in the file _Menu.ascx. You will then look at web.config to see how the connection string for the Conf database is stored.

↙ To read the DataView from the cache

1. Open the file _Menu.ascx file in the folder InetPub\wwwRoot\ASPNET.
2. At the beginning of the **Page_Load** event procedure, immediately after the **dvMenuItems** variable is declared, add code to read the contents of the dvMenuItems item from the cache and save it in the **dvMenuItems** variable.

   Your code should look like the following:

   ```
   dvMenuItems = Cache("dvMenuItems")
   ```

3. Test the contents of dvMenuItems. If the content is empty:
   a. Create the **DataView** from the database by using the code that is already in the event procedure.
   b. Store the **DataView** in the cache. Create the cache such that it will be stored for one hour.

   Your code should look like the following:

   ```
   If (dvMenuItems Is Nothing) Then
      Response.Write ("Getting items from the database.<P>")
      Dim products As New Conference.ProductsDB

      dvMenuItems = products.GetProductCategories(). _
          Tables(0).DefaultView
      Cache.Insert("dvMenuItems", dvMenuItems, Nothing, _
          DateTime.Now.AddHours(1), TimeSpan.Zero)
   Else
      Response.Write ("Got items from the cache.<P>")
   End If
   ```

↙ To save and test your work

1. Save your changes to _Menu.ascx.
2. By using Internet Explorer, go to the home page of the ASPNET Web site by viewing http://localhost/ASPNET/default.aspx

   The items are retrieved from the database.
3. Refresh the page in Internet Explorer.

   The items are retrieved from the cache.

↙ To investigate web.config application settings

As you can see in the _Menu.ascx file, the **GetProductCategories()** method is used to get the items for the menu from the database. This method is in the **ProductsDB.vb** component.

1. Open the ProductsDB.vb file from the Components folder and locate the **GetProductCategories** method.

   The SQL connection string is obtained from ConferenceDB.ConnectionString.

2. Open the ConferenceDB.vb file from the Components folder and locate the **ConnectionString** property.

   The connection string is read from the DSN key in the <appSettings> section of the web.config file.

3. Open the web.config file in the ASPNET Web site and locate the DSN key in the <appSettings> section.

Exercise 2
Using Forms Authentication

In this exercise, you will validate the user's name and password from the login.aspx page with information stored in the database. If the information is valid, you will transfer the contents of the temporary shopping cart into a permanent shopping cart for the user. Next, you will create an authentication cookie that will be used throughout the Web site.

You will use two different classes in the conference component: **Conference.ShoppingCartDB** manipulates the shopping cart and **Conference.CustomersDB** accesses the customer information in the Conf database.

Note The source code for these classes can be found in the \Components\ShoppingCartDB.vb and \Components\CustomerDB.vb files in the ASPNET Web site.

You will call the following methods:

- **Conference.ShoppingCartDB.GetShoppingCartID()** returns a string with the ID of the current shopping cart.

  ```
  Public Function GetShoppingCartId() As String
  ```

- **Conference.ShoppingCartDB.MigrateCart()** migrates the items from one shopping cart (temporary) to another (permanent). The permanent shopping cart uses the customer's ID as the shopping cart ID.

  ```
  Public Sub Procedure MigrateCart(oldCartId As String,
  NewCartId As String)
  ```

- **Conference.CustomersDB.Login()** validates an e-mail name and password pair against credentials stored in the customers table of the database. If the e-mail name and password pair is valid, the method returns the customer ID. In the case of an invalid e-mail name and password pair, this method returns an empty string.

  ```
  Public Function Login(email As String, password As String)
  As String
  ```

↙ To connect to a data source

1. Open the file Login.aspx in the folder InetPub\wwwRoot\ASPNET.
2. In the **cmdLogin_Click** event procedure, declare the variables as shown in the following table.

Variable name	Data Type
shoppingCart	New Conference.ShoppingCartDB
strTempCartID	String
accountSystem	New Conference.CustomersDB
strCustomerID	String

3. Call the **Login** method of the **accountSystem** object and store the returned customer ID in the variable **strCustomerID**. Pass the e-mail name and password to the **Login** method:

   Your code should look like the following:

```
strCustomerId = _
  accountSystem.Login(txtEmail.Value, txtPassword.Value)
```

4. If the returned CustomerID is empty, this indicates that the user is not authorized to log on. Output a "login failed" message in the spnInfo label.
5. If the returned CustomerID is not empty, the user is authorized to log on and you need to migrate the contents of the temporary shopping cart to a permanent one by performing the following operations:

   a. Get the shopping cart ID by using the **GetShoppingCartId** method of the **shoppingCart** object and store it in the **strTempCartID** variable.

   b. Migrate any existing temporary shopping cart items to the permanent shopping cart by using the **MigrateCart** method of the **shoppingCart** object. Pass the temporary shopping cart ID and the customer ID to the method.

   c. Create a non-persistent cookie and direct the user back to the Login page by calling the **RedirectFromLoginPage** method of the **FormsAuthentication** object.

