Chapter 1: Security: Using Login Controls

In This Chapter

- **Understanding authentication and authorization**
- **Using the Security Administration tool**
- **Restricting access**
- **Handling logins and lost passwords**
- **Managing users and roles programmatically**

Most of us feel uneasy about implementing Web site security, perhaps because it's hard to be 100% sure that you've got it right. Inadvertently allowing the Internet's bad guys to get in could be a Career Limiting Move (CLM) or worse. Therefore, it's comforting to put security in the hands of people who've done it before. Enter Microsoft's ASP.NET team. The team realized that so many of us were reinventing the security wheel (sometimes creating an oval wheel, out of whack) that it made sense to build membership and login capabilities directly into ASP.NET 2.0.

Out of the box, we have all the tools we need to let people log in to the site, view what we allow them to view, and recover their lost passwords. Our goal in this chapter is to implement security while writing as little code as possible. We can do this by leveraging the standard authorization tools and functions in ASP.NET.

As you work with membership terminology, note that *roles* refer to groups or categories of users. In addition, the terms *users* and *members* are interchangeable.

Understanding Authentication and Authorization

Authentication and authorization are easy to confuse. It might help to look at how these concepts work in a restaurant. In our scenario, areas such as the restaurant's entrance, dining room, and kitchen represent Web pages with different access levels.

Anyone off the street can open the restaurant door and stand in the entrance. In that location, a visitor can look around while remaining completely

anonymous because no one needs to know who he or she is. There's no need to grant any approval for him or her to be there.

Our restaurant visitor, John Oliver, passes through the entrance. He intends to eat in the restaurant's dining room and has a reservation. He presents himself at the maitre d's stand at the dining room door. Up until now, Mr. Oliver has been anonymous and unchallenged. The "security" context changes at this point. To claim his reserved table Mr. Oliver must lose his anonymity. He identifies himself by telling the maitre d' his name. In this social context, Mr. Oliver's word is sufficient proof for the maitre d' to validate the person in front of him as Mr. Oliver. The maitre d' could have asked for identification, but that would drive Mr. Oliver away. It is, after all, just a restaurant. Mr. Oliver has been *authenticated* at the restaurant because the restaurant employee knows the person with whom he's dealing.

Next, the maitre d' looks for Mr. Oliver's name on the evening's reservation list, which is like a database. The name appears on the list, confirming that the guest may sit at the table set aside for Mr. Oliver. You could say that Mr. Oliver has been *authorized* to enter the dining room and sit at a table.

You can see that authentication and authorization are different issues. *Authentication* establishes who you are; *authorization* establishes what you can do.

Authentication can also establish a pecking order for different groups of people. For example, although Mr. Oliver has been authenticated and authorized to enter the dining room, he has not been authorized to enter the VIP room unless Mr. Oliver's name appears on the VIP list. Nor can he enter the kitchen unless he is a member of the staff.

Implementing Forms Authentication

We're going to walk through the construction of a barebones Web site that uses forms authentication, the normal mode for a password-protected Internet Web site. Forms authentication requires the visitor to submit a valid username and password to gain access to protected pages. If the credentials are valid, the Web server issues a temporary cookie to the browser that acts as a token to allow entry into other protect pages without forcing the user to type the credentials each time.

Before putting a shovel in the ground, it might help to understand the roles of the Web pages in our sample application.

- `default.aspx`**:** This is the entrance to our Web site. As with the restaurant, anyone can browse here anonymously.
- `login.aspx`**:** This is the page where visitors present their credentials for validation. In the restaurant example in the previous section, the

customer identifies himself to the maitre d' to claim a reservation. Behind the scenes, we verify the username and password — just as the maitre d' checks his reservation list.

- `reserved.aspx`**:** Browsers can only view the contents of this page if they have specific permission. In the restaurant scenario, this is the reserved table. Before a customer gets to this place, the maitre d' knows you and specifically grants you access.
- `register.aspx`**:** This is where visitors to our site can request access to private pages. They must provide information about themselves before approval. The comparable step in the restaurant example is giving your name and phone number when making a reservation.
- `regular.aspx`**:** We might allow anonymous browsers to view this page under certain conditions. In a restaurant, this would be an unreserved table in the dining room.

Creating the Web site

The example we are creating in this chapter uses Visual Web Developer Express (VWDE) to create a file-based Web site. We use the Express edition of SQL Server 2005 as the database engine. For maximum simplicity, we use the Visual Basic (VB) language and all the code (both HTML and server-side) goes into the `.aspx`. file. Let's get started!

1. **In Visual Web Developer Express, from the File menu, click New Web Site.**
2. **In the New Web Site dialog box, under Templates, select ASP.NET Web Site.**
3. **In the Location boxes, select File System and enter `c:\resto` as the location for the site.**
4. **In the Language box, select Visual Basic.**
5. **Click OK.**

   As shown in Figure 1-1, VWDE creates a site that includes three files: `Default.aspx`, `Default.aspx.vb`, `web.config` and a folder, `App_Data`.

Figure 1-1: Solution Explorer after creating a new Web site.

6. **Delete** `Default.aspx`**.**

   The default page uses the code-behind model rather than the one-page model that we're using. You build the pages from scratch in the next section.

Adding pages to the resto Web site

Because the chapter is about security rather than design, I won't deal with creative aspects of pages here. Let's just add some plain ASP.NET pages to the site so we have something to configure. To do so:

1. **In Solution Explorer, select the root of the site and right-click it to bring up the context menu.**
2. **Click Add New Item.**

   The Add New Item dialog box opens, as shown in Figure 1-2.

