6

Security

VERSION 1.1 OF ASP.NET provided many built-in security services for developers to take advantage of. A common favorite is Forms-based authentication.

Forms-based authentication allows Web developers to easily build applications that require authentication to access secured resources. However, rather than relying on Windows Authentication, Forms-based authentication allows us to author a simple ASP.NET login page. ASP.NET is then configured so that any unauthenticated requests are redirected to the login page (see Figure 6.1).

The login page is a simple ASP.NET page used to collect and verify the user's credentials. It is the responsibility of the login page to determine whether the user credentials are valid; typically this information is stored in a database.

Listing 6.1 shows an example of a login page written in ASP.NET 1.1.

LISTING 6.1. Example Login Page

```
<%@ Page Language="VB" %>
<%@ import namespace="System.Data" %>
<%@ import namespace="System.Data.SqlClient" %>

<script runat="server">

  Public Sub Login_Click(ByVal sender As Object, ByVal e As EventArgs)
```

continues

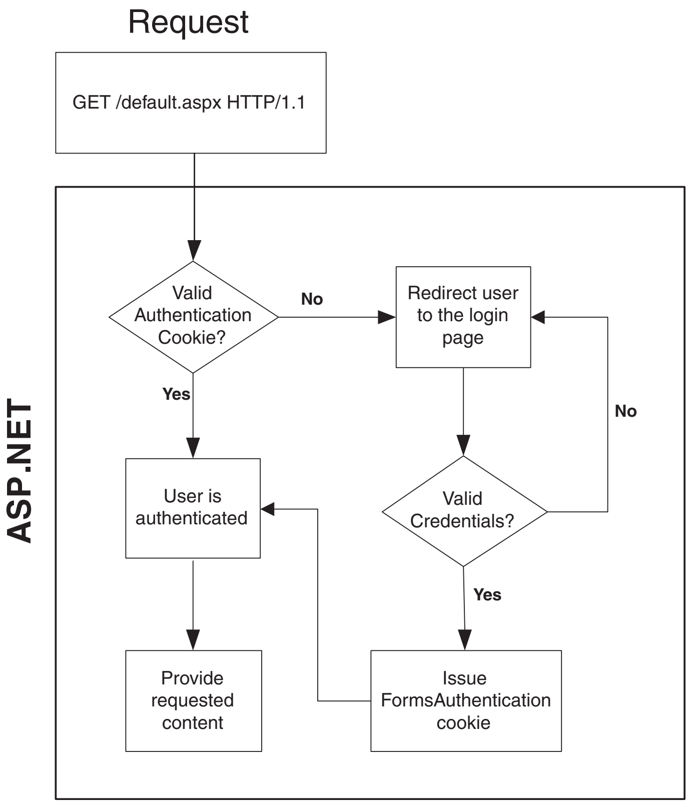

Request
GET /default.aspx HTTP/1.1
ASP.NET
Valid Authentication Cookie?
No
Redirect user to the login page
Yes
No
User is authenticated
Valid Credentials?
Yes
Provide requested content
Issue FormsAuthentication cookie

FIGURE 6.1. Forms Authentication

```
Dim userId As Integer
Dim reader As SqlDataReader
Dim connectionString = _
   ConfigurationSettings.ConnectionStrings("MyConnectionString")
Dim conn As New SqlConnection(connectionString)
Dim command As New SqlCommand("dbo.Authenticate", conn)

' Set the command type to stored procedure
command.CommandType = CommandType.StoredProcedure

' Set @Username and @Password
command.Parameters.Add("@Username", _
        SqlDbType.NVarChar, 256).Value = Username.Text
command.Parameters.Add("@Password", _
        SqlDbType.NVarChar, 256).Value = Password.Text

' Open the connection and execute the reader
conn.Open()
reader = command.ExecuteReader()
```

```
    ' Read the value we're looking for
    reader.Read()

    userId = Integer.Parse(reader("UserId"))

    ' Close connections
    reader.Close()
    conn.Close()

    ' Did we find a user?
    If (userId > 0) Then
        FormsAuthentication.RedirectFromLoginPage(Username.Text, _
                                                  False)
    Else
      Status.Text = "Invalid Credentials: Please try again"

    End If

  End Sub

</script>

<html>
  <body style="FONT-FAMILY: Verdana">

  <H1>Enter your username/password</H1>

  <form id="Form1" runat="server">
    Username: <asp:textbox id="Username" runat="server" />
    <br>
    Password: <asp:textbox id="Password" runat="server" />
    <p>
    <asp:button id="Button1"
                text="Check if Member is Valid"
                onclick="Login_Click" runat="server"/>
  </form>

  <font color="red" size="6">
    <asp:label id="Status" runat="server"/>
  </font>

  </body>
</html>
```

In this sample the login page raises the `Login_Click` event, connects to a database, calls a stored procedure to verify the submitted username and password, and then either uses the `FormsAuthentication` APIs to log the user in or tells the user that the credentials are invalid.

The ASP.NET `FormsAuthentication` class is used to encrypt the username and store it securely in an HTTP cookie. On subsequent requests this HTTP cookie, with its encrypted contents, is decrypted and the user automatically reauthenticated.

Forms Authentication is definitely a great feature, but what makes it even better is the reduction in the amount of code developers must write. Forms Authentication isn't something new introduced by ASP.NET. Rather, ASP.NET is simply providing an easier way to solve the problem; in the past, most developers would have needed to author this code plus infrastructure on their own.

One of the things you may have noticed about the ASP.NET team members: They are always looking for ways to make things easier. They want developers to solve problems without writing hundreds of lines of code. For ASP.NET 2.0 they're again tackling many security-related problems and providing new features to make things simpler.

In this chapter we're going to examine some of the security infrastructure and controls that have been added in ASP.NET 2.0. We'll start by looking at the new Membership feature. Membership solves the user credential storage problem, a problem most developers solved themselves in ASP.NET 1.0.

Membership

After Microsoft released ASP.NET 1.0, the team members immediately started looking for areas where they could simplify. One area was the management of user credentials, personalization, and user roles. These problems could be solved in ASP.NET 1.1, but the team wanted to make the process better and easier!

The Membership feature of ASP.NET does just that—makes it better and easier. Membership provides secure credential storage with simple, easy-to-use APIs. Rather than requiring you to repeatedly develop infrastructure features for authenticating users, it is now part of the platform. More importantly, it's a pluggable part of the platform through the new provider pattern, allowing you to easily extend the system (e.g., to add support for LDAP or existing corporate user account systems).

Forms Authentication and Membership complement one another. However, they can also act independently; that is, you don't have to use them together. The code sample in Listing 6.2 demonstrates how Membership is used with Forms Authentication.

Listing 6.2. Using the Membership API

```
<script runat="server">

  Public Sub Login_Click(sender As Object, e As EventArgs e)

    ' Is the user valid?
    If (Membership.ValidateUser (Username.Text, Password.Text)) Then

      FormsAuthentication.RedirectFromLoginPage (Username.Text, false)

    Else

      Status.Text = "Invalid Credentials: Please try again"

    End If

  End Sub

</script>

<html>
    <body style="FONT-FAMILY: Verdana">

    <H1>Enter your username/password</H1>

    <form id="Form1" runat="server">
      Username: <asp:textbox id="Username" runat="server" />
      <br>
      Password: <asp:textbox id="Password" runat="server" />
      <p>
      <asp:button id="Button1"
                  text="Check if Member is Valid"
                  onclick="Login_Click" runat="server"/>
    </form>

    <font color="red" size="6">
      <asp:label id="Status" runat="server"/>
    </font>

    </body>
</html>
```

As you can see, our custom code to validate the credentials is now replaced with a single call to the static `Membership.ValidateUser()` method. The code is also much cleaner and more readable as a result—and much more concise!

The `Membership` class contains only static methods. You don't have to create an instance of the class to use its functionality; for example, you don't have to `new` the `Membership` class to use it. Behind the scenes the

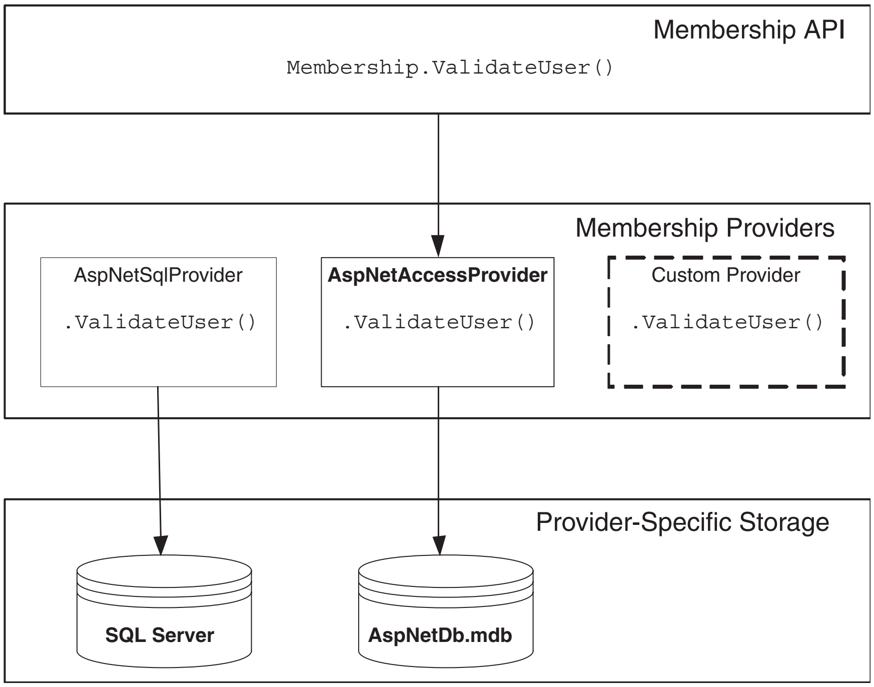


FIGURE 6.2. The provider model

`Membership` class is forwarding the calls through a configured provider. The provider in turn knows which data source to contact and how to verify the credentials (see Figure 6.2).

Providers are a new design pattern introduced with ASP.NET 2.0. Providers are pluggable data abstraction layers used within ASP.NET. All ASP.NET 2.0 features that rely on data storage expose a provider layer. The provider layer allows you to take complete control over how and where data is stored.[1]

Membership Providers

The beauty of the provider model is the abstraction that it affords the developer. Rather than being pigeonholed into a particular data model or fixed API behavior, the provider pattern allows you to determine how and where the actual data storage takes place and the behavior of the API itself.

[1] For more details on providers, see Chapter 7. Also, the provider specification is published on MSDN at http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnaspnet/html/asp02182004.asp.

ASP.NET 2.0 will ship with several providers for Membership (not a complete list):

- Access
- SQL Server
- Active Directory[2]

You can easily author your own provider and plug it in. The provider design pattern allows for one common API that developers can familiarize themselves with, such as Membership, but under the covers you still have control over what exactly is happening. For example, if you had all of your customer information stored in an AS/400, you could write a provider for Membership. Users would call the familiar Membership APIs, but the work would actually be handled by the configured AS/400 provider.

The goal of Membership is to simplify managing and storing user credentials while still allowing you to control your data, but it does much more. Let's dig deeper.

