# ASP.NET login controls

Once you've created restricted access to some or all of the pages of your web application, you need to allow users with the proper permissions:

1. To log in and access the restricted pages
2. To allow users to log out
3. To create an account by themselves
4. To recover a forgotten password, or to change a password.

With ASP.NET 2.0, you can use the controls in the *Login* group of the *Toolbox* to automatically handle these tasks.

## Login control

When you create a login page, you should name it *Login.aspx*. That is because ASP.NET looks for a page with this name when it attempts to authenticate a user. After that you can add all of the login functionality just by adding a *Login* control to the page.

When the user clicks the *Log In* button within the *Login* control, the code for the control tries to authenticate the user. It does that by checking to see whether the username and password are in the membership data store. Then, if the user is authenticated, the code checks the role provider to see the user has the proper authorization for the requested page. If so, this code continues by redirecting the browser to that page.

**Common attributes of the *Login* control**

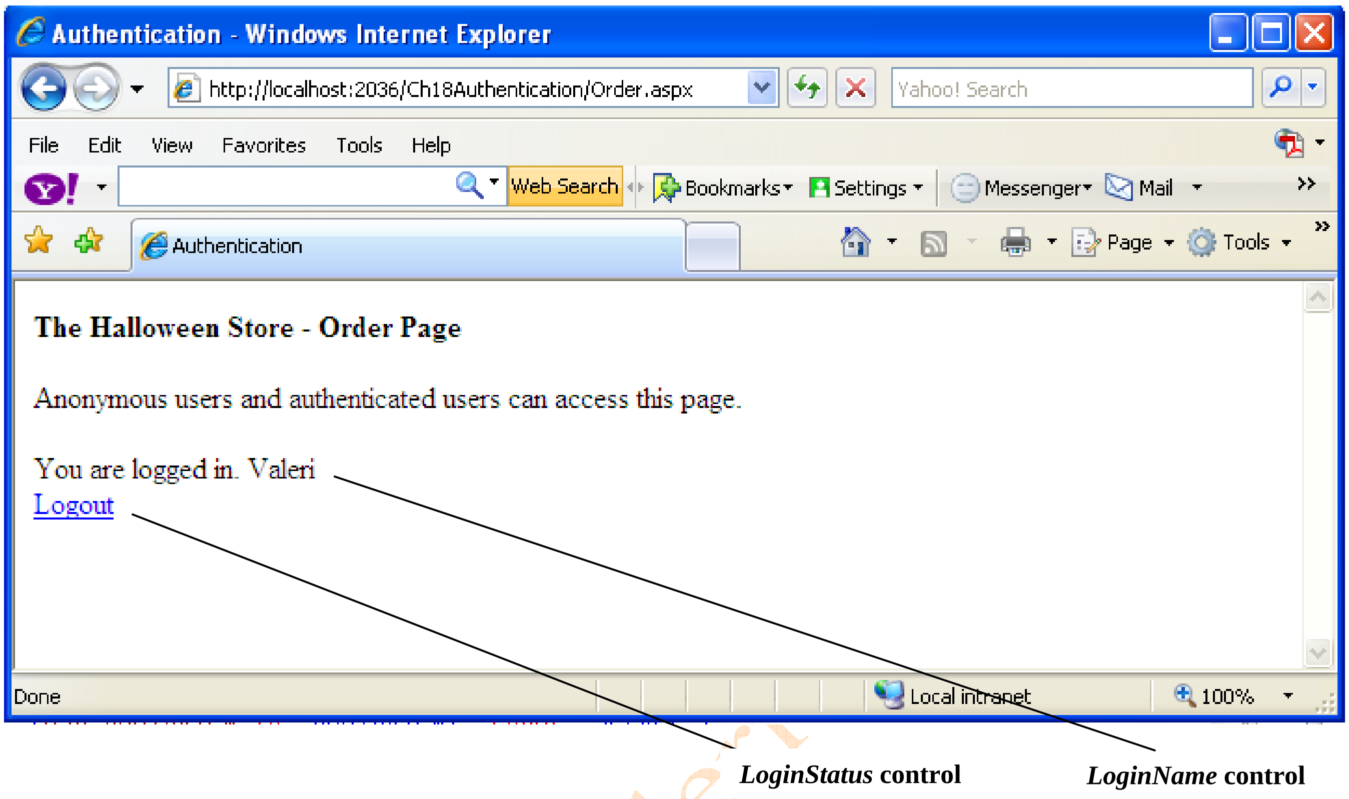
| Attribute | Description |
|---|---|
| *DisplayRememberMe* | Determines whether the *DisplayRememberMe* check box is displayed. By default, this is set to *True*. |
| *RememberMeText* | The text for the label of the *RememberMe* text box |
| *RememberMeSet* | Determines whether a persistent cookie is sent to the user's computer, which is based on whether the *RememberMe* check box is selected |
| *FailureText* | The text that's displayed when a login attempt fails |