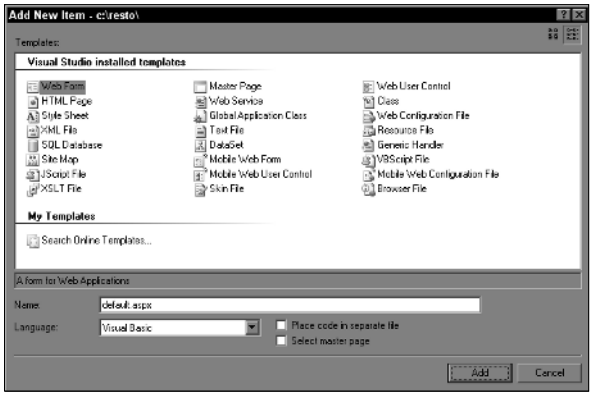

Figure 1-2: The Add New Item dialog box.

1. **From the Visual Studio Installed Templates select Web Form.**
2. **In the Name box, enter the name of the start page,** `default.aspx`**.**
3. **In the Language box, select Visual Basic.**
4. **Uncheck the check boxes for Place Code In Separate File and Select Master Page.**
5. **Click Add.**

   The new ASP.NET page appears in Solution Explorer.

6. **Repeat the preceding steps to add the following pages:** `login.aspx`**,** `reserved.aspx`**,** `register.aspx`**, and** `regular.aspx`**.**

   When you're finished, Solution Explorer should look like Figure 1-3.

Figure 1-3: Solution Explorer after adding files.

That takes care of creating the raw pages. We add functionality to the pages in subsequent procedures. Before getting to that, however, we have to fix the site's directory structure. We need a directory for the exclusive use of logged in members. Although it's possible to secure individual pages in the `root` folder, ASP.NET's membership features are easiest to apply to whole directories rather than pages. To create the directory, follow these steps:

1. **In Solution Explorer, select the root of the site and right-click it to bring up the context menu.**
2. **Select the New Folder item.**
3. **Name the new folder** `members`**.**
4. **Drag** `reserved.aspx` **from the** `root` **folder of the Web site and move it into the** `members` **folder.**

We encounter the `members` folder again when we set permissions. First, we have to set up ASP.NET's membership features.

Implementing membership features

Our Web site needs a database to store user information and credentials. When a person logs in, we look up the name and credentials before deciding what pages that person can visit.

You need SQL Server 2005 Express on your workstation for these procedures. (If you haven't installed a copy, now's a good time to do so.)

ASP.NET provides all you need for basic authentication and user management — with little or no code — thanks to the Web Site Administration tool. The

hardest part is knowing where to click; the way to get a handle on that is to configure membership for our site. You should still have the `resto` project open. To configure membership for our site, follow these steps:

1. **In the IDE (that's the VWDE development environment), from the Website menu, click ASP.NET Configuration.**

   The ASP.NET development server starts up and navigates to the Web Application Administration startup page, as shown in Figure 1-4.

Figure 1-4: Security tab of the Web Site Administration Tool.

2. **Click the Security tab.**

   The page may take a few seconds to appear the first time.

3. **Click the link marked Use The Security Setup Wizard to configure security step by step.**

4. **On the Welcome page, click Next.**

5. **On the Select Access Method page, select the radio button for From The Internet and then click Next.**

6. **On the Advanced provider settings page, click Next.**

7. **On the Define Roles page, make sure the check box is cleared and then click Finish.**

   The Web Site Administration Tool displays the Security tab. We deal with roles in section "Assigning users to roles," later in this chapter.

Notice that we didn't complete all the wizard steps. That's not a problem because you can restart the wizard at any time to explore its capabilities. There are also other paths to the same functions, such as creating a user.

Before moving on, let's investigate what the wizard has done for us. In your IDE, open Solution Explorer and click the Refresh button. Expand all the nodes and look under the `App_Data` folder — you should see a database file called `ASPNETDB.MDF`, as shown in Figure 1-5.

Figure 1-5: A database file, ASPNETDB.MDF, added to the site.

So far we haven't written any code — but we've managed to create a database that includes numerous tables and stored procedures. You can investigate the database using the Database Explorer. Just go to View➪Database Explorer. Expand the nodes, as shown in Figure 1-6. (We add data in the section, "Creating users," later in this chapter.)

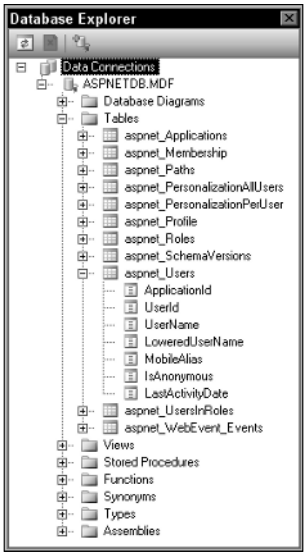

Figure 1-6: Database Explorer shows the membership tables.

Creating users

Our database is now ready for us to add some users, so let's add two user accounts. Once again, we turn to the Web Site Administration tool. To add users, follow these steps:

1. **Navigate to the Security tab (Website➪ASP.NET Configuration and select the Security tab).**
2. **In the table at the bottom of the page, locate the Users column and click the Create User hyperlink.**

   The Create User page appears.

3. **Fill in the user's name, password, e-mail address, security question, and security answer, as shown in Figure 1-7.**

   You can make up your own data, but you'll find it easier to follow along later with these values:

   - **Username:** JohnOliver
   - **Password:** OliverJoh!
   - **E-mail:** jo@nowhere.com
   - **Security Question:** Your dog's name?
   - **Security Answer:** Goldie

   ASP.NET requires that your passwords include a combination of upper- and lowercase letters and at least one non-alphanumeric character, such as a punctuation symbol.