Setting Up Membership

Setting up Membership is easy: It just works. By default all the providers that ship with ASP.NET 2.0 use a Microsoft Access provider and will use the default `AspNetDB.mdb` file created in the `\data\` directory of your application.[3]

If the `\data\` directory of your application does not exist, ASP.NET will attempt to create it. If ASP.NET is unable to create the `\data\` directory or the `AspNetDB.mdb` file because of the security policy on the machine, an exception is thrown detailing what needs to be done.

Before we can begin using Membership for its most common task—validating user credentials—we need to have users to validate!

Creating a New User

The Membership API exposes two methods for creating new users:

[2] Not available with the beta release of ASP.NET 2.0.

[3] Access is the configured default since it works without requiring the user to perform any further setup. SQL Server is the recommended provider for large applications.

```
CreateUser(username As String, password As String)

CreateUser(username As String, password As String,
           email As String)
```

These two APIs are somewhat self-explanatory. We call them to create a user with a username and password, optionally also providing the e-mail address. Both of these methods return a `MembershipUser` instance, which we'll look at later in this chapter.

Which of these two methods you use is determined by the Membership configuration settings. We can examine the settings in `machine.config` for the defaults (see Listing 6.3, where the line in bold indicates whether or not an e-mail address must be unique).[4]

LISTING 6.3. Membership Configuration

```
<configuration>
  <system.web>

    <membership defaultProvider="AspNetAccessProvider"
                userIsOnlineTimeWindow="15">
      <providers>

        <add
          name="AspNetAccessProvider"
          type="System.Web.Security.AccessMembershipProvider,
                System.Web,
                Version=2.0.3600.0,
                Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
          connectionStringName="AccessFileName"
          enablePasswordRetrieval="false"
          enablePasswordReset="true"
          requiresQuestionAndAnswer="false"
          applicationName="/"
          requiresUniqueEmail="false"
          passwordFormat="Hashed"
          description="Stores and retrieves membership data from
                       the local Microsoft Access database file"/>

      </providers>
    </membership>

  </system.web>
</configuration>
```

[4] You can also define these settings or change the defaults in the `web.config` file of your application.

Table 6.1 shows an explanation of the various configuration settings.

TABLE 6.1. Configuration Elements for the Membership Provider

Attribute	Description
`connectionStringName`	Names the key within the `<connectionStrings />` configuration section where the connection string is stored. The default value for the Access provider is `AccessFileName`, and for the SQL Server provider it is `LocalSqlServer`.
`enablePasswordRetrieval`	Controls whether or not the password can be retrieved through the Membership APIs. When set to `false`, the password cannot be retrieved from the database. The default value is `false`.
`enablePasswordReset`	Allows the password to be reset. For example, although the password may not be retrieved, the APIs will allow for a new random password to be created for the user. The default value is `true`.
`requiresQuestionAndAnswer`	Allows the use of a question and answer to retrieve the user's password. Only valid when the `passwordFormat` setting is not `Hashed` and `enablePasswordRetrieval` is `true`. The default value is `false`.
`applicationName`	Indicates the application to which the Membership data store belongs. Multiple applications can share the same Membership data store by specifying the same `applicationName` value. The default value is `/`.
`requiresUniqueEmail`	Requires that a given e-mail address can be used only once. This attribute can be used to prevent users from creating multiple accounts. Note that the uniqueness is constrained to the `applicationName` the user is created within. The default value is `false`.
`passwordFormat`	Controls how the password is stored in the data store. `Hashed` is the most secure but does not allow password retrieval. Additional valid values include `Encrypted` and `Clear`. The default value is `Hashed`.
`description`	Describes the provider. This is an optional attribute; when present, tools capable of working with providers can optionally display this description string.

Knowing what the defaults are, we can write a simple page for creating new users (see Listing 6.4).

Listing 6.4. Creating Users with the Membership API

```
<%@ Page Language="VB" %>

<script runat="server">

  Public Sub CreateUser_Click (sender As Object, e As EventArgs)

    Try

      ' Attempt to create the user
      Membership.CreateUser(Username.Text, Password.Text)

      Status.Text = "Created new user: " & Username.Text

    Catch ex As MembershipCreateUserException

      ' Display the status if an exception occurred
      Status.Text = ex.ToString()

    End Try


  End Sub

</script>
<html>
  <head>
  </head>

  <body style="FONT-FAMILY: Verdana">

    <H1>Create a new user</H1>

    <hr />

    <form runat="server">
    Desired username: <asp:TextBox id="Username" runat="server"/>
    <br>
    Password: <asp:TextBox id="Password" runat="server" />
    <p>
    <asp:button Text="Create Member"
                OnClick="CreateUser_Click" runat="server"/>
    </form>

    <font color="red" size="6">
    <asp:Label id="Status" runat="server" />
    </font>

  </body>
</html>
```

The code in Listing 6.4 calls the `Membership.CreateUser()` method, which accepts a username and a password.[5] If there is a problem creating the user, a `MembershipCreateUserException` is thrown. If there are no problems, the new user is created.

Once we've created some users, we can test the `Membership .ValidateUser()` method.

Validating User Credentials

The primary purpose for Membership is to validate credentials. This is accomplished through the static `ValidateUser()` method:

```
ValidateUser(username As String,
             password As String) As Boolean
```

We can use this method, as seen earlier, along with Forms Authentication to validate user credentials. Here is a partial code example:

```
  If (Membership.ValidateUser (Username.Text, Password.Text)) Then

    FormsAuthentication.RedirectFromLoginPage (Username.Text, False)

  Else

    Status.Text = "Invalid Credentials: Please try again"

  End If
```

Apart from `ValidateUser()`, most of the remaining Membership APIs are used for retrieving a user or users.

Retrieving a User

There are a few ways you can retrieve users that have already been created:

```
GetUser() As MembershipUser

GetUser(userIsOnline As Boolean) As MembershipUser

GetUser(username As String) As MembershipUser

GetUser(username As String,
        userIsOnline As Boolean) As MembershipUser
```

[5] You may already wonder what we do with additional user data, such as first names. Membership is not used to store this type of data. Instead, the new Personalization feature is used to store user data—Membership is used only for storing user credentials used in authentication. Personalization is covered in Chapter 7.

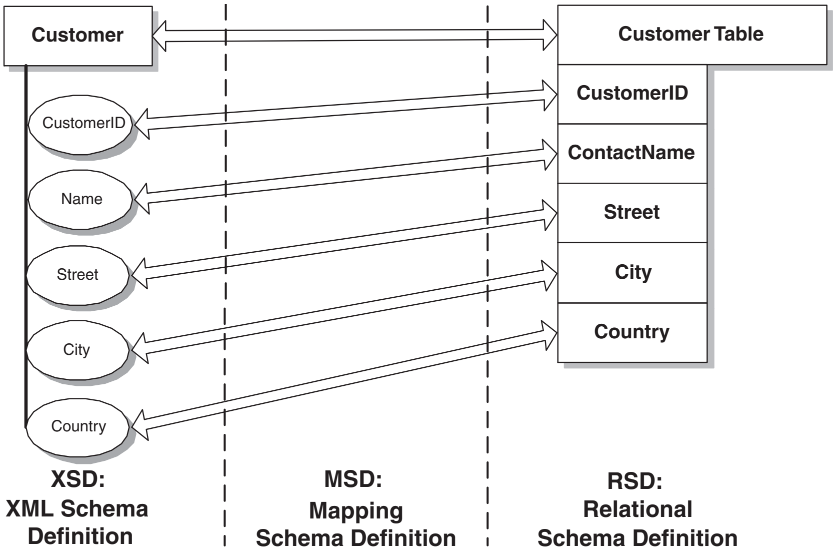


FIGURE 6.3. Getting a user

The first two methods that don't have a *username* parameter will attempt to return the currently logged-on user. The parameter *userIsOnline*, when set to True, will update a timestamp in the data store indicating the date/time the user was last requested. This timestamp can then be used to calculate the total number of users online.[6] The remaining methods will perform similar operations but on a specified user.

Figure 6.3 shows an example of getting the MembershipUser class for the currently logged-on user.

Listing 6.5 provides the code used for this page.

LISTING 6.5. Fetching the Logged-on User

```
<%@ Page Language="VB" %>

<script runat="server">

  Public Sub Page_Load()
```

[6] This functionality is similar to that used on the ASP.NET forums (http://www.asp.net/Forums/). All users have a timestamp that can be updated. The number of users online is calculated by finding all users whose timestamps fall within a calculated window of time. This time window is configured in the <membership> configuration setting userIsOnlineTimeWindow.

```
    Dim user As MembershipUser

    ' Get the currently logged-on user and
    ' update the user's online timestamp
    user = Membership.GetUser(True)

    UserName.Text = user.Username

  End Sub

</script>

<html>

  <body style="FONT-FAMILY: Verdana">

  <H1>Get User</H1>

  <hr />

     <form runat="server">
       The currently logged-on user is:
       <asp:literal id="UserName" runat="server" />
  </form>

  </body>
</html>
```

If we want to find a user but don't have the username (e.g., the user forgot his or her username), we can use the `GetUserNameByEmail()` method:

```
GetUserNameByEmail(email As String) As String
```

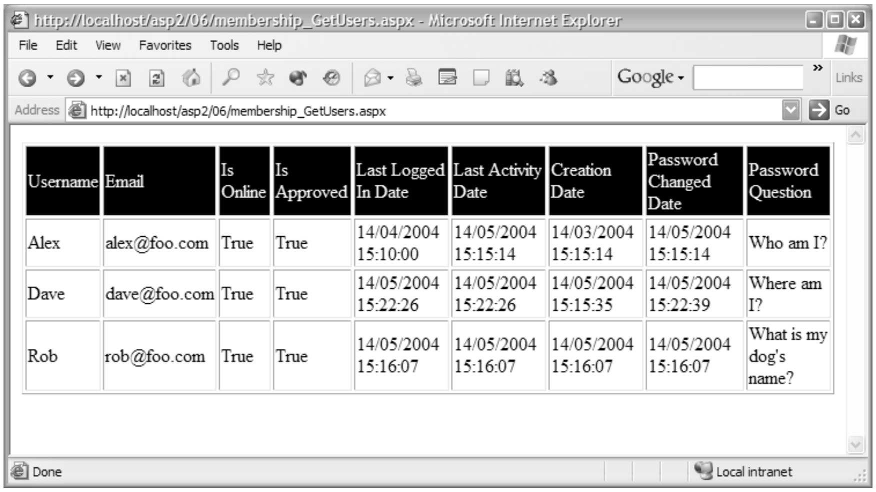
Once we have the username, we can then look up the user with one of the `GetUser()` methods listed earlier.

We can additionally get multiple users with the following method:

```
Membership.GetAllUsers() As MembershipUserCollection
```

`Membership.GetAllUsers()` simply returns a `MembershipUserCollection`, which we can use to enumerate users or bind to a server control, such as a `Repeater` or `DataGrid` (see Figure 6.4).