## *LoginStatus* and *LoginName* controls

In the page, presented below, you can see that the *LoginName* control provides the name of a logged in user. In contrast, the *LoginStatus* control provides a *Login* link if a user hasn't logged in yet and a *Logout* link if the user has logged in.

When the user clicks on the *Login* link of a *LoginStatus* control, the user will be redirected to the *Login* page and required to enter a username and password. Then, after the user has been authenticated, the user will be redirected to the original page. Conversely, when the user clicks the *Logout* link, the user will be redirected to the login page *(Login.aspx)*

**The *LoginName* and *LoginStatus* controls displayed in a browser**



The Halloween Store - Order Page

Anonymous users and authenticated users can access this page.

You are logged in. Valeri
Logout

**Common attribute of the *LoginName* control**

| Attribute | Description |
|---|---|
| *FormatString* | The text that's displayed with the username. This string uses *"{0}"* to identify the *UserName* parameter, and you can add text before or after this parameter. |

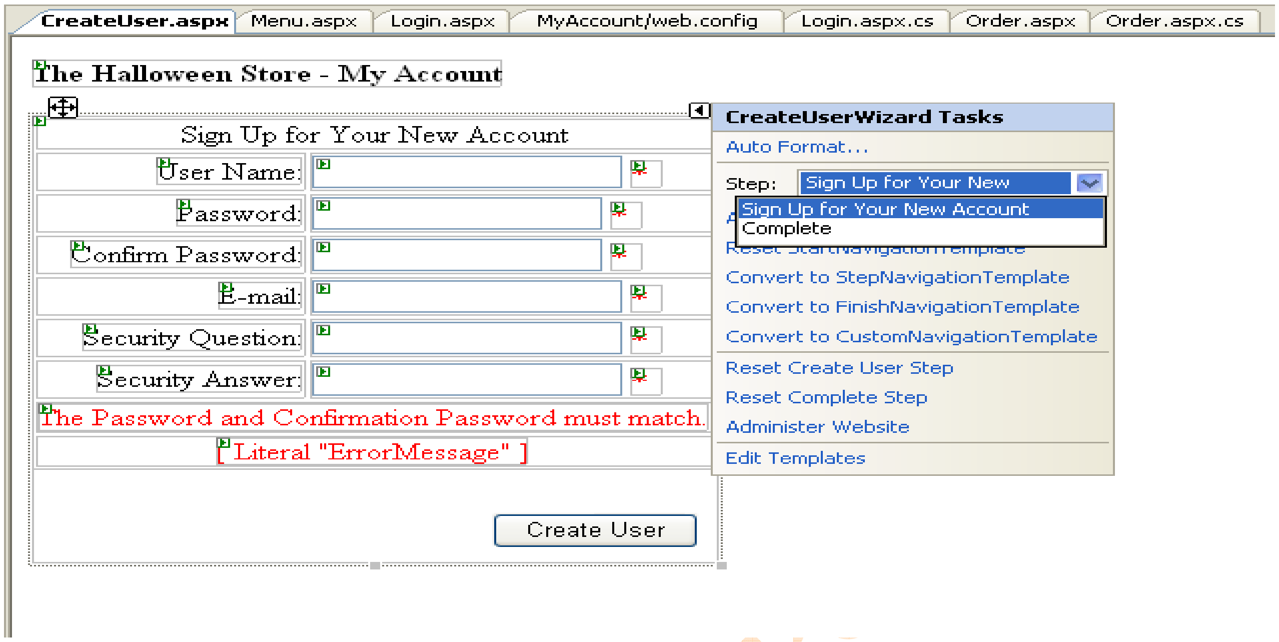
**Common attribute of the**

| Attribute | Description |
|---|---|
| *LoginText* | The text that's displayed for the login link |
| *LogoutText* | The text that's displayed for the logout link |