Figure 1-7: The Create User page in the Web Site Administration Tool.

4. **Click Create User.**

   The confirmation message appears.

5. **Click Continue.**

Fixing connection woes

ASP.NET's Web Site Administration Tool does a lot of work behind the scenes. When it creates the membership database, it also builds a database connection string. You can view the string by opening the Database Explorer, selecting ASPNETDB.MDF, and opening the Properties page (that's the F4 key in the default environment).

For example, on my machine the connection string looks like this:

```
"Data
          Source=.\SQLEXPRESS;AttachD
          bFilename=C:\resto\App_Data
          \ASPNETDB.MDF;
Integrated Security=True;
User Instance=True"
```

If you're having trouble connecting to a database that you've created yourself, you might get going again by adapting ASP.NET's membership settings. Also, keep in mind that the IDE recognizes databases more readily when you put the file in the special `App_Data` folder.

6. **Repeat the preceding steps and create another user with the following values:**
   - **Username:** JillAnon
   - **Password:** AnonJill!
   - **E-mail:** ja@nowhere.com
   - **Security Question:** How high is Up?
   - **Security Answer:** Very

Creating access rules for the pages

Recall that our goal is to allow anyone to browse the default page but permit only specific users to view pages in the `members` subdirectory. To do this, we have to create some access rules.

Allowing anonymous users access to the root

The first task is to ensure that everyone can reach the pages in the root of the Web, including the home page, `default.aspx`. To do so, follow these steps:

1. **Navigate to the Security tab (Website➪ASP.NET Configuration and select the Security tab).**
2. **In the table at the bottom of the page, locate the Access Rules column and click the Create Access Rules hyperlink.**

   The Add New Access Rule page appears.
3. **In the left column, select the root folder (`resto`).**

4. **In the right column, under Rule Applies To, select the Anonymous Users radio button.**
5. **In the right column, under Permission, select the Allow radio button.**

   The resulting access rule should look like Figure 1-8.

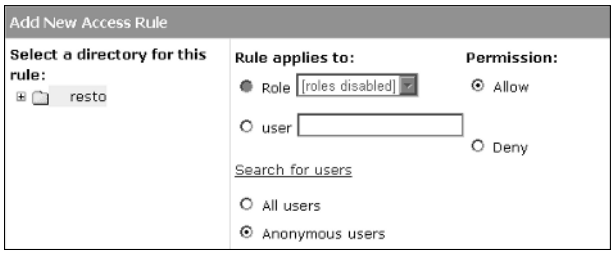

Figure 1-8: Allowing access to anonymous users in the root.

Denying access for all users

Our next step is to secure the `members` subdirectory by keeping everyone out. This exclusion includes anonymous users and users who are logged in. To secure the members subdirectory:

1. **In the Web Site Administration Tool, navigate to the Security tab (Website➪ASP.NET Configuration and select the Security tab).**
2. **Click Create access rules.**

   The Add New Access Rule page appears.
3. **In the left column, expand the root folder (`resto`) and select the subdirectory called `members`.**
4. **In the right column, under Rule Applies To, select the All Users radio button.**

   We're creating a rule that applies to everyone.
5. **In the right column, under Permission, select the Deny radio button.**

   The resulting access rule should look like Figure 1-9.

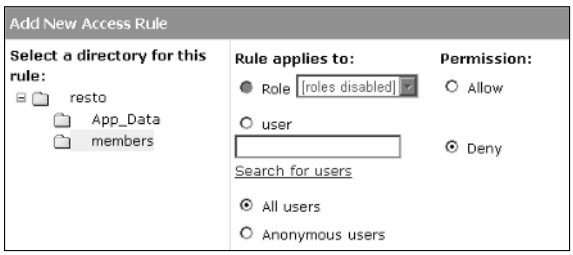

Figure 1-9: Denying access to all users.

6. **Click OK. The Security page reappears.**

As of now, nobody can see pages in the `members` folder. We have to add one more rule to make the folder usable by that one special user, `John Oliver`.

Allowing access to one user — John Oliver

Of the two users we created previously, only `John Oliver` is allowed to access the `members` folder, including the `reserved.aspx` page. The following steps provide access to him after he logs in:

1. **Navigate to the Security tab (Website➪ASP.NET Configuration and select the Security tab).**
2. **Click Create Access Rules.**
3. **Select the subdirectory called `members`.**
4. **Select the User radio button.**
5. **Click the Search for Users link.**

   The Search for Users page lists the users you added previously. Figure 1-10 shows part of the page with the usernames.

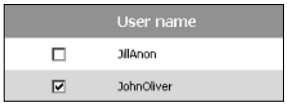

Figure 1-10: Partial view of Search for Users page.

6. **Check the check box for JohnOliver and click OK.**

   The browser returns to the Add New Access Rule page with the username `JohnOliver` in the text box, as shown in Figure 1-11.

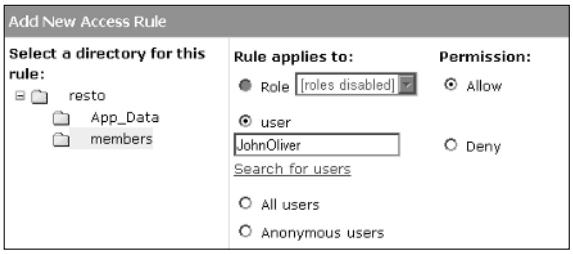

Figure 1-11: Allowing access to John Oliver.