Listing 6.6 shows the code.


http://localhost/asp2/06/membership_GetUsers.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Google
Links
Address http://localhost/asp2/06/membership_GetUsers.aspx
Go
Done
Local intranet

Username	Email	Is Online	Is Approved	Last Logged In Date	Last Activity Date	Creation Date	Password Changed Date	Password Question
Alex	alex@foo.com	True	True	14/04/2004 15:10:00	14/05/2004 15:15:14	14/03/2004 15:15:14	14/05/2004 15:15:14	Who am I?
Dave	dave@foo.com	True	True	14/05/2004 15:22:26	14/05/2004 15:22:26	14/05/2004 15:15:35	14/05/2004 15:22:39	Where am I?
Rob	rob@foo.com	True	True	14/05/2004 15:16:07	14/05/2004 15:16:07	14/05/2004 15:16:07	14/05/2004 15:16:07	What is my dog's name?

FIGURE 6.4. Getting all users

LISTING 6.6. Displaying All Users

```
<%@ Page Language="VB" %>

<script runat="server">

  Public Sub Page_Load()

    Users.DataSource = Membership.GetAllUsers()
    Users.DataBind()

  End Sub

</script>

<html>
  <head>
  </head>

  <body style="FONT-FAMILY: Verdana">

    <H1>Users in Membership Database</H1>

    <hr />

    <asp:repeater id="Users" runat="server">
      <headertemplate>
        <table border="1">
          <tr>
            <td bgcolor="black" style="color:white">
                Username
            </td>
            <td bgcolor="black" style="color:white">
                Email
            </td>

            <td bgcolor="black" style="color:white">
                Is Online
            </td>

            <td bgcolor="black" style="color:white">
                Is Approved
            </td>

            <td bgcolor="black" style="color:white">
                Last Logged In Date
            </td>

            <td bgcolor="black" style="color:white">
                Last Activity Date
            </td>

            <td bgcolor="black" style="color:white">
                Creation Date
            </td>

            <td bgcolor="black" style="color:white">
                Password Changed Date
            </td>

            <td bgcolor="black" style="color:white">
                Password Question
            </td>
          </tr>
      </headertemplate>

      <itemtemplate>
        <tr>
          <td>
            <%# Eval("Username") %>
          </td>

          <td>
            <%# Eval("Email") %>
          </td>

          <td>
            <%# Eval("IsOnline") %>
          </td>

          <td>
            <%# Eval("IsApproved") %>
          </td>
```

```
          <td>
            <%# Eval("LastLoginDate") %>
          </td>

          <td>
            <%# Eval("LastActivityDate") %>
          </td>

          <td>
            <%# Eval("CreationDate") %>
          </td>

          <td>
            <%# Eval("LastPasswordChangedDate") %>
          </td>

          <td>
            <%# Eval("PasswordQuestion") %>
          </td>
        </tr>
      </itemtemplate>

      <footertemplate>
        </table>
      </footertemplate>
    </asp:repeater>


  </body>
</html>
```

The `GetAllUsers()` method now also supports paging for working with large sets of users. This overloaded version expects `pageIndex`, `pageSize`, and `totalRecords` out parameters, where `pageIndex` is the location within the result set and `pageSize` controls the number of records returned per page. For example, a `pageIndex` of `2` with a `pageSize` of `25` in a system with 2,000 records would return users 26–50.

Now that we've looked at how to create users and retrieve named users, let's look at the `MembershipUser` class, which allows us to set and retrieve extended properties for each user.

The MembershipUser Class

The `MembershipUser` class represents a user stored in the Membership system. It provides the following methods for performing user-specific operations, such as retrieving or resetting a user's password.

```
GetPassword() As String

GetPassword(answer As String) As String

ChangePassword(oldPassword As String,
               newPassword As String) As Boolean

ChangePasswordQuestionAndAnswer(password As String,
                                question As String,
                                answer As String) As Boolean

ResetPassword() As String

ResetPassword(answer As String) As String
```

Note that if a question and answer are being used, the overloaded `GetPassword(`*`answer`*` As String)` method requires the case-insensitive question answer.

The `ChangePassword()` method allows changes to the user's password, and the `ChangePasswordQuestionAndAnswer()` method allows changes to the user's password question and answer. The code in Listing 6.7 allows the currently logged-on user to change his or her password question and answer.[7]

LISTING 6.7. Changing a Password

```
<%@ Page Language="VB" %>

<script runat="server">

  Public Sub Page_Load()

    If Not Page.IsPostBack Then
      DisplayCurrentQuestion()
    End If

  End Sub

  Public Sub SetQandA_Click(sender As Object, e As EventArgs)

    Dim u As MembershipUser = Membership.GetUser()
```

continues

[7] When the `<membership />` configuration's `requiresQuestionAndAnswer` is set to `true`, the `GetPassword(`*`answer`*` As String)` and `ResetPassword(`*`answer`*` As String)` methods must be used to either retrieve or reset the user's password. (The *`answer`* value is the answer to the user's question.)

```
    u.ChangePasswordQuestionAndAnswer(CurrentPassword.Text, _
                                      Question.Text, _
                                      Answer.Text)

    Membership.UpdateUser(u)

    DisplayCurrentQuestion()
  End Sub

  Public Sub DisplayCurrentQuestion()

    Status.Text = Membership.GetUser().PasswordQuestion

  End Sub

</script>

<html>
  <body style="FONT-FAMILY: Verdana">

  <H1>Set Question Answer</H1>

  <hr />

  <form id="Form1" runat="server">
    Current Password: <asp:textbox id="CurrentPassword"
                                   runat="server" />
    <p></p>
    Question: <asp:textbox id="Question" runat="server" />
    <p></p>
    Answer:  <asp:textbox id="Answer" runat="server" />
    <p></p>
    <asp:button id="Button1" text="Set Question/Answer"
                onclick="SetQandA_Click" runat="server"/>
  </form>

  <font size="6"> Your new password question is:
  <asp:label id="Status" runat="server"/>
  </font>

</html>
```

The `ResetPassword()` methods are similar to the `GetPassword()` methods. However, rather than retrieving the user's password, they reset and then return a random password for the user.

Keep in mind that the ability to retrieve, change, or reset the user's password is determined by the settings within the configuration.

In addition to password management, the `MembershipUser` class has some useful properties that provide us some details about how and when the user last logged in, last changed passwords, and so on (see Table 6.2).

TABLE 6.2. MembershipUser Properties

Property	Description
`LastLoginDate`	Sets or returns a timestamp for the last time `ValidateUser()` was called for the current `MembershipUser`.
`CreationDate`	Sets or returns a timestamp value set when the user was first created.
`LastActivityDate`	Sets or returns a timestamp value set when the user authenticates or is retrieved using the overloaded `GetUser()` method that accepts a `userIsOnline` parameter.
`LastPasswordChangedDate`	Sets or returns a timestamp value set when the user last changed his or her password.
`Email`	Sets or returns the e-mail address, if set, of the user.
`IsApproved`	Sets or returns a value that indicates whether or not the user is approved. Users whose `IsApproved` property is set to `false` cannot log in, even when the specified credentials are valid.
`PasswordQuestion`	Returns the question used in question/answer retrieval.
`Provider`	Returns an instance (of type `MembershipProvider`) of the current provider used to manipulate the data store.
`UserName`	Returns the username of the current user.

Updating a User's Properties

When changes are made to the user, for example, updating the user's e-mail address, we need to use the `Membership.UpdateUser(`*`user`*` As MembershipUser)` method to save the values.[8] For example, in Listing 6.7 earlier, the `SetQandA_Click` event (repeated here for convenience) shows an example of `Membership.UpdateUser()`:

[8] The goal of this design is to allow multiple values to be changed without requiring multiple round-trips to the data store. By using the `UpdateUser()` method, all updates are batched together.

```
  Public Sub SetQandA_Click(sender As Object, e As EventArgs)

    Dim u As MembershipUser = Membership.GetUser()

    u.ChangePasswordQuestionAndAnswer(CurrentPassword.Text,
                                      Question.Text,
                                      Answer.Text)

    Membership.UpdateUser(u)

    DisplayCurrentQuestion()
  End Sub
```

So far we've learned how to create and update users, but what about removing users from the Membership system?

Deleting a User

Deleting a user from Membership is easy. Membership supports a single method for removing users:

```
DeleteUser(username As String) As Boolean
```

We simply need to name the user we wish to delete. If the operation is successful, the method returns `True`. If the delete operation fails, for example, if the user doesn't exist, `False` is returned.

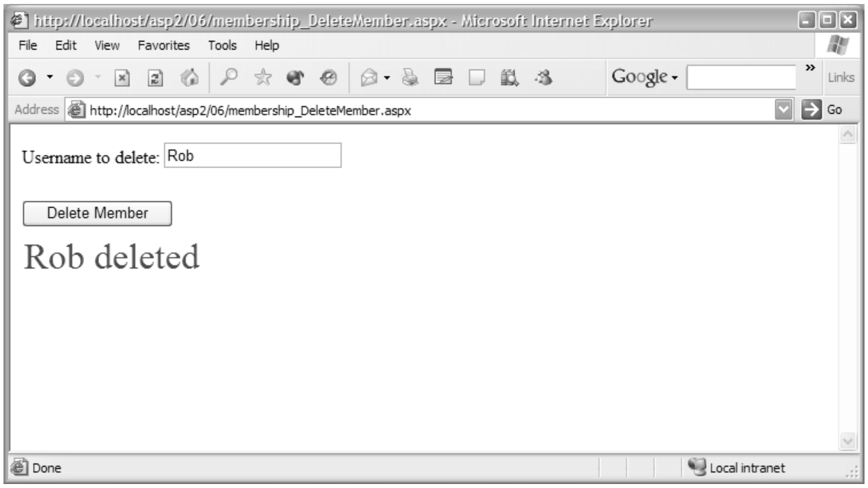
Listing 6.8 shows a code example that allows us to specify a user to be removed from the Membership system.

LISTING 6.8. Deleting a User

```
<%@ Page Language="VB" %>

<script runat="server">

  Public Sub DeleteUser_Click(sender As Object, e As EventArgs)

    If (Membership.DeleteUser(Username.Text)) Then
      Status.Text = Username.Text & " deleted"
    Else
      Status.Text = Username.Text & " not deleted"
    End If

  End Sub

</script>
<html>
  <head>
  </head>
  <body style="FONT-FAMILY: Verdana">


    <H1>Delete a user</H1>

    <hr />

    <form runat="server">
      Username to delete: <asp:TextBox id="Username"
                                 runat="server"/>
      <p>
      <asp:button Text="Delete User"
                  OnClick="DeleteUser_Click" runat="server"/>
    </form>

    <font color="red" size="6">
    <asp:label id="Status" runat="server" />
    </font>

  </body>
</html>
```

Figure 6.5 shows how this page looks.