   Your code should look like the following:

```
If (strCustomerId <> "") Then
   ' Get the old Shopping Cart ID
   strTempCartID = shoppingCart.GetShoppingCartId()

   ' Migrate existing shopping cart into
   'permanent shopping cart
   shoppingCart.MigrateCart(strTempCartID, strCustomerId)

   ' Redirect browser back to originating page
   FormsAuthentication.RedirectFromLoginPage( _
                              strCustomerId, false)
Else ' Login failed
   spnInfo.innerHTML = "Login Failed"
End If
```

↙ To save and test your work

1. Save your changes to login.aspx.
2. By using Internet Explorer, go to the Login page of the ASPNET Web site by viewing http://localhost/ASPNET/Login.aspx
3. Fill out the e-mail field with someone@microsoft.com with a blank password and click **Sign In Now**.

   You will get the **Login Failed** message.
4. Fill in the password field with the value **password** and click **Sign In Now**.

   You will be redirected to the default page of the ASPNET Web site, default.aspx.

Note The e-mail name and password pair someone@microsoft.com/password is already in the database.

Exercise 3
Retrieving Cookie Data in ASP.NET Pages

In this exercise, you will add code to the GetID.aspx page that reads the customer ID (also known as Cart ID) from the authentication cookie. If the customer has not been authenticated, you will use a temporary ID. The GetID.aspx page is not directly linked to the ASPNET Web site. Instead, it simply reads the customer ID or temporary ID and displays it.

↙ To read the customer ID

1. Add the GetID.aspx page to the ASPNET Web site.
   a. On the **Project** menu, click **Add Existing Item**.
   b. In the **Add Existing Item** dialog box, navigate to the <*install folder*>\Labs\Lab07\Starter folder.
   c. Select the GetID.aspx file, and then click **Open**.

   Note You will be asked if you want to create a new class file. Click **No**.

2. Open the GetID.aspx file.
3. Locate the comment:

   ```
   ' TO DO: Read the customer ID
   ```

4. Test the value of **User.Identity.Name**.
5. If the string is not empty (""), output the value of **User.Identity.Name** using **Response.write**.

6. If the string is empty, the user has not previously been authenticated.
   a. Read the value of the **Conference_CartID** cookie. This cookie is issued the first time a non-authenticated user accesses the shopping cart.
   b. If the cookie is not empty, output the value of the **Conference_CartID** cookie.
   c. If the cookie is empty, output a message that says that the customer is not authenticated, and has not yet accessed the shopping cart.

   Your code should look like the following:

```
If User.Identity.Name <> "" Then
   Response.write("Customer ID from authentication " & _
   "cookie:<br>")
   Response.write(User.Identity.Name)
Else
   If Request.Cookies("Conference_CartID") Is Nothing Then
      Response.write("Customer not authenticated " & _
      "and has not accessed the shopping cart yet")

   Else
      Response.write("Temporary Customer ID stored " & _
      "in a cookie:<br>")
      Response.write( _
      Request.Cookies("Conference_CartID").Value)

   End If
End If
```

↙ To save and test your work

1. Save your changes to GetID.aspx.
2. Start a new instance of Internet Explorer and go to the GetID.aspx page by viewing http://localhost/ASPNET/GetID.aspx

   You should be able to see the message stating that the customer is not authenticated and has not yet accessed the shopping cart.
3. Click **Go to Home Page**.
4. Select an item and add it to the shopping cart.
5. Go back to the GetID.aspx page and click **Refresh**.

   You should see the message that a temporary customer ID is stored in a cookie, as well as the value of the temporary ID.
6. Click **Go to Login Page**.
7. Enter a valid e-mail name and password, and then click **Sign In Now**. For this lab, use **someone@microsoft.com** for the user name and **password** for the password.
8. Go back to the GetID.aspx page, and click **Refresh**.