## *CreateUserWizard* control

If you have a few users for your application, you can use the Web Site Administrative Tool to create and manage users. Then, you can use the *Login, LoginStatus*, and *LoginName* controls to allow users to log in and out. Often, though, you'll want to allow users to create user accounts for themselves. To do that, you can use the *CreateUserWizard* control as shown below.

**The *CreateUserWizard* control with the smart tag menu**



CreateUser.aspx | Menu.aspx | Login.aspx | MyAccount/web.config | Login.aspx.cs | Order.aspx | Order.aspx.cs

The Halloween Store - My Account

Sign Up for Your New Account

User Name:

Password:

Confirm Password:

E-mail:

Security Question:

Security Answer:

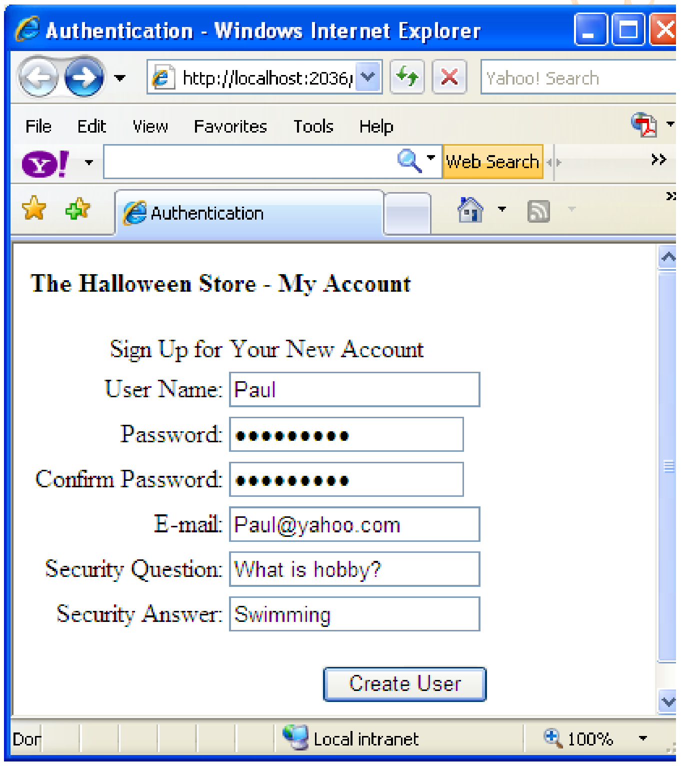
The Password and Confirmation Password must match.

[Literal "ErrorMessage" ]

Create User

CreateUserWizard Tasks
Auto Format...
Step: Sign Up for Your New
Sign Up for Your New Account
Complete
Reset StartNavigationTemplate
Convert to StepNavigationTemplate
Convert to FinishNavigationTemplate
Convert to CustomNavigationTemplate
Reset Create User Step
Reset Complete Step
Administer Website
Edit Templates

By default, the *CreateUserWizard* control uses two steps to create a new user: the *Create User* step and the *Complete* step. Usually, these steps work like you want them to. In that case, you can simply set the *ContinueDestinationPageUrl* property of the control to indicate what page the user is directed to when the *Continue* button in the *Complete* step is clicked.