While the Membership APIs definitely simplify day-to-day tasks, there is also an alternative to using programmatic APIs: security server controls. In many cases we can use these server controls and never have to write code that uses the Membership APIs!



http://localhost/asp2/06/membership_DeleteMember.aspx - Microsoft Internet Explorer

Address http://localhost/asp2/06/membership_DeleteMember.aspx

Username to delete: Rob

Rob deleted

Done

FIGURE 6.5. Deleting a user

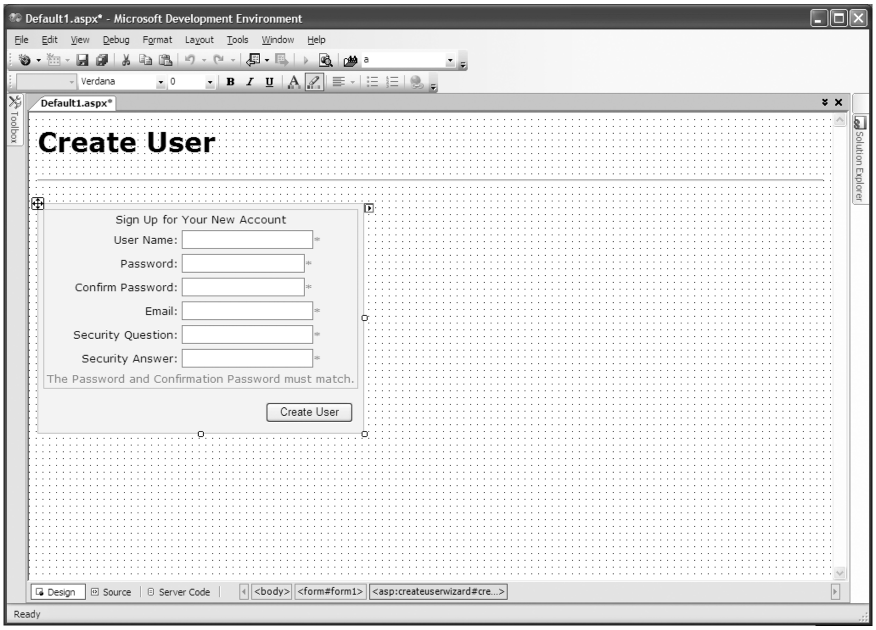
Security Server Controls

The new Membership infrastructure feature of ASP.NET simplifies the management and storage of user credentials. Using APIs, such as `Membership.ValidateUser()`, for Forms Authentication definitely makes things easy. However, some techniques are made even easier through the use of several new security-related server controls. For example, you can author your own login page with zero lines of code by using the new `Login` control or create users using the new `CreateUserWizard` control.

The CreateUserWizard Control

Earlier in the chapter we showed the code required to create a new user. In the alpha version of ASP.NET (then code-named "Whidbey") this was the only way to create new users. In the beta version the new `CreateUserWizard` control can be used instead for a no-code alternative.

The `CreateUserWizard` control, `<asp:CreateUserWizard runat= "server" />`, uses the new `Wizard` control and allows for multiple steps to be defined during the user creation process. Figure 6.6 shows the `CreateUserWizard` control in design time. As you can see, the options



The Password and Confirmation Password must match.

FIGURE 6.6. The CreateUserWizard control

available are specific to Membership, but we could just as easily add another step to the control for collecting more user information, such as first name, last name, and so on. Listing 6.9 shows the source to the `control_createuser.aspx` page.

LISTING 6.9. Using the CreateUserWizard Control

```
<html>

  <body style="FONT-FAMILY: Verdana">

    <H1>Validate Credentials</H1>
    <hr />
    <form id="Form1" runat="server">

      <asp:CreateUserWizard runat="server" />

    </form>

  </body>

</html>
```

The Login Control

Figure 6.7 shows a `control_login.aspx` page that authenticates the user's credentials against the default Membership provider and uses the new `<asp:Login runat="server" />` control. As you can see in Figure 6.7, it looks nearly identical to the login page we built with the Membership APIs. Listing 6.10 shows the source to the `control_login.aspx` page.

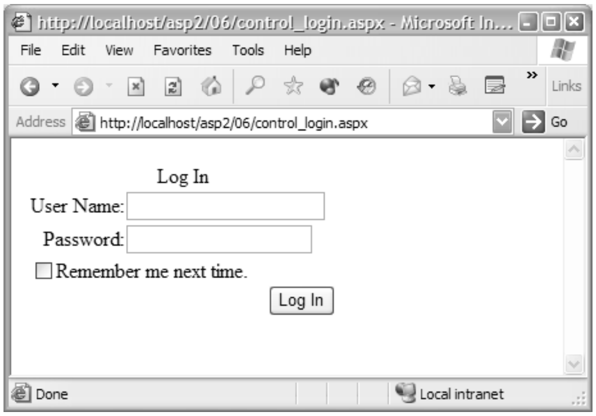


FIGURE 6.7. The login control

LISTING 6.10. Using the Login Control

```
<html>

  <body style="FONT-FAMILY: Verdana">

    <H1>Validate Credentials</H1>
    <hr />
    <form id="Form1" runat="server">

      <asp:Login id="Login1" runat="server" />

    </form>

  </body>

</html>
```

When the username and password are entered, this control will automatically attempt to log in the user by calling `Membership.ValidateUser()`. If successful, the control will then call the necessary `FormsAuthentication.RedirectFromLoginPage` API to issue a cookie and redirect the user to the page he or she was attempting to access. In other words, all the code you would have needed to write in ASP.NET 1.1 is now neatly encapsulated in a single server control!

The `<asp:Login runat="server" />` control automatically hides itself if the user is logged in. This behavior is determined by the `AutoHide` property, set to `True` by default. If the login control is used with Forms Authentication and hosted on the default login page (specified in the configuration for Forms Authentication) the control will not auto-hide itself.

We can further customize the login control's UI. To preview one UI option—we can't cover all of them in depth in this book—right-click on the login control within Visual Studio 2005 and select Auto Format. This will bring up the dialog box shown in Figure 6.8.

Once you've chosen an auto-format template, such as Classic, you can see the changes in the login control (see Listing 6.11).

LISTING 6.11. A Formatted Login Control

```
<asp:Login id="Login1"
           runat="server"
           font-names="Verdana"
           font-size="10pt"
           bordercolor="#999999"
           borderwidth="1px"
           borderstyle="Solid"
```

```
          backcolor="#FFFFCC">
  <TitleTextStyle Font-Bold="True"
                  ForeColor="#FFFFFF"
                  BackColor="#333399">
  </TitleTextStyle>
</asp:Login>
```

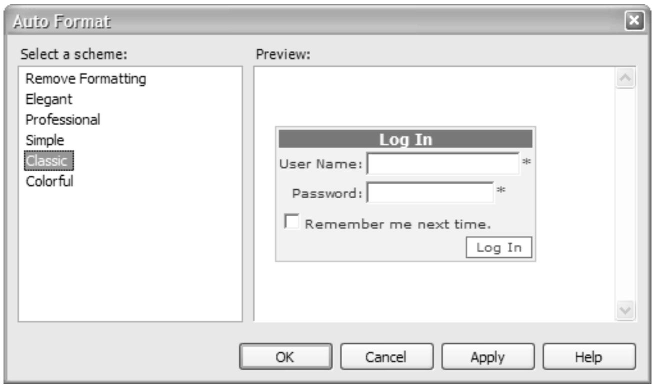


FIGURE 6.8. The Auto Format dialog

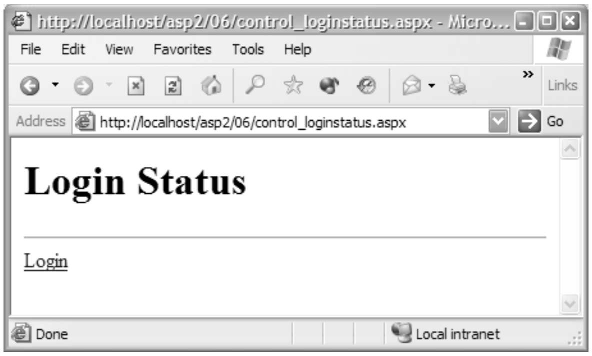
If you desire more control over the display of the control, right-click on the control, or from the Common Tasks dialog, select Convert to Template. You'll see no changes in the rendered UI of the control. However, you will see a notable difference in the declarative markup generated for the control. For brevity we are not including the updated markup.[9] What you will see is a series of templates that allow you to take 100% control over the UI rendering of the control. Note that it is important that the IDs of the controls within these templates remain because the control expects to find these IDs.

While the login control simplifies authoring the login page, several other controls help us display content to users based on their login status. Let's take a look at the login status control first.

The Login Status Control

The login status control, `<asp:LoginStatus runat="server" />`, is used to display whether the user is logged in or not. When the user is not logged in, the status displays a Login link (see Figure 6.9). When the user is logged in, the status displays a Logout link (see Figure 6.10).

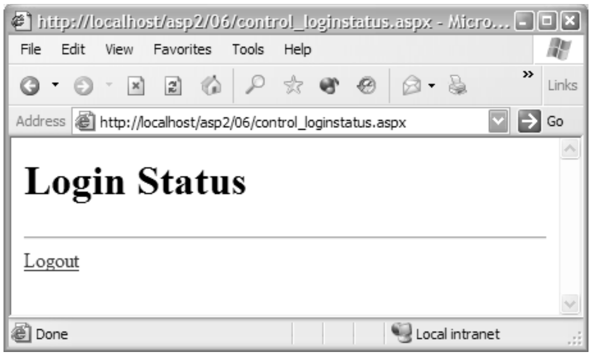
[9] One of the nice features in the Designer is a Reset feature. When you change your mind about using the template layout of the login control, you can simply select Reset from the Common Tasks menu.


http://localhost/asp2/06/control_loginstatus.aspx - Micro...
File Edit View Favorites Tools Help
Links
Address http://localhost/asp2/06/control_loginstatus.aspx
Go
Done
Local intranet

Login Status

Login

FIGURE 6.9. The login status control when the user is not logged in


http://localhost/asp2/06/control_loginstatus.aspx - Micro...
File Edit View Favorites Tools Help
Links
Address http://localhost/asp2/06/control_loginstatus.aspx
Go
Done
Local intranet

Login Status

Logout

FIGURE 6.10. The login status control when the user is logged in

Listing 6.12 shows the code required.

LISTING 6.12. The Login Status Control

```
<html>

  <body style="FONT-FAMILY: Verdana">

  <h1>Login Status</h1>

  <hr />

  <form runat="server">
    <asp:LoginStatus id="LoginStatus1" runat="server" />
  </form>

  </body>

</html>
```

By default the text displayed for the link is "Login" when the user is not logged in and "Logout" when the user is logged in.[10] However, this text can easily be changed; you simply need to change the `LoginText` or `LogoutText` properties of the control:

```
<asp:LoginStatus id="Loginstatus1" runat="server"
                 LoginText="Please log in"
                 LogoutText="Please log out" />
```

Other properties can also be set to control the behavior of this control. For example, you can use the `LoginImageUrl` and the `LogoutImageUrl` to use images rather than text for displaying the login status. Finally, there are two properties for controlling the behavior upon logout:

- `LogoutAction`: This property specifies the behavior when the logout button is clicked. Options include `Refresh`, `Redirect`, and `RedirectToLoginPage`.
- `LogoutPageUrl`: When the `LogoutAction` is set to `Redirect`, the `LogoutPageUrl` is the location to which the browser is redirected.

Whereas `<asp:LoginStatus runat="server" />` provides an easy way for the user to log in and log out, another server control, `<asp:LoginView runat="server" />`, allows us to easily determine what content is shown to the user based on his or her login status.