7. **In the Permission area, select the Allow radio button.**
8. **Click OK.**

Reviewing the access rules

You now have two rules in effect for the `members` subdirectory — *deny all* and *allow John Oliver*. You can (and should) review the rules to confirm that they'll produce the desired result. To review the access rules, follow these steps:

1. **Navigate to the ASP.NET Configuration Security tab.**
2. **Click Manage Access Rules.**

   The Manage Access Rules page opens, as shown in Figure 1-12.

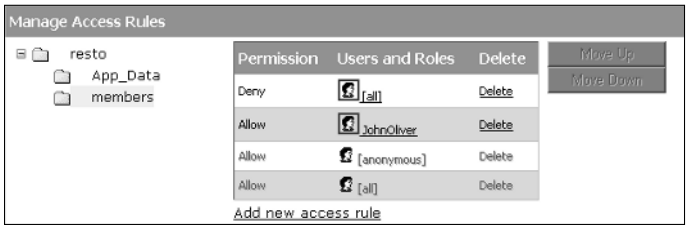

Figure 1-12: The Manage Access Rules page.

3. **Expand the `resto` node and click the `members` subdirectory.**

   The existing permissions for the subdirectory appear.

It might not be obvious on first viewing, but an analysis of Figure 1-12 shows that we have a problem. John Oliver is denied entry — even though his permission is listed as `Allow`.

The rule at the top of the list takes precedence over the rules below it. Likewise, the second rule in the list overrides conflicting instructions further down.

Here you can see that Deny All is king of the castle and overshadowing Allow `JohnOliver`. It may be hard to see in the figure, but the bottom two rules are dimmed (grayed out) because they are inherited from the parent directory.

To fix the hierarchy, move John Oliver's Allow permission higher than the Deny All entry. Here's how:

1. **On the Manage Access Rules page, in the Users and Roles column, click the username `JohnOliver`.**

   The Move Up button becomes active.

2. **Click Move Up.**

   The Allow rule for username `JohnOliver` moves to the top of the table and overrides the rules below it. Figure 1-13 shows the correct order for the access rules.

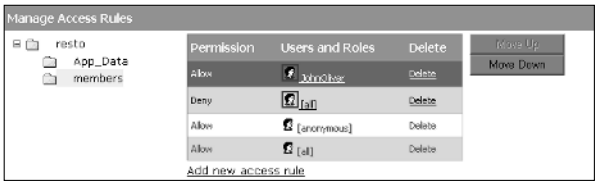

Figure 1-13: Putting access rules in order.

3. **Click Done.**

We revisit access rules in the section "Assigning users to roles," later in this chapter, to add a role. However, it's time to put the rules to use and demonstrate how they affect ASP.NET pages.

Using the Login control

When Microsoft's ASP.NET team set out the goals for the 2.0 version, they wanted to reduce the amount of code that developers have to write by 70 percent. The `Login` control contributes to the code reduction by providing tons of code-free functionality with its default settings.

By the way, have you noticed that we haven't written *any* code yet in this chapter?

Adding the Login control to the page

To allow John Oliver to browse to `reserved.aspx` in the `members` subdirectory, you have to provide him with the ASP.NET `Login` control as a way to present his credentials for authentication.

1. **In Solution Explorer, open `login.aspx` in Design view.**
2. **From the `Login` tab of the toolbox, drag the `Login` control (the one with an icon showing a padlock and a person) and drop it on the design surface.**

   Figure 1-14 shows the control and its Smart Tasks menu.

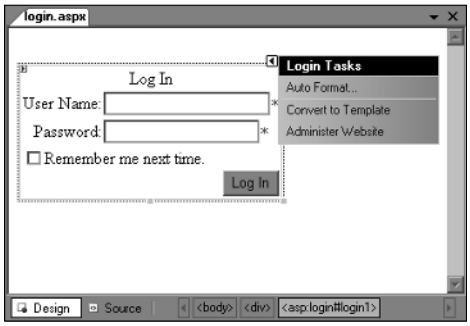

Figure 1-14: The Login control.

Script your login

When you test forms authentication, it's highly likely that you'll access pages many times with different usernames and passwords. The `Login` control is a great convenience at design time — but at runtime, you still have to enter the credentials. Copying and pasting the username and password works okay, but even that becomes tedious after a few logins.

You can semi-automate the logins with a little client-side script. I threw together a JavaScript routine to paste into `login.aspx` during development. When you select a username from the drop-down list, the script pushes that username and its password into the appropriate fields in the `Login` control. The sooner you implement this arrangement, the more time you'll save.

In Source view, put the following in the `<head>` of `login.aspx`:

```
<script type="text/javascript">
function autologin()
{
var username =
          document.getElementById("Lo
          gin1_UserName");
var pwd =
          document.getElementById("Lo
          gin1_Password");
var cntrl =
          document.getElementById("Se
          lect1");
username.value=cntrl.options[cntrl.se
          lectedIndex].text;
pwd.value=cntrl.options[cntrl.selecte
          dIndex].value;
}
</script>
```

Right after the `<body>` tag, insert a drop-down list with the sample usernames and passwords:

```
<select id="Select1"
          onchange="autologin()">
    <option value=""></option>
    <option
          value="OliverJoh!">JohnOliv
          er</option>
    <option
          value="AnonJill!">JillAnon<
          /option>
</select>
```

You might need to adjust the IDs passed to the `getElementById()` function to match the control IDs that ASP.NET generates on the rendered page. The values here assume that your `Login` control's name is `Login1`.