Authentication - Windows Internet Explorer

The Halloween Store - My Account

Sign Up for Your New Account

User Name: Paul

Password: •••••••••

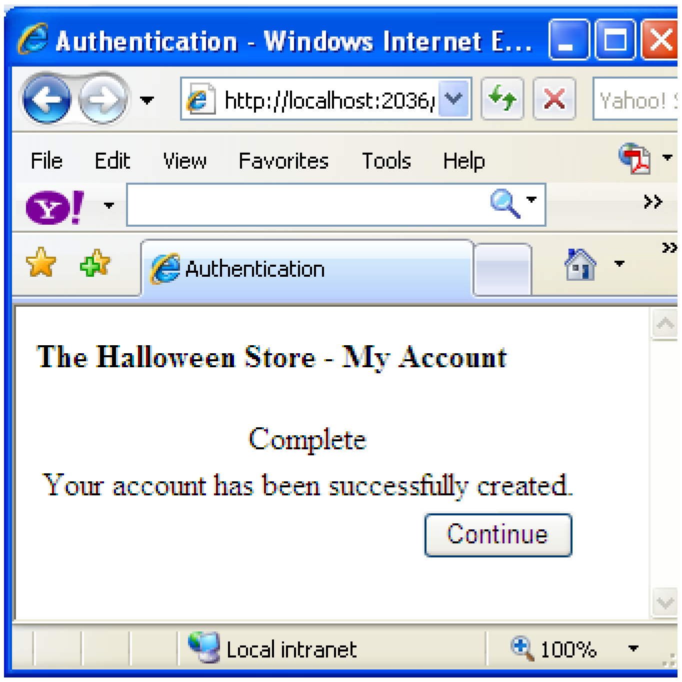
Confirm Password: •••••••••

E-mail: Paul@yahoo.com

Security Question: What is hobby?

Security Answer: Swimming

Create User



Authentication - Windows Internet E...

The Halloween Store - My Account

Complete

Your account has been successfully created.

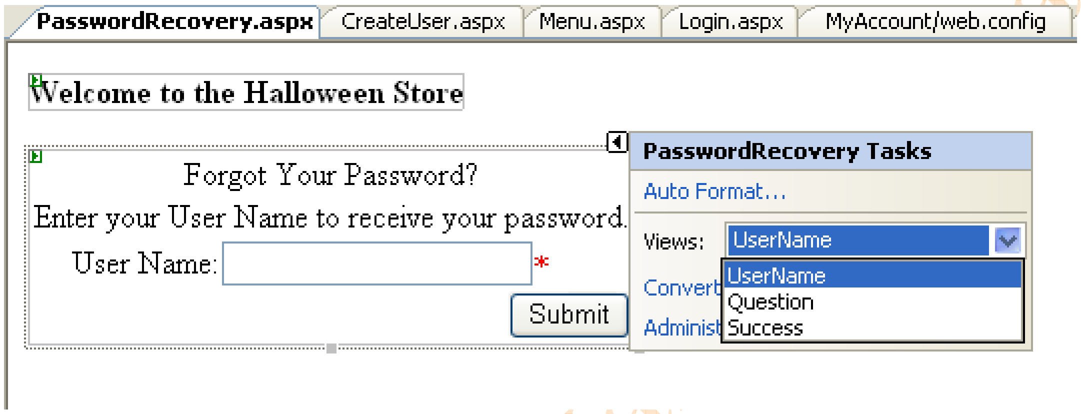
Continue

When you use the pages that rezult from the *CreateUserWizard* control, you should realize that the membership provider is being used to write the data to the appropriate data source. This work the same as it does for the Web Site Administrative Tool.

## The *PasswordRecovery* control

The *PasswordRecovery* control makes the process of recovering forgotten password. When you use the *PasswordRecovery* control, the password is sent to the user via e-mail. As a result, you need to make sure that your system is set up so it can send e-mail before you test this control. By default, an application will try to send e-mail to an SMTP server set to localhost on port 25.