The Login View Control

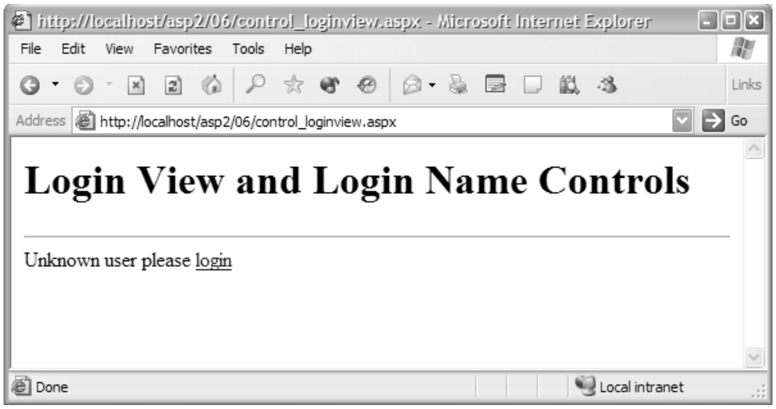
The `<asp:LoginView runat="server" />` server control is used to display different output depending on the login status of the user. Furthermore, the control can also be used to display different content based on the role(s) the user belongs to. Figure 6.11 shows an example of what the control might display for an anonymous user. The code that generates this page appears in Listing 6.13.

LISTING 6.13. Using the Login View Control

```
<html>

  <body style="FONT-FAMILY: Verdana">
```

continues

[10] Note the handy feature in Visual Studio 2005 in the Tasks dialog, Views, which allows you to select the view of the control in the Designer.


http://localhost/asp2/06/control_loginview.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Links
Address http://localhost/asp2/06/control_loginview.aspx
Go
Done
Local intranet

Login View and Login Name Controls

Unknown user please login

FIGURE 6.11. The login view for an anonymous user

```
  <h1>Login View and Login Name Controls</h1>

  <hr />

  <form runat="server">

    <asp:LoginView id="Loginview1" runat="server">
      <anonymoustemplate>
        Unknown user please <asp:LoginStatus runat="server"
                                        logintext="login" />
      </anonymoustemplate>

      <rolegroups>
        <asp:rolegroup roles="Admin">
          <contenttemplate>
            This is admin only content!
          </contenttemplate>
        </asp:rolegroup>
      </rolegroups>

      <loggedintemplate>
        You are logged in as: <asp:LoginName id="LoginName1"
                                         runat="server" />
      </loggedintemplate>
    </asp:LoginView>
  </form>

  </body>

</html>
```

In this code you can see that two templates are defined for the `<asp:LoginView runat="server" />` control:

- `<anonymoustemplate />` is used to control the displayed content when the user is not logged in.
- `<loggedintemplate />` is used to control the displayed content when the user is logged in.

In addition to the templates, there is also a special `<rolegroups />` section that allows us to create different templates that are displayed if the user is in a corresponding role or roles. If the user is logged in but no roles apply, the `<loggedintemplate />` is used.[11]

You'll notice that we also made use of another control in Listing 6.13: `<asp:LoginName runat="server" />`. This control simply displays the name of the logged-in user. If the user is not logged in, the control does not render any output.

The last security-related server control is `<asp:PasswordRecovery runat="server" />`, which is used to help users obtain their forgotten passwords.

The Password Recovery Control

The `<asp:PasswordRecovery runat="server" />` control works in conjunction with the Membership system to allow users to easily recover their passwords.[12]

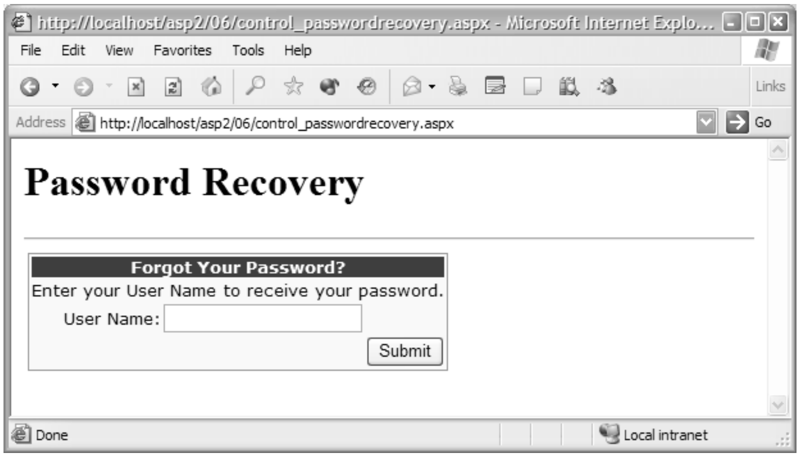
The `<asp:PasswordRecovery runat="server" />` control relies on the `<smtpMail />` configuration options to be correctly set to a valid SMTP server—the control will mail the password to the user's e-mail address. By default, the `<smtpMail />` section will have the SMTP mail server set to `localhost` and the port set to `25` (the default SMTP port).

Similar to `<asp:Login runat="server" />`, this control supports auto-format and full template editing. Assuming we select an auto-format template, such as Classic, and use the default Membership settings, we should see the page shown in Figure 6.12.

Listing 6.14 presents the code that generates this page (auto-formatting removed).

[11] Template evaluation for `rolegroups` is done top to bottom. The first matched role group is used. If no match is found, the `<loggedintemplate />` is used.

[12] For this server control to work, your Membership configuration must be set up to allow for password recovery.


http://localhost/asp2/06/control_passwordrecovery.aspx - Microsoft Internet Explo...
File Edit View Favorites Tools Help
Links
Address http://localhost/asp2/06/control_passwordrecovery.aspx
Go
Done
Local intranet

Password Recovery

Forgot Your Password?

Enter your User Name to receive your password.

User Name:

Submit

FIGURE 6.12. Password recovery

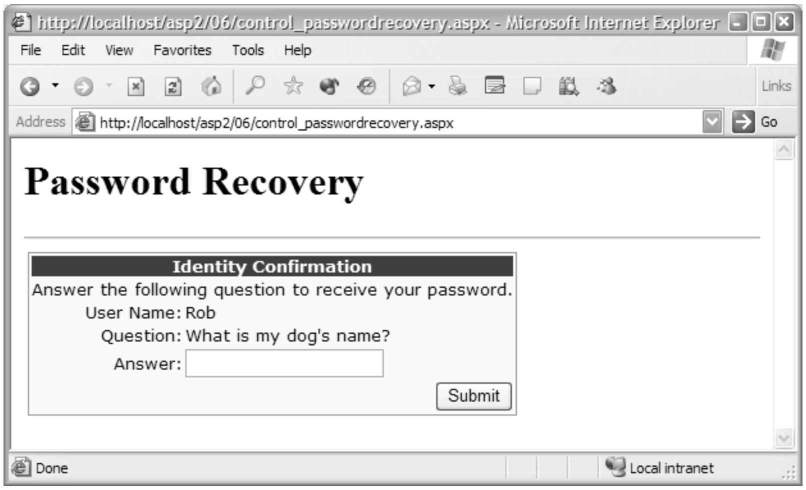
LISTING 6.14. Using the Password Recovery Control

```
<html>
  <body style="FONT-FAMILY: Verdana">

    <H1>
      Password Recovery
    <hr />
    </H1>
    <form id="Form1" runat="server">
      <asp:PasswordRecovery runat="server">
        <maildefinition from="admin@mywebsite.com" />
      </asp:PasswordRecovery>
    </form>

  </body>

</html>
```

If we attempt to use the control with the default Membership settings—which do not allow password recovery—we will receive the following error:

“Your attempt to retrieve your password was not successful. Please try again.”

To allow us to recover the password, we need to change some of the default membership settings. Below are the necessary changes to the `<membership />` configuration settings to allow for password recovery:


http://localhost/asp2/06/control_passwordrecovery.aspx - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Links
Address http://localhost/asp2/06/control_passwordrecovery.aspx
Go
Submit
Done
Local intranet

Password Recovery

Identity Confirmation

Answer the following question to receive your password.

User Name: Rob

Question: What is my dog's name?

Answer:

FIGURE 6.13. Password recovery with question and answer

- `enablePasswordRetrieval="True"`
- `passwordFormat="Clear"`

The `enablePasswordRetrieval` attribute must be set to `True` to allow for password retrieval, and the `passwordFormat` attribute must be set to either `Clear` or `Encrypted`.[13] Another alternative is that when `enablePasswordReset` is set to `True`, the new password can be e-mailed to the user.

In addition to configuring the `<membership />` configuration settings, we must specify the `<maildefinition />` element of the `<asp:PasswordRecovery runat="server" />` control. The `<maildefinition />` names the address from whom e-mails are sent.

Finally, we can also use the `<asp:PasswordRecovery runat="server" />` control to retrieve the user's password using the question/answer support of Membership (see Figure 6.13). The control still requires that we enter the username first, but before simply mailing the password it will first also request the answer to the user's question.

[13] The default `passwordFormat` value, `Hashed`, can be best thought of as one-way encryption; thus it is impossible to retrieve the original value once the value is hashed. The hashed value is also salted (combined with random characters) to further obscure the password.

This behavior is forced by setting the `<membership />` configuration setting `requiresQuestionAndAnswer` to `true` (the default is `false`).[14] Note that this configuration change is in addition to changing the `enablePasswordRetrieval` to `true` and setting the `passwordFormat` to a value other than `Hashed`.

Managing and storing user credentials are only one part of securely controlling access to resources within your site. In addition to validating who the user is, you need to determine whether the user is allowed to access the requested resource. The process of validating credentials is known as *authentication*; *authorization* is the process of determining whether the authenticated user is allowed to access a particular resource.

ASP.NET 1.x already provides authorization facilities, but just as we have shown with Membership, there is more simplification to be done.

Role Manager

The ASP.NET Role Manager feature is designed to simplify managing roles and the users that belong to those roles. After authentication, when Role Manager is enabled, ASP.NET will automatically add the users to the role(s) he or she belongs to. When ASP.NET authorization occurs, the user is either allowed or denied access to the requested resource based on his or her role(s).[15]

URL-based role authorization is a feature of ASP.NET 1.0. We can control what users are allowed to access by specifying access permissions within `configuration` (see Listing 6.15).

Listing 6.15. Configuring Roles

```
<configuration>

  <system.web>
    <authorization>
      <deny users="?" />
    </authorization>
  </system.web>
```

[14] You will also need to ensure that the user set a question and answer when his or her account in the Membership system was created.

[15] Authorization always occurs after authentication; that is, first the user's credentials are validated, and if the credentials are valid, then the determination is made whether the user is allowed to access the requested resource.

```
  <location path="PremiumContent.aspx">
    <system.web>
      <authorization>
        <allow roles="Premium" />
        <deny users="*" />
      </authorization>
    </system.web>
  </location>

</configuration>
```

The above `web.config` file could be added to any application. It states that anonymous users are denied access to all resources. Furthermore, only users in the role `Premium` are allowed access to `PremiumContent.aspx`. All other users are denied access.

Before we can control access to resources through roles, we need to create some roles and then add users to them. Let's look at how we can do this with the new Role Manager feature.

Setting Up Role Manager

Similar to Membership, Role Manager relies on a provider to store data and thus allow us to create roles and associations between users and their roles.[16] Unlike Membership, Role Manager is not enabled by default. Therefore, before we can use the Role Manager API, we need to enable the feature in its configuration settings.