   You should see the message that the customer ID is from an authentication cookie and the value of the Customer ID.

↙ To view the GetShoppingCartId method

1. Open the ShoppingCartDB.vb file in the Components folder of the ASPNET Web site.
2. Locate the **GetShoppingCartID** method.

   This method returns an ID to be used for adding items to a shopping cart in the database. The method returns the value of the authenticated user, the value stored in the **Conference_CartID** cookie, or a new temporary ID created by calling **Guid.NewGuid**

Exercise 4
Using Forms Authentication in Register.aspx

In this exercise, you will add a new customer to the database. Next, you will create an authentication cookie for the new customer. Afterward, you will transfer the content of the temporary shopping cart into the new customer's permanent shopping cart and redirect to the shopping cart page (ShoppingCart.aspx).

As you did in the previous exercise, you will call two different components: the first, named **Conference.ShoppingCartDB**, is used to manipulate the shopping cart; the second, **Conference.CustomersDB** is used to access to the customer information.

You will need to call the **Conference.CustomersDB.AddCustomer()** method, which inserts a new customer record into the customers database. This method returns a unique Customer ID.

```
Public Function AddCustomer(fullName As String, email As
String, password As String) As String
```

↙ To connect to a data source

1. Open the file Register.aspx in the folder InetPub\wwwRoot\ASPNET.
2. Delete the following instruction from the **cmdValidation_Click** event procedure:

   ```
   Response.Redirect("validOK.aspx")
   ```

3. In the **cmdValidation_Click** event procedure, declare the following variables:

Variable Name	Data Type
shoppingCart	New Conference.ShoppingCartDB
strTempCartID	String
accountSystem	New Conference.CustomersDB
strCustomerID	String

4. Call the **GetShoppingCartId** method of the **shoppingCart** object to read the current shopping cart ID. Save the value in the **strTempCartID** variable.
5. Create a new customer in the Customers table of the database by calling the **AddCustomer** method of the **accountSystem** object. Save the created customer ID in the **strCustomerID** variable.

   Your code should look like the following:

   ```
   strCustomerId = _
      accountSystem.AddCustomer(txtFullName.Text, _
            txtEmail.Text, txtPassword.Text)
   ```

Note Use the **Text** property of these controls because they are Web controls.

6. If the returned CustomerID is not empty, the user was added to the database and you need to migrate the items in the temporary shopping cart to the permanent shopping cart:

   a. Authenticate a non-persistent cookie by passing the **strCustomerID** parameter to the **SetAuthCookie** method.

   b. Get the shopping cart ID by using the **GetShoppingCartId** method of the **shoppingCart** object and store it in the **strTempCartID** variable.

   c. Migrate any existing shopping cart items to the permanent shopping cart by using the **MigrateCart** method of the **shoppingCart** object. Pass the temporary cart ID and the customer ID to the **MigrateCart** method.

   d. Redirect the user to the shoppingCart.aspx page he or she came from by calling the **Navigate** method of the **Page** object.

   Your code should look like the following:

```
If (strCustomerId <> "") Then
   ' Set the user's authentication name to the customerId
   FormsAuthentication.SetAuthCookie(strCustomerId, False)

   ' Migrate any existing shopping cart items into
   ' the permanent shopping cart
   shoppingCart.MigrateCart(strTempCartID, strCustomerId)

   ' Redirect browser back to shopping cart page
   Response.Redirect("ShoppingCart.aspx")
End If
```

↙ To save and test your work

1. Save your changes to Register.aspx.
2. Using Internet Explorer, go to the register page of the ASPNET Web site by viewing http://localhost/ASPNET/register.aspx
3. Create a new user with your full name, your e-mail address, and a password, and then click **Submit**.
4. To verify whether this new user was successfully registered, open the Login page by viewing http://localhost/ASPNET/login.aspx. Enter the e-mail address and password of the new user you just registered, and then click **Sign In Now!**

   You should receive validation and be redirected to the home page of the ASPNET Web site, default.aspx.

Review

- Requirements of a Web Application
- What Is New in ASP.NET?
- Sharing Information Between Pages
- Securing an ASP.NET Application

---

1. How do you set up a Web application to use cookie-less sessions?

2. Where is global.asax file of an application located?

3. Where is the web.config file for an application located?

4. In web.config, all configuration information must reside under what tags?

5. How do you specify which authentication method will be used by an ASP.NET application?

6. What is the difference between output cache and ASP.NET cache?

7. How can you secure an entire application?