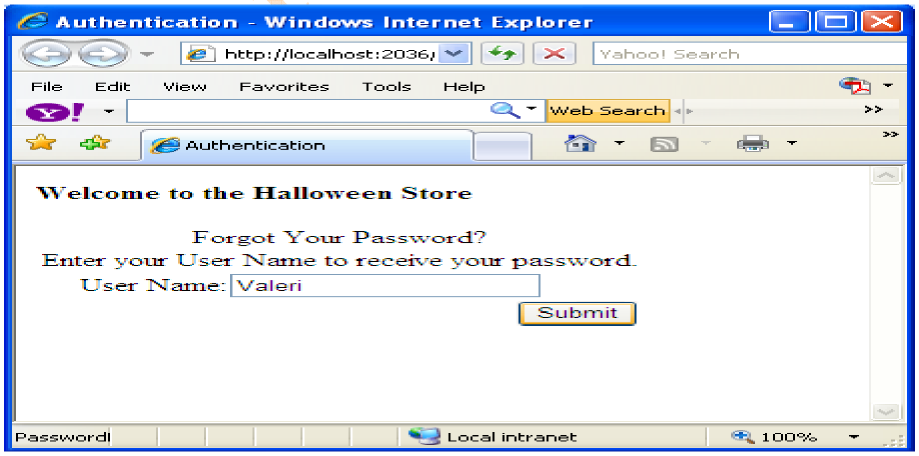
### The *PasswordRecovery* controls in the Web Forms Designer



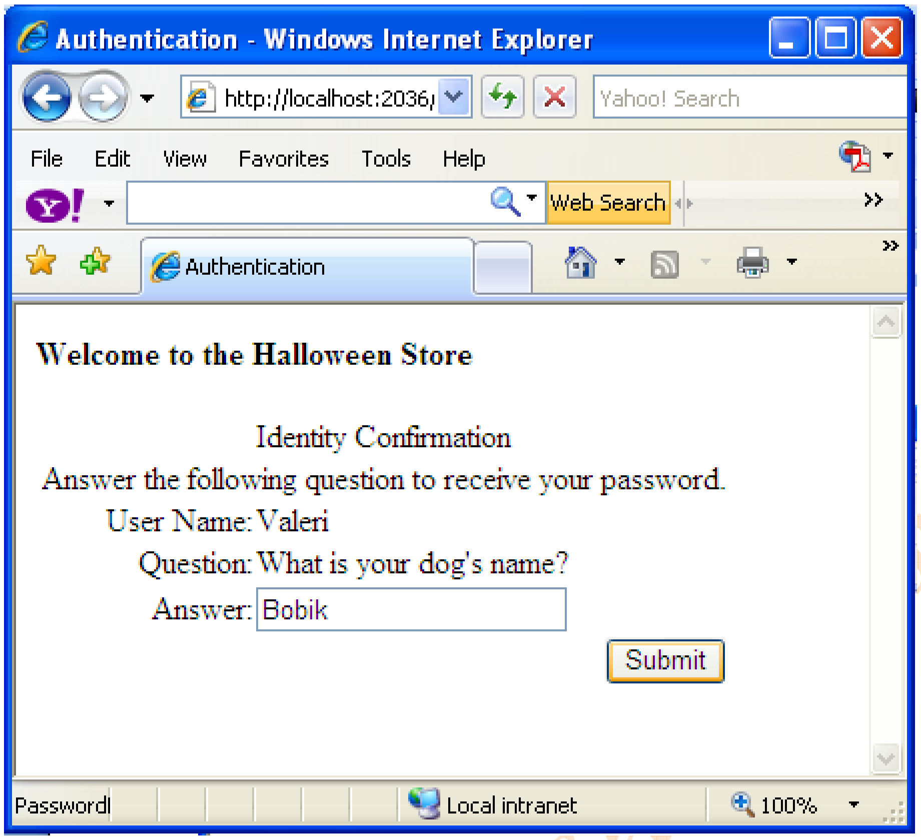
The most important element of the *PasswordRecovery* control is the *MailDefinition* element:

```
<asp:PasswordRecovery ID="PasswordRecovery1" runat="server">
   <MailDefinition From="vpougatchev@utech.edu.com">
   </MailDefinition>
</asp:PasswordRecovery>
```