Similar to Membership configuration settings, Role Manager configuration settings are defined in `machine.config` and can be overridden or changed within an application's `web.config` file. Listing 6.16 shows a sample `web.config` enabled for Role Manager.

LISTING 6.16. Configuring Role Manager

```
<configuration>
  <system.web>

    <roleManager enabled="true"
                 cacheRolesInCookie="true"
                 cookieName=".ASPXROLES"
                 cookieTimeout="30"
```

continues

[16] This is identical to how Membership uses a provider. The difference is that a different set of providers exists for the Role Manager feature. Different providers are used to avoid making Role Manager providers dependent on Membership.

```
                cookiePath="/"
                cookieRequireSSL="false"
                cookieSlidingExpiration="true"
                cookieProtection="All"
                defaultProvider="AspNetAccessProvider" >
      <providers>
        <add name="AspNetAccessProvider2"
             type="System.Web.Security.AccessRoleProvider, System.Web"
             connectionStringName="AccessFileName"
             applicationName="/"
             description="Stores and retrieves roles data from
                          the local Microsoft Access database file" />
      </providers>
    </roleManager>

  </system.web>
</configuration>
```

Table 6.3 shows an explanation of the various configuration settings.

Table 6.3. Role Manager Configuration Settings

Attribute	Default Value	Description
enabled	false	Controls whether or not the Role Manager feature is enabled. By default it is disabled because enabling breaks backward compatibility with ASP.NET 1.0.
cacheRolesInCookie	true	Allows for the roles to be cached within an HTTP cookie. When the roles are cached within a cookie, a lookup for the roles associated with the user does not have to be done through the provider.
cookieName	.ASPXROLES	Sets the name of the cookie used to store the roles when cookies are enabled.
cookieTimeout	30	Sets the period of time for which the cookie is valid. If cookieSlidingExpiration is true, the cookie timeout is reset on each request within the cookieTimeout window.
cookiePath	/	Sets the path within the application within which the cookie is valid.
cookieRequireSSL	false	Specifies whether or not the cookie must be sent over an SSL channel.

TABLE 6.3. Role Manager Configuration Settings (continued)

Attribute	Default Value	Description
`cookieSlidingExpiration`	`true`	Sets the cookie timeout. When `true`, the cookie timeout is automatically reset each time a request is made within the `cookieTimeout` window, effectively allowing the cookie to stay valid until the user's session is complete.
`cookieProtection`	`All`	Controls how the data stored within the cookie is secured.
`defaultProvider`	*`string`*	Sets the friendly name of the provider to use for `roleManager`. By default this is `AspNetAccessProvider`.

Now that we've seen how to configure the settings of Role Manager, let's create some roles.

Creating Roles

The `Roles` API supports a single method for creating roles:

```
CreateRole(rolename As String)
```

This API is used to create the friendly role name, such as Administrators, used to control access to resources. Listing 6.17 provides sample code for creating roles in an ASP.NET page.

LISTING 6.17. Creating and Viewing Roles

```
<script runat="server">

  Public Sub Page_Load (sender As Object, e As EventArgs)

    If Not Page.IsPostBack Then
      DataBind()
    End If

  End Sub


  Public Sub CreateRole_Click(sender As Object, e As EventArgs)

    Try

      ' Attempt to create the role
      Roles.CreateRole (Rolename.Text)
```

continues

```
    Catch ex As Exception

      ' Failed to create the role
      Status.Text = ex.ToString()

    End Try

    DataBind()

  End Sub

  Public Overrides Sub DataBind()

    RoleList.DataSource = Roles.GetAllRoles()
    RoleList.DataBind()

  End Sub

</script>


<html>
  <body style="FONT-FAMILY: Verdana">

  <H1>Create Role</H1>
  Below is a list of the current roles:
  <asp:datagrid id="RoleList" runat="server" />

  <hr />

  <form runat="server">

    Rolename to create: <asp:TextBox id="Rolename" runat="server" />
    <asp:button Text="Create Role"
                OnClick="CreateRole_Click" runat="server"/>

  </form>

  <font color="red" size="6">
  <asp:Label id="Status" runat="server"/>
  </font>

  </body>
</html>
```

This code sample allows us to enter a role name, which is then created using the `Roles.CreateRole()` API. If the role already exists, an exception is thrown. Finally, all of the available roles are enumerated through a `DataGrid` using the `Roles.GetAllRoles()` API, discussed shortly.

Now that we can create roles, let's add some users to the roles.

Adding Users to Roles

Membership and Role Manager are not rigidly coupled. They are designed to work together, but you do not have to use one to use the other. Both use the authenticated username as the only shared piece of data. For example, it is possible to add a user to a role even if the user is not created through the Membership system.[17]

Adding users to roles is accomplished by using the following methods supported by the `Roles` API:

```
AddUserToRole(username As String, rolename As String)

AddUserToRoles(username As String, rolenames() As String)

AddUsersToRole(usernames() As String, rolename As String)

AddUsersToRoles(usernames() As String, rolenames() As String)
```

These various methods allow for adding users to roles in bulk or individually. Listing 6.18 demonstrates `Roles.AddUserToRole()`.

LISTING 6.18. Adding Users to Roles

```
<%@ Page Language="VB" %>

<script runat="server">

  Public Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)

    DataBind()

  End Sub

  Public Sub AddUserToRole_Click(sender As Object, e As EventArgs)

    Roles.AddUserToRole(Username.Text, _
                        RoleList.SelectedItem.Value)

    DataBind()

  End Sub
```

continues

[17] This flexible design allows using various providers for data storage. For example, you could have an AS/400 Membership provider and an Access Role Manager provider.

```
  Public Overrides Sub DataBind()

    RoleList.DataSource = Roles.GetAllRoles()
    RoleList.DataBind()

  End Sub

</script>

<html>
  <body style="FONT-FAMILY: Verdana">

  <H1>Add User to Role</H1>

  <form runat="server">
    User: <asp:TextBox id="Username" runat="server" />
    Role to add user to: <asp:DropDownList id="RoleList"
                                         runat="server" />
    <asp:button Text="Add User To Role"
                OnClick="AddUserToRole_Click" runat="server"/>
  </form>

  <font size="6" color="Red">
  <asp:Label id="StatusCheck" runat="server"/>
  </font>

  </body>

</html>
```

This code sample data binds the results of `Roles.GetAllRoles()` to a `DropDownList` control and then allows us to enter a user to add to a role.[18] The `Roles.AddUserToRole()` API is then used to add the user to the role.

When adding multiple users to roles or a user to multiple roles, the addition occurs within the context of a transaction. Either all updates succeed or all fail.

We can now use another `Roles` API to determine to what roles a particular user belongs.

Returning a User's Roles

To return a list of the roles to which a user belongs, we can simply use one of the following APIs:

[18] You could also data-bind to a list of the Members from the Membership system.

```
GetRolesForUser() As String()

GetRolesForUser(username As String) As String()

GetUsersInRole(rolename As String) As String()
```

The `Roles.GetRolesForUser()` method will return a string array of all the roles that the current user is in. The overloaded version of this method that accepts a *`username`* parameter allows us to specify for which user we want a listing of roles. The last method, `Roles.GetUsersInRole()`, allows us to get a string array listing of usernames that belong to the specified role.

Listing 6.19 demonstrates the overloaded version of `Roles.GetRolesForUser()`.

LISTING 6.19. Finding the Roles for a User

```
<script runat="server">

  Public Sub GetRolesForUser_Click(sender As Object, e As EventArgs)

    RolesForUser.DataSource = Roles.GetRolesForUser(Username.Text)
    RolesForUser.DataBind()

  End Sub

</script>

<html>
  <body style="FONT-FAMILY: Verdana">

  <H1>Roles user is in</H1>
  <hr />

  <form runat="server">
    Username: <asp:TextBox id="Username" runat="server" />
    <asp:button Text="Roles User Is In"
                OnClick="GetRolesForUser_Click" runat="server"/>
  </form>

  User is in roles:

  <asp:DataGrid runat="server" id="RolesForUser" />

  </body>
</html>
```

This code sample simply asks for the name of a user. When the page is posted back, the `Roles.GetRolesForUser()` API is called, passing in the name of the specified user. The results are then data-bound to a `DataGrid`.

Checking Whether a User Is in a Role

Access to resources can be controlled by which roles the user belongs to. As shown in the beginning of this section, it is possible to control access to URLs based on settings made in the configuration file. In addition to this declarative security access control, we can also perform programmatic checks for the role the user belongs to.

ASP.NET 1.1 allowed for programmatic checks for determining whether the user was in a role through `User.IsInRole(username as String)`; the result of this method returned `True` or `False`. The `Roles` API supports a similar `Roles.IsUserInRole(rolename As String)` API:

```
IsUserInRole(rolename As String) As Boolean

IsUserInRole(username As String, rolename As String) As Boolean
```

Now that we've seen how to add users to roles and check whether users are in a particular role, let's look at how we can remove a user from a role.

Removing Users from Roles

Similar to the methods used for adding a user to roles, we have four different methods for removing users from roles:

```
RemoveUserFromRole(username As String, rolename As String)

RemoveUserFromRoles(username As String, rolenames() As String)

RemoveUsersFromRole(usernames() As String, rolename As String)

RemoveUsersFromRoles(usernames() As String, rolenames() As String)
```

Again, similar to adding users to roles, when the process of removing users from roles is transacted, either all succeed or all fail.

Deleting a Role

Roles can be deleted easily by using the `Roles.DeleteRole(rolename As String)` method. Listing 6.20 shows a sample ASP.NET page that demonstrates how to use this API.