The `Login` control has dozens of properties, including many to customize the text that appears. For example, the `FailureText` property value is a polite string, `"Your login attempt was not successful. Please try again."` You can change the text to wording more appropriate for your site, such as `"Nope, that's not it!"`

Testing the Login control

So far we've generated a user database, added two users to it, added a folder and some ASP.NET pages to our site, configured the permissions, and implemented the login function. Finally, we can test the security by trying to access the restricted page — using three different personas.

The first persona is the anonymous user. We allow this user to browse to pages in the root of the Web site but deny access to the `members` subdirectory. To test this persona:

1. **In Solution Explorer, in the `members` folder, right-click `reserved.aspx`.**
2. **From the context menu, click View in Browser.**

   The browser opens, but instead of the `reserved.aspx` page, you `see login.aspx`.

This proves that the anonymous user was not allowed to see the page. In fact, ASP.NET's default behavior automatically redirected the browser to the login page. Notice the URL in the browser, repeated in the next line:

```
http://localhost:3235/resto/login.aspx?
ReturnUrl=%2fresto%2fmembers%2freserved.aspx
```

The portion after `ReturnUrl=` is the page the anonymous user tried to reach. The `%2f` parts are escape codes for the forward slash, so the URL translates to:

```
/resto/members/reserved.aspx
```

If the login is successful, `reserved.aspx` is the user's intended destination.

For the next attempt to access the `members` subdirectory, we call on a known user, `JillAnon`. Testing the login for our known user looks like this:

1. **Browse to `reserved.aspx` in the `members` folder.**

   The browser redirects to the login page.
2. **In the User Name box, type `JillAnon`.**

3. **In the Password box, type** `AnonJill!` **(case-sensitive).**
4. **Click Log In.**

   The user failed to access `reserved.aspx`. The login page reappears.

Actually the login only *appears* to have failed. In the section "Using the LoginName control," later in this chapter, we show that `JillAnon` did in fact log in — and can navigate to other pages. ASP.NET's default behavior is to bounce browsers to the login page when they're denied access to a page. That can be changed (as detailed in the next section).

Now to test whether the only user who *has* access to `reserved.aspx` can *get* there. Here goes:

1. **Browse to** `reserved.aspx` **in the** `members` **folder.**

   The browser redirects to the login page.
2. **In the User Name box, type** `JohnOliver`**.**
3. **In the Password box, type** `OliverJoh!` **(case-sensitive).**
4. **Click Log In.**

   The browser navigates to . . . a blank page!

The page is blank because we didn't put anything in it — but look at the address in the browser: It landed on `reserved.aspx`. You can make the target page more obvious by opening `reserved.aspx` in Design view and adding identifying text. Browse to the page again and you'll see something like Figure 1-15.

Figure 1-15: Browsing to reserved.aspx.

Using the LoginName control

When we tested the login by `JillAnon` in the previous section, it *appeared* to have failed. There was nothing obvious to indicate success. You can fix that by adding the `LoginName` control to a page. If the user is logged in, it displays the name. Here's the fix:

1. **Open** `default.aspx` **in Design view.**
2. **From the** `Login` **tab of the toolbox, drag the** `LoginName` **control to the design surface.**
3. **Browse to** `login.aspx`.

   If you browse directly to `default.aspx`, you do so as the anonymous user and won't see a name.
4. **Log in as** `JillAnon` **with the password** `AnonJill!` **(case-sensitive).**

   The default page appears with the username, as shown in Figure 1-16.

Figure 1-16: The LoginName control displays the username.

The `LoginName` control is handy for personalizing Web pages. People tend to forget themselves when browsing exciting Web pages; this way you can show them who they are at all times while they're logged in.

Using the LoginStatus control

The `LoginStatus` control shows more than whether a user is logged in or out. It detects the user's status and, based on that, creates links to log out or log in.

1. **Open** `regular.aspx` **in Design view.**
2. **From the** `Login` **tab of the toolbox, drag the** `LoginStatus` **control to the design surface.**
3. **Browse to** `regular.aspx` **as an anonymous user**.

   The page shows a Login hyperlink.
4. **Click Login.**

   The browser navigates to `login.aspx`.
5. **Log in as** `JillAnon` **with the password** `AnonJill!` **(case-sensitive).**

   You return to `regular.aspx`, but this time the hyperlink reads Logout.

6. **Click Logout.**

   Behind the scenes, ASP.NET cancels your authentication (that is, logs you out); the `LoginStatus` control reflects this change by displaying the Login hyperlink.

Be sure to provide a way for the user to log out from your secure pages. Often, when people navigate away from secure pages, they think (mistakenly) that they've logged out. If they leave the browser open during a break, a ne'er-do-well could browse back to the secure page while the authentication is valid. Using the Logout link forces the user to re-authenticate.

Using the LoginView control

`LoginView` is a templated control that lets you show completely different content to a user who has logged in and one who has logged out. Templated controls let you go wild with your own customizations. That's because you add your own content rather than just manipulating the properties that Microsoft provides. The templates, `AnonymousTemplate` and `LoggedInTemplate`, act as containers for all kinds of markup, including ASP.NET controls. Here's a short demonstration:

1. **Open `regular.aspx` in Design view.**
2. **From the `Login` tab of the toolbox, drag the `LoginView` control to the design surface.**

   The control defaults to the `AnonymousTemplate`.