The *PasswordRecovery* control uses three views. In the Web Forms Designer, you can switch between these views by using the smart tag menu, and you can edit the properties for any of these views as necessary. The first view asks the user to enter a username:


Authentication - Windows Internet Explorer
http://localhost:2036/
Yahoo! Search
File Edit View Favorites Tools Help
Web Search
Authentication
Welcome to the Halloween Store
Forgot Your Password?
Enter your User Name to receive your password.
User Name: Valeri
Submit
Passwordl
Local intranet
100%

The second view requires the user to answer the security question. If the answer to this question is correct, the password is emailed to the address that's associated with the username:



Welcome to the Halloween Store

Identity Confirmation

Answer the following question to receive your password.

User Name: Valeri

Question: What is your dog's name?

Answer: Bobik

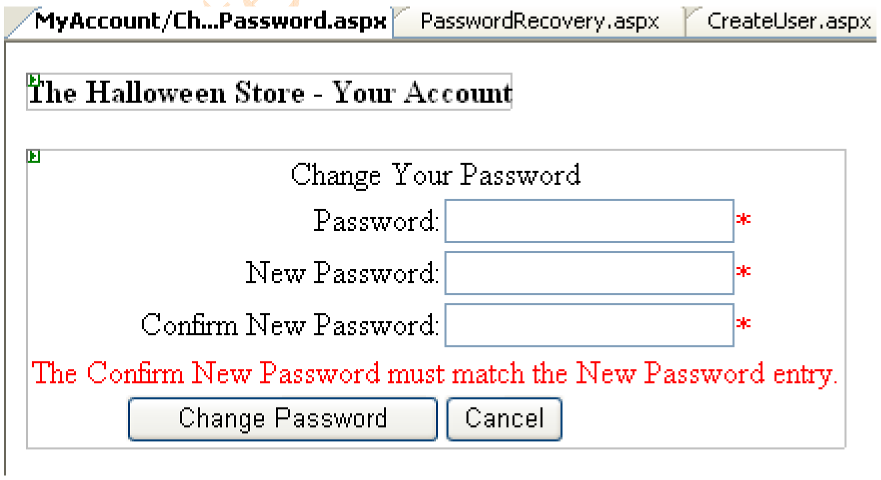
Submit

The third view displays a message that indicates that the password recovery was successful and that the password has been sent to the user via e-mail.

## The *ChangePassword* control

The *ChangePassword* control uses two views. The first view lets the user enter the current password and the new password twice.

**The first view of the *ChangePassword* control in the Web Forms Designer**



The Halloween Store - Your Account

Change Your Password

Password: *

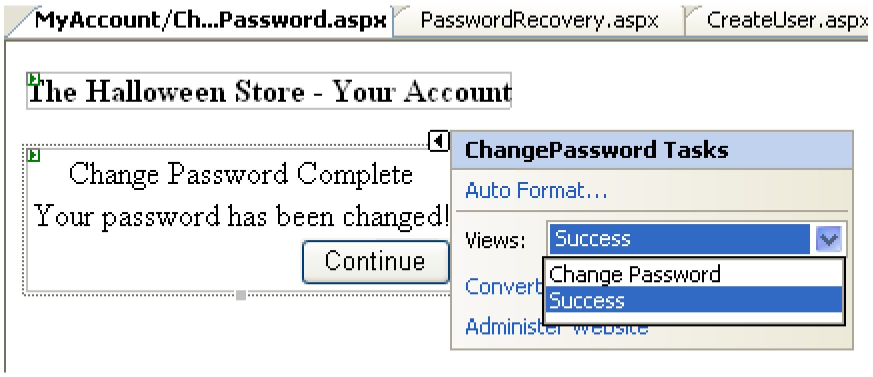
New Password: *

Confirm New Password: *

The Confirm New Password must match the New Password entry.

Change Password Cancel

**The second view of the *ChangePassword* control**



**The aspx code for the *ChangePassword* control**