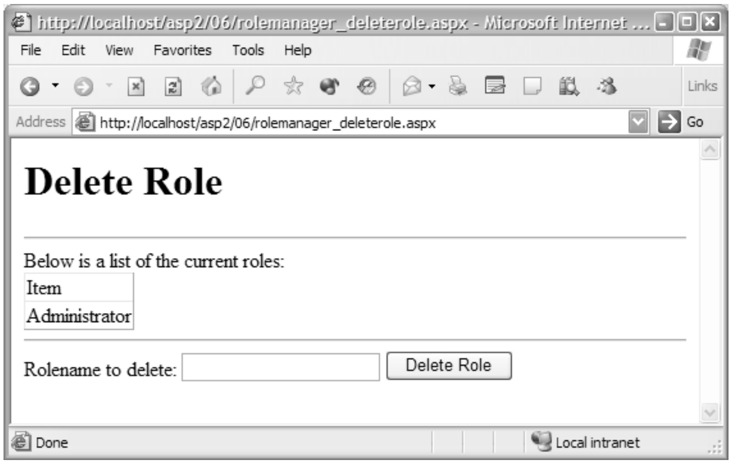
LISTING 6.20. Deleting a Role

```
<%@ Page Language="VB" %>

<script runat="server">

  Public Sub Page_Load()

    If Not Page.IsPostBack Then
      DataBind()
    End If

  End Sub

  Public Sub DeleteRole_Click(sender As Object, e As EventArgs)

    Try
      Roles.DeleteRole(Rolename.Text)
    Catch ex As Exception
      StatusCheck.Text = "There was an error removing the role(s)"
    End Try

    DataBind()

  End Sub

  Public Overrides Sub DataBind()
    RoleList.DataSource = Roles.GetAllRoles()
    RoleList.DataBind()

  End Sub

</script>

<html>
  <body style="FONT-FAMILY: Verdana">

  <H1>Delete Role</H1>
  Below is a list of the current roles:
  <asp:datagrid id="RoleList" runat="server" />

  <hr />

  <form runat="server">
    Rolename to delete: <asp:TextBox id="Rolename" runat="server" />
    <asp:button Text="Delete Role" OnClick="DeleteRole_Click"
                runat="server"/>
  </form>

  <font color="red" size="6">
```

continues

```
  <asp:Label id="StatusCheck" runat="server"/>
  </font>

  </body>

</html>
```

This code lists all the available roles by binding the result of `Roles.GetAllRoles()` to a `DataGrid`. It then allows for a specific role to be named and deleted using the `Roles.DeleteRole()` method, as shown in Figure 6.14. There is a second form of `DeleteRoles` that takes two parameters: the first is the role name and the second a Boolean to indicate whether an exception should be thrown if the role being deleted has users.


http://localhost/asp2/06/rolemanager_deleterole.aspx - Microsoft Internet ...
File Edit View Favorites Tools Help
Links
Address http://localhost/asp2/06/rolemanager_deleterole.aspx
Go
Done
Local intranet

Delete Role

Below is a list of the current roles:

Item
Administrator

Rolename to delete: Delete Role

FIGURE 6.14. Deleting roles

Role Manager uses a provider to write back to and read from a data store in which the roles and user-to-role mapping is done. Rather than reading/writing to this database on each request—since a list of the roles the user belongs to must be obtained—a cookie can optionally be used to cache roles, as described in the next subsection.

Role Caching

Role caching is a feature of Role Manager that enables user-to-role mappings to be performed without requiring a lookup to the data store on each request.[19] Instead of looking up the user-to-role mapping in the data store, the roles the user belongs to are stored, encrypted, within an HTTP cookie. If the user does not have the cookie, a request is made against the provider to retrieve the roles the user belongs to. The roles are then encrypted and stored within a cookie. On subsequent requests the cookie is decrypted and the roles obtained from the cookie.

Internally, in cases where there are more roles than can fit in the cookie, the cookie is marked as an incremental role cookie. That is, the cookie stores as many roles as possible but likely not all the roles. When role checking is performed, and the user is not in one of the roles being checked for, ASP.NET will call the `Roles` API and check whether the user belongs to that role. If not, access is denied. If the user is in the role and the role is not currently stored in the cookie, the last role stored within the cookie is removed and the requested role is added. Thereby, in cases where the user has more roles than can fit in the cookie, the cookie over time will contain a list of the most frequently accessed roles.

Cookieless Forms Authentication

ASP.NET 1.0 introduced the Forms Authentication feature to allow developers to easily author ASP.NET applications that rely on an authentication mechanism they could control. Forms Authentication exposed a set of APIs that developers can simply call to authenticate the user, such as

```
FormsAuthentication.RedirectFromLoginPage(Username.Text, False)
```

Forms Authentication in ASP.NET 1.0 would then take the username, encrypt it, and store it within an HTTP cookie. The cookie would be presented on subsequent requests and the user automatically reauthenticated.

One of the common feature requests the ASP.NET team continually received was the ability for Forms Authentication to support cookieless authentication, that is, to not require an HTTP cookie. This is just what the team has provided in ASP.NET 2.0.

Enabling Cookieless Forms Authentication

Cookieless Forms Authentication is enabled within the `machine.config` file or the `web.config` file of your application by setting the new `cookieless` attribute (see Listing 6.21).

[19] This implementation is similar to what is done within all the ASP.NET Starter Kits.

LISTING 6.21. Default Configuration for Forms Authentication

```
<configuration>
  <system.web>
    <authentication mode="Forms">
      <forms name=".ASPXAUTH"
             loginUrl="login.aspx"
             protection="All"
             timeout="30"
             path="/"
             requireSSL="false"
             slidingExpiration="true"
             defaultUrl="default.aspx"
             cookieless="UseCookies" />
    </authentication>
  </system.web>
</configuration>
```

The `cookieless` attribute has four possible values:[20]

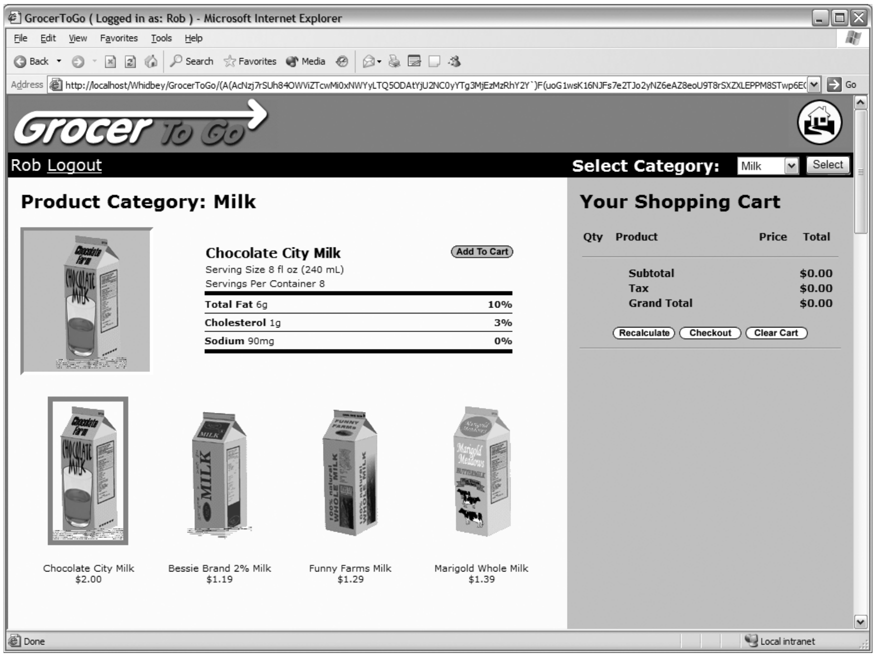
- `UseUri`: Forces the authentication ticket to be stored in the URL.
- `UseCookies`: Forces the authentication ticket to be stored in the cookie (same as ASP.NET 1.0 behavior).
- `AutoDetect`: Automatically detects whether the browser/device does or does not support cookies.
- `UseDeviceProfile`: Chooses to use cookies or not based on the device profile settings from `machine.config`.

If we set the `cookieless` value to `UseUri` within `web.config` and then request and authenticate with Forms Authentication, we should see something similar to what Figure 6.15 shows within the URL of the requested page.

Below is the requested URL—after authentication—in a more readable form:

```
http://localhost/Whidbey/GrocerToGo/(A(AcNzj7rSUh84OWViZTcwMioxNWYyLTQ5
ODAtYjU2NCoyYTg3MjEzMzRhY2Y`)F(uoG1wsK16NJFs7e2TJo2yNZ6eAZ8eoU9T8rSXZX
LEPPM8STwp6EONVtt4YCqEeb-9XDrrEpIHRpOOlKh8rO9foAhP6AXWwL*obM bxYcfZ
c`))/default.aspx
```

[20] In case you forget the values, an incorrect value set for `cookieless` will cause an ASP.NET error page to be generated that lists the acceptable values.



http://localhost/Whidbey/GrocerToGo/(A(AcNzj7rSUh84OWViZTcwMi0xNWYyLTQ5ODAtYjU2NC0yYTg3MjEzMzRhY2Y`)F(uoG1wsK16NJFs7e2TJo2yNZ6eAZ8eoU9T8rSXZXLEPPM8STwp6E(

Product Category: Milk

Chocolate City Milk
Serving Size 8 fl oz (240 mL)
Servings Per Container 8

Total Fat 6g	10%
Cholesterol 1g	3%
Sodium 90mg	0%

Chocolate City Milk
$2.00

Bessie Brand 2% Milk
$1.19

Funny Farms Milk
$1.29

Marigold Whole Milk
$1.39

Your Shopping Cart

Qty	Product	Price	Total
	Subtotal		$0.00
	Tax		$0.00
	Grand Total		$0.00

FIGURE 6.15. Cookieless Forms Authentication

The Web Site Administration Tool

Administration of ASP.NET applications has always been easy, although diving into the XML-based configuration file isn't the most user-friendly way to do it. For the 2.0 release of ASP.NET, there is the Web Site Administration Tool, which allows configuration of a Web application via an easy browser interface.

The Web Site Administration Tool is useful for two main reasons. First, it abstracts the XML configuration into an easy-to-use interface, and second, it provides administration features via a browser. This means that for remote sites (such as those provided by a hosting company), it's easy to administer an application without having to edit the configuration file (e.g., to add new security credentials) and then upload it.

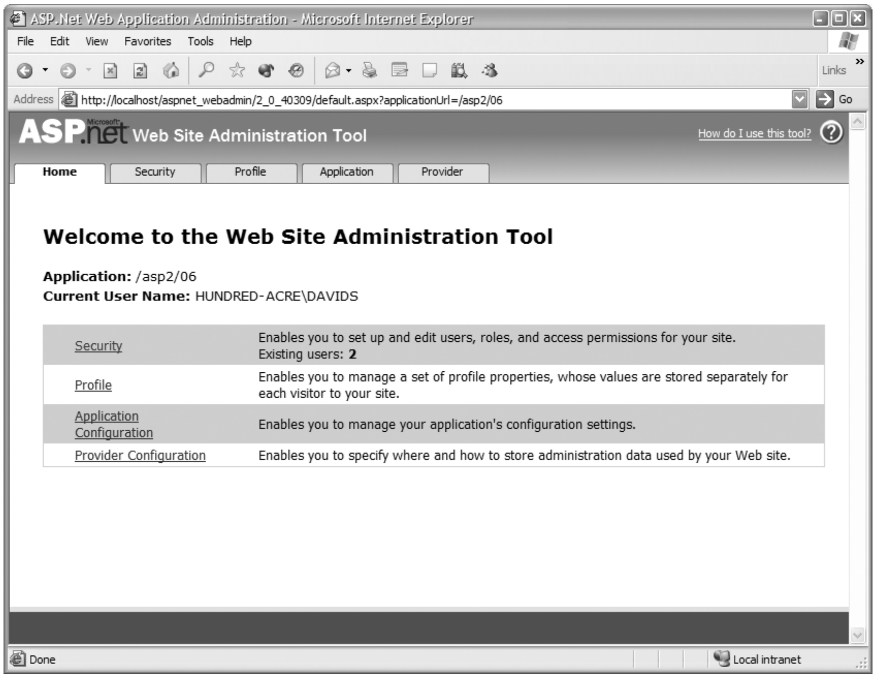
The Web Site Administration Tool is available for each directory configured as an application, by way of a simple URL:

```
http://website/WebAdmin.axd
```

This presents you with a home page and menu consisting of five main options.

The Home Page

The Home page, shown in Figure 6.16, details the current application and security details, as well as links to the other main sections.


ASP.Net Web Application Administration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Links
Address http://localhost/aspnet_webadmin/2_0_40309/default.aspx?applicationUrl=/asp2/06
Go
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Profile
Application
Provider
Done
Local intranet

Welcome to the Web Site Administration Tool

Application: /asp2/06
Current User Name: HUNDRED-ACRE\DAVIDS

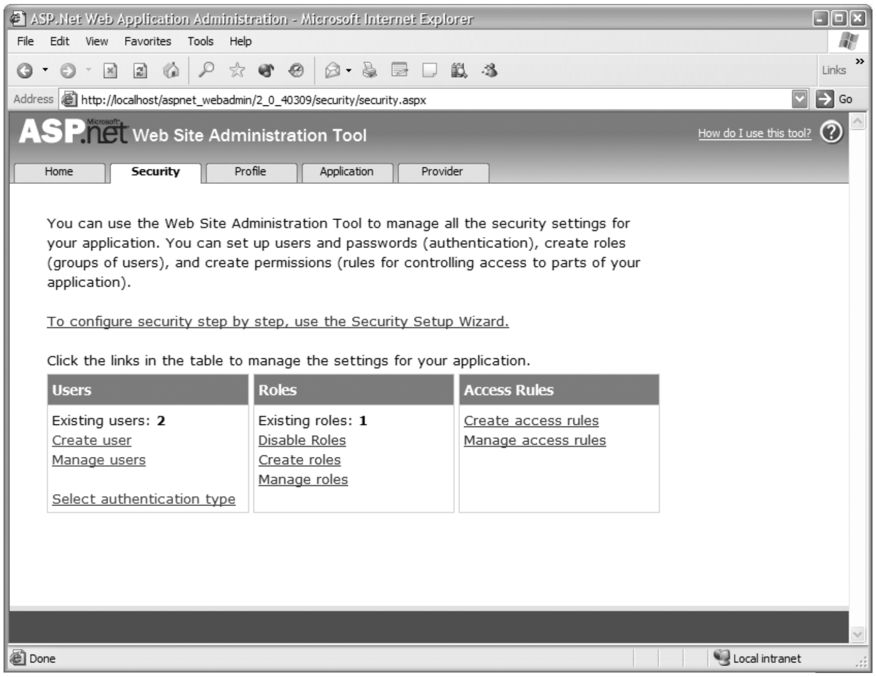
Security	Enables you to set up and edit users, roles, and access permissions for your site. Existing users: **2**
Profile	Enables you to manage a set of profile properties, whose values are stored separately for each visitor to your site.
Application Configuration	Enables you to manage your application's configuration settings.
Provider Configuration	Enables you to specify where and how to store administration data used by your Web site.

FIGURE 6.16. The Web Site Administration Tool Home page

The Security Page

The Security page, shown in Figure 6.17, offers two options for configuring security. The first is a wizard that takes you through the following steps:

1. **Select Access Method,** which defines whether the application is available from the Internet (in which case Forms Authentication is used) or from the LAN (in which case Windows Authentication is used)
2. **Specify Data Source,** where you can specify the database (Access or SQL Server) that will store the user credentials and what details are required (e.g., unique e-mail address, allow password retrieval, and so on)


ASP.Net Web Application Administration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Links
Address http://localhost/aspnet_webadmin/2_0_40309/security/security.aspx
Go
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Profile
Application
Provider
Done
Local intranet

You can use the Web Site Administration Tool to manage all the security settings for your application. You can set up users and passwords (authentication), create roles (groups of users), and create permissions (rules for controlling access to parts of your application).

To configure security step by step, use the Security Setup Wizard.

Click the links in the table to manage the settings for your application.

Users	Roles	Access Rules
Existing users: **2** Create user Manage users Select authentication type	Existing roles: **1** Disable Roles Create roles Manage roles	Create access rules Manage access rules

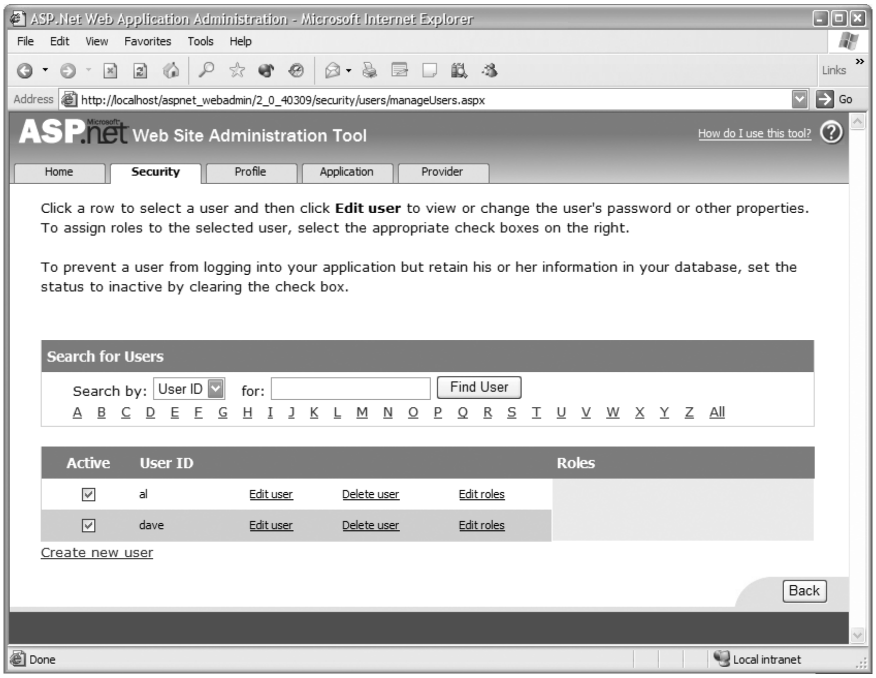
FIGURE 6.17. The Web Site Administration Tool Security page

3. **Define Roles,** where you can optionally specify Authorization roles
4. **Add New Users,** which allows addition of new users and allocation to roles
5. **Add New Access Rules,** which defines which files and folders users and roles have permissions for
6. **Complete,** to indicate that the security settings have been configured

The second option is for configuration of security that has already been enabled. If configured, the details are shown at the bottom of the main security page (see the following subsection for more information).

Security Management

Once you have initially set up security (or selected the second option titled Security Management on the main Security page), the Security page allows management of users, roles, and permissions without the use of the wizard.


ASP.Net Web Application Administration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Links
Address
Go
ASP.net Web Site Administration Tool
How do I use this tool?
Home Security Profile Application Provider
Back
Done
Local intranet

http://localhost/aspnet_webadmin/2_0_40309/security/users/manageUsers.aspx

Click a row to select a user and then click **Edit user** to view or change the user's password or other properties. To assign roles to the selected user, select the appropriate check boxes on the right.

To prevent a user from logging into your application but retain his or her information in your database, set the status to inactive by clearing the check box.

Search for Users

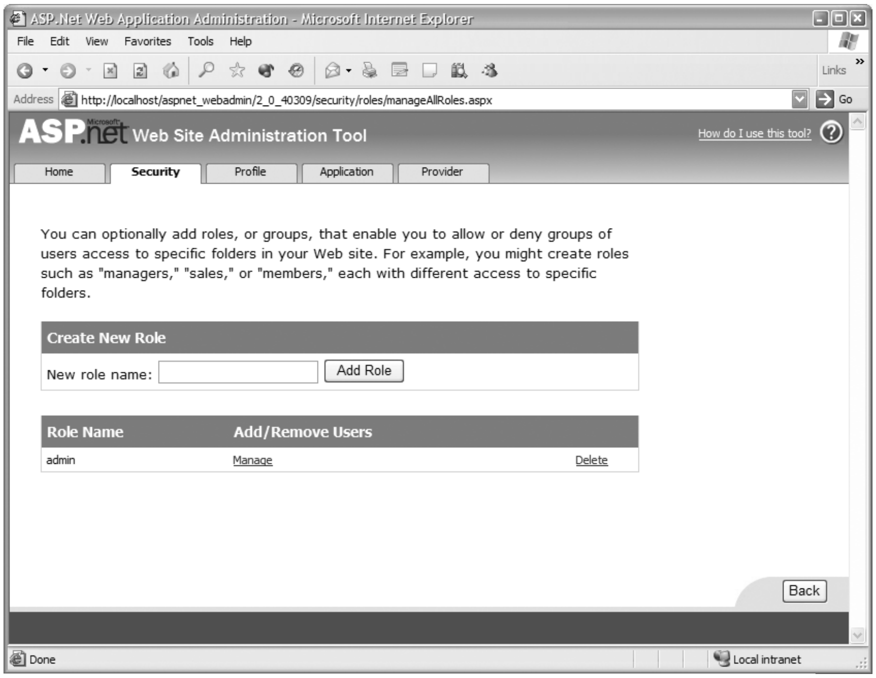
Search by: User ID for: [Find User]

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z All

Active	User ID				Roles
☑	al	Edit user	Delete user	Edit roles	
☑	dave	Edit user	Delete user	Edit roles	

Create new user

FIGURE 6.18. Web Site Administration Tool user configuration


ASP.Net Web Application Administration - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Links
Address
Go
ASP.net Web Site Administration Tool
How do I use this tool?
Home Security Profile Application Provider
Back
Done
Local intranet

http://localhost/aspnet_webadmin/2_0_40309/security/roles/manageAllRoles.aspx

You can optionally add roles, or groups, that enable you to allow or deny groups of users access to specific folders in your Web site. For example, you might create roles such as "managers," "sales," or "members," each with different access to specific folders.

Create New Role

New role name: [Add Role]

Role Name	Add/Remove Users	
admin	Manage	Delete

FIGURE 6.19. Web Site Administration Tool role configuration

For example, consider the users added earlier in the chapter. If we select the Manage Users options, we see the User Management features shown in Figure 6.18.

Likewise, selecting Manage Roles allows you to customize roles and members, as shown in Figure 6.19.

Other Pages

Three other pages are used in the Web Site Administration Tool:

- **Profile**, which allows configuration of the Personalization Profile (see Chapter 7)
- **Application**, which allows configuration of application settings, site and page counters, SMTP settings, and debugging and tracing (see Chapter 13)
- **Provider**, which allows configuration of the data provider or providers to be used in the application (see Chapter 13)

SUMMARY

We've sampled only some of the new security capabilities in ASP.NET 2.0. The Membership and Role Manager features are specifically designed to solve problems the ASP.NET team saw developers addressing over and over again. Although both complement and can be used easily with Forms Authentication, they were also designed to work independently—independently of one another and independently of Forms Authentication. Furthermore, both support the provider design pattern. This design pattern allows you to take complete control over how and where the data used for these features is stored. The provider design pattern gives you ultimate control and flexibility, because you can control the business logic, while developers can learn a simple, friendly, and easy-to-use API.

Although writing code using Membership has become more concise, there are also now cases where no code is required. The new security server controls make many scenarios, such as login or password recovery, much easier to implement. The other security-related server controls simply save you the time formerly required to write code to perform simple tasks such as checking who is logged in.

The cookieless support for Forms Authentication means you don't have to require the use of cookies for authenticating users—something many of you have been requesting.

Finally, the Web Site Administration Tool provides a simple way to administer site security without building custom tools.

Now it's time to extend the topic of users interacting with a site and look at how sites can be personalized.