3. **Drag an ASP.NET `Label` control from the toolbox, drop it inside the `LoginView` control's outline, and set the label's text to `"You are not worthy."`**
4. **Select the `LoginView`, open its Smart Tasks menu, and select `LoggedIn Template` from the drop-down list, as shown in Figure 1-17.**

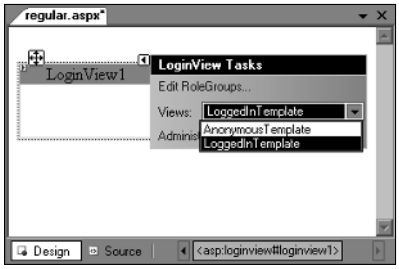

Figure 1-17: Choosing a view in the LoginView control.

5. **Drag an ASP.NET `Hyperlink` control onto the `LoginView`.**

6. **Set the hyperlink's `Text` property to `Members` and the `NavigateUrl` property to `~/members/reserved.aspx`.**
7. **Browse to `regular.aspx` as an anonymous user.**

   The page displays the contents of the label.
8. **Log in at `login.aspx` and then browse to `regular.aspx`.**

   The page displays the hyperlink.

As you've seen here, you can fill the templates with entirely different content — such as error messages for logged-in users.

Using the PasswordRecovery control

IT departments know too well that forgotten passwords are among the most common support tasks that users need (especially after vacations). To save you the hassle of manually resetting forgotten passwords, ASP.NET offers the self-service `PasswordRecovery` control. It sends the password to the e-mail address used at registration, and it works — as long as the person knows his or her username.

Configuring the SMTP settings

The `PasswordRecovery` control requires access to a Simple Mail Transport Protocol (SMTP) server to actually send the e-mail containing the password. Most network operators place restrictions on the use of their SMTP server to deter spammers. Most mail servers require authentication before sending e-mail. Some Internet service providers (ISPs) block individual users from accessing the commonly used ports for SMTP.

It's quite likely that you'll have to check with a system administrator if you want the exact SMTP settings. For that reason, we can only give you general instructions on configuring a Web site to use the `PasswordRecovery` control.

We start with the Web Site Administration tool:

1. **Navigate to the Application tab (Website➪ASP.NET Configuration and select the Application tab).**
2. **Click Configure SMTP e-mail settings.**
3. **Fill in the settings for your SMTP server.**

   You can refer to the following example data:

   - **Server Name:** `smtp.mydomain.com`
   - **smtp.mydomain.com:** 25
   - **E-mail:** `admin@mydomain.com`

- **Sender's username:** `pwdrequest@mydomain.com`
- **Sender's password:** pwd@#$@%!

4. **Click Save.**

If you don't get the SMTP e-mail settings to work after a few tries, check with your ISP before proceeding. Hitting the mail server with scads of incorrect data could get your e-mail privileges suspended — because you might be mistaken for a hacker.

Adding the PasswordRecovery control to a page

The second phase in setting up password recovery is adding the `PasswordRecovery` control to a Web page. Here's the drill, using the default configuration:

1. **Open `default.aspx` in Design view.**
2. **From the `Login` tab of the toolbox, drag the `PasswordRecovery` control to the design surface.**

Testing the password recovery feature

To test the functionality of the password recovery feature, follow these steps:

1. **Browse to `default.aspx` as an anonymous user (that is, not logged in).**

   Figure 1-18 shows the default appearance.

Figure 1-18: The Password-Recovery control.

2. **Type a registered username (for example, `JillAnon`) and then click Submit.**

   The Identity Confirmation page appears with the challenge question, as shown in Figure 1-19.

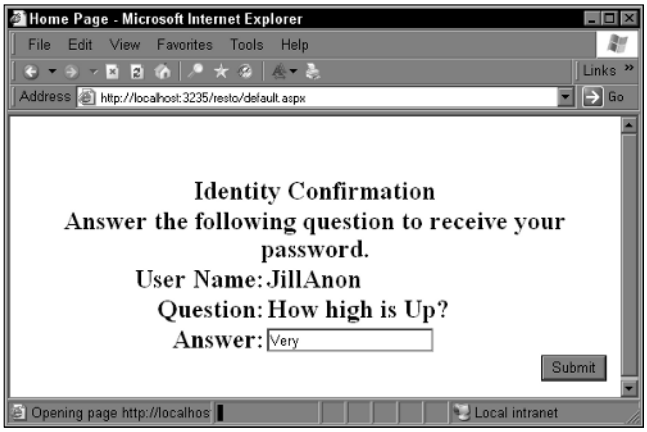

Figure 1-19: Answer the challenge question.

3. **Answer the challenge question (`JillAnon`'s answer is `Very`) and click Submit.**

   Behind the scenes, ASP.NET sends the password to the e-mail address in the database.

If the SMTP settings and permissions aren't correct, you'll probably see a timeout error message on the Web page.

We used the default settings for the `PasswordRecovery` control, but you can customize the question text, error messages, success messages, and the URL of the page to display after a password recovery.

Use the `MailDefinition` property to define the e-mail message including the e-mail's subject, content, format (plain text or HTML), priority, and sender name. If you're sending the message as HTML, you can include image files such as a background graphic or your site's logo. Add the images in the `EmbeddedObjects` collection editor (MailDefinition➪EmbeddedObjects).

Using the ChangePassword control

The `PasswordRecovery` control sends the password in unencrypted format, so users should change their password as soon as possible. (Many of us feel that we forget passwords because administrators force us to change them too often, but that's a rant for another time.)

The `ChangePassword` control simplifies the process while enforcing rules (default or custom) about password complexity. Like the other login controls, you can configure the appearance, text, and redirect page. To add a `ChangePassword` control:

1. **Add a new ASP.NET page called `chgpwd.aspx` to the root of your project.**
2. **From the `Login` tab of the toolbox, drag a `ChangePassword` control to the design surface.**
3. **In the properties page for the `ChangePassword` control, set the `DisplayUserName` property to `true`.**
4. **Browse to the page and fill in the text boxes using a registered username (for example, `JillAnon`) and password (`AnonJill!`), as shown in Figure 1-20.**

Figure 1-20: The Change Password control.

5. **Click Submit.**

   The `ChangePassword` function confirms the change.

Wrap the `ChangePassword` control inside a `LoggedInView` template of the `LoginView` control so users who aren't logged in won't see the control.

Assigning users to roles

Previously in the chapter, you created two test users. You gave username `JohnOliver` permission to browse to `reserved.aspx` in the restricted `members` subdirectory. Chances are, a group of users would have access to that resource rather than just one individual user, so it's far more efficient to manage them as a group. Enter ASP.NET roles. By creating a role (such as `EliteMember`), you can assign usernames to the role. If you want to give members of the `EliteMember` role access to a new portion of the site, you set the permissions for the role once — and all the elitists can enter immediately.

You use the Web Site Administration tool to manage roles and add usernames. In our scenario, we create a role, add users to it, and give the role rights to the `members` subdirectory. Here's how we do it:

1. **Navigate to the Security tab (Website➪ASP.NET Configuration and select the Security tab).**
2. **In the Roles column, click Enable roles.**

   The Create Or Manage Roles link is enabled.
3. **Click Create Or Manage Roles.**

   The Create New Role page appears.
4. **Type a new role name (for example, `EliteMember`) and click Add Role.**

   The new role appears in the list.
5. **Next to the role name, click Manage.**

   The Search for Users page appears.
6. **Search for all users by entering an asterisk (*) in the text box, and then click Find User.**

   The list of usernames appears.
7. **Under the User Is In Role column, select the check box for each user and then click Back.**

   The selected users are added to the `EliteMember` role.

The users are now in a role (group), but the role doesn't have permission to enter the `members` area. That's the task of the next section.

Giving permissions to a role

After you add users to a role, you can manage them as a collection. In this case, we want to give special permissions to the `EliteMember` role.

1. **Navigate to the Security tab by clicking Website➪ASP.NET Configuration and selecting the Security tab.**
2. **In the Access Rules column, click Create access rules.**

   The Add New Access Rule page opens.
3. **Select the `members` subdirectory.**
4. **Select the Role radio button and from the drop-down list, select `EliteMember`.**

5. **Select the Allow permission.**

   Figure 1-21 shows the resulting settings.

Figure 1-21: Adding an access rule for a role.

6. **Click OK.**

Giving the role access to the `members` folder was easy, but we're not finished yet. You can't be sure that members of the `EliteMember` group actually have access until you analyze the hierarchy of access rules. Figure 1-22 shows that username `JohnOliver` is allowed in because his `Allow` is above everything else. However, the `EliteMember` role's `Allow` is trumped by the `Deny all` above it.

Permission	Users and Roles	Delete
Allow	JohnOliver	Delete
Deny	[all]	Delete
Allow	EliteMember	Delete
Allow	[anonymous]	Delete
Allow	[all]	Delete

Figure 1-22: Incorrect order of access rules.

A couple of fixes are required at this point: `JohnOliver` is now a member of `EliteMember`, so his special permission is unnecessary; the `EliteMember` role needs to move above the `Deny all`. To do this fix, follow these steps:

1. **Navigate to the Security tab by clicking Website⇨ASP.NET Configuration and selecting the Security tab.**
2. **Click Manage access rules.**

3. **Select the `members` subdirectory.**
4. **Click Delete next to the `JohnOliver` username and confirm the deletion.**
5. **Click the `EliteMember` role and then click Move Up.**

   The `EliteMember` role is now at the top of the rules list, as shown in Figure 1-23.

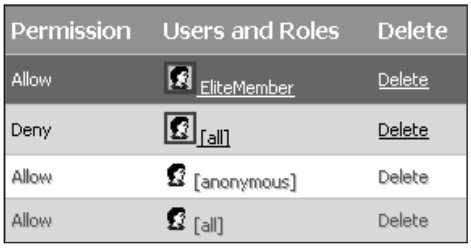

Figure 1-23: Correct order of access rules.

6. **Click Done.**

You can confirm the access rules by logging in as `JillAnon` and navigating to `reserved.aspx` in the `members` folder. You can see that `reserved.aspx` is no longer reserved just for `JohnOliver`.

Without writing a single line of code, we've implemented a very functional membership system. The Web Site Administration tool manages the database and shielded us from the syntax and quirks of `web.config` files. The ASP.NET login controls ship with default settings that work with little or no configuration.

Peering into the Application Programming Interface (API)

It won't come as a surprise to know that there are APIs for everything we've done so far in this chapter. Microsoft's tools and controls offer a developer-friendly front-end to the extensive `Membership`, `MembershipUser`, and `Roles` classes found in the `System.Web.Security` namespace. The upshot is that you can work with the classes in code. This section shows how to use some of the capabilities in your own programs.

Using the Membership and MembershipUser classes

The `Membership` and `MembershipUser` classes offer functions for creating, deleting, updating, and validating a user. You can use these classes to search

for a user, create a list of users, and get the number of logged in users. Some of the examples in this section assume that you're using the membership database with the usernames that we created in the "Creating users" section near the beginning of the chapter.

Adding members programmatically

In the Resto site that we built in the "Creating the Web site" section, the user information includes the username, password, e-mail address, password challenge, and the answer to the challenge. The following VB code re-creates that as it adds a user programmatically. The last parameter of the `CreateUser()` function is a status report.

```
Dim mbrCurrentMember As System.Web.Security.MembershipUser
Dim status As System.Web.Security.MembershipCreateStatus
Try
    mbrCurrentMember =
    System.Web.Security.Membership.CreateUser _
              ("JackieReeve", _
              "ReeveJack!", _
              "rj@nowhere.com", _
              "Your favourite band", _
              "Beatles", _
               True, _
               status)
Catch exc As Exception
    Label2.Text = "Problem: " & exc.Message
End Try
Label2.Text = status.ToString()
```

The first time you run the preceding code, it adds the member and reports `Success`. Run it again without deleting the member. The return code is `DuplicateUserName`, indicating a failure because the username exists.

Deleting members programmatically

Deleting a user account requires less effort than adding one. The `DeleteUser()` function takes the username and a `Boolean` to indicate whether to wipe out all the data related to the user. The method returns `True` if the user was deleted. Here's a VB example:

```
Dim blnRetValue As Boolean
Try
    blnRetValue = System.Web.Security.Membership.DeleteUser _
                      ("JackieReeve", _
                       True)
Catch exc As Exception
```

```
    Label2.Text = "Problem: " & exc.Message
End Try
Label2.Text = "User deleted?: " & blnRetValue.ToString()
```

Updating members programmatically

To change the stored values for a user, call the Membership's `UpdateUser()` method. It takes a `MembershipUser` object as its value. You can use the `GetUser()` function to return a `MembershipUser` object, change the values, and push the data back to the database. Here's a VB example:

```
Dim mbrCurrentMember As System.Web.Security.MembershipUser
mbrCurrentMember = Membership.GetUser("JackieReeve")
If Not IsNothing(mbrCurrentMember) Then
    mbrCurrentMember.E-mail = "jackie@nowhere.com"
    Try
        Membership.UpdateUser(mbrCurrentMember)
        Label2.Text = "User updated: " & _
            mbrCurrentMember.UserName & ", " & _
            mbrCurrentMember.E-mail
    Catch exc As Exception
        Label2.Text = "Problem: " & exc.Message
    End Try
Else
    Label2.Text = "Problem: Not found"
End If
```

Displaying all members programmatically

You can fetch a list of all members — and all their properties — by using the Membership's `GetAllUsers()` function. The following Visual Basic code dumps all the data into a `DataGrid` control that auto-generates the columns:

```
DataGrid1.DataSource = Membership.GetAllUsers()
DataGrid1.DataBind()
```

Using the Roles class

The `Roles` class lets you manage *roles* (that is, groups of users such as those who are all doing the same type of work or have identical privileges). You can use the functions to add and remove users from roles, find users in a given role, determine whether a user is in a role, and create lists of roles. All the capabilities you find in the graphical Web Site Administration tool are available to your code. (This section assumes that you're using the membership database with the usernames from the "Creating users" section earlier in this chapter.)

Adding a role programmatically

You can pass the name of a role to the `CreateRole()` method to create a role. The code below does just that but calls the `RoleExists()` function to check whether the role is already in use. Here's a VB example:

```
Dim strRoleName As String
strRoleName = "Poobahs"
If Not Roles.RoleExists(strRoleName) Then
    Try
        Roles.CreateRole(strRoleName)
        Label2.Text = "Added Role"
    Catch exc As Exception
        Label2.Text = "Problem: " & exc.Message
    End Try
Else
    Label2.Text = "Role exists. Not added again"
End If
```

Deleting a role programmatically

If you know its name, you can delete a role by passing the name to the `DeleteRole()` function. The following code in Visual Basic does a quick check to make sure the role exists before it tries to remove it.

```
Dim strRoleName As String
strRoleName = "Poobahs"
If Roles.RoleExists(strRoleName) Then
    Try
        Roles.DeleteRole(strRoleName)
        Label2.Text = "Deleted Role"
    Catch exc As Exception
        Label2.Text = "Problem: " & exc.Message
    End Try
Else
    Label2.Text = "Role doesn't exist."
End If
```

Adding users to a role programmatically

The `RemoveUsersFromRole()` method handles the task of adding members to a role. The method takes an array of usernames and the name of the role. It's a good idea to check for the existence of the role by using the `RoleExists()` function. Here's a VB example:

```
Dim arrUsers() As String = {"JackieReeve", "JohnOliver"}
Dim strRoleName As String
strRoleName = "Poobahs"
```

```
If Roles.RoleExists(strRoleName) Then
    Try
        Roles.RemoveUsersFromRole(arrUsers, strRoleName)
        Label2.Text = "User(s) Removed."
    Catch exc As Exception
        Label2.Text = "Problem: " & exc.Message
    End Try
Else
    Label2.Text = "Role " & strRoleName & " doesn't exist."
End If
```

Listing all roles

Fetching the list of roles is as simple as calling the `GetAllRoles()` function. It returns a string array that can serve as the data for many ASP.NET data controls. The following VB example gets the list of roles and then passes the name of the first role (via the zero index of the array) to the `GetUsersIn Role()` function. That function then returns the names of all users assigned to the role it specifies.

```
Dim arrRoles() As String
Dim arrMembers() As String
arrRoles = Roles.GetAllRoles()
DataGrid1.DataSource = arrRoles
DataGrid1.DataBind()
arrMembers = Roles.GetUsersInRole(arrRoles(0))
DataGrid2.DataSource = arrMembers
DataGrid2.DataBind()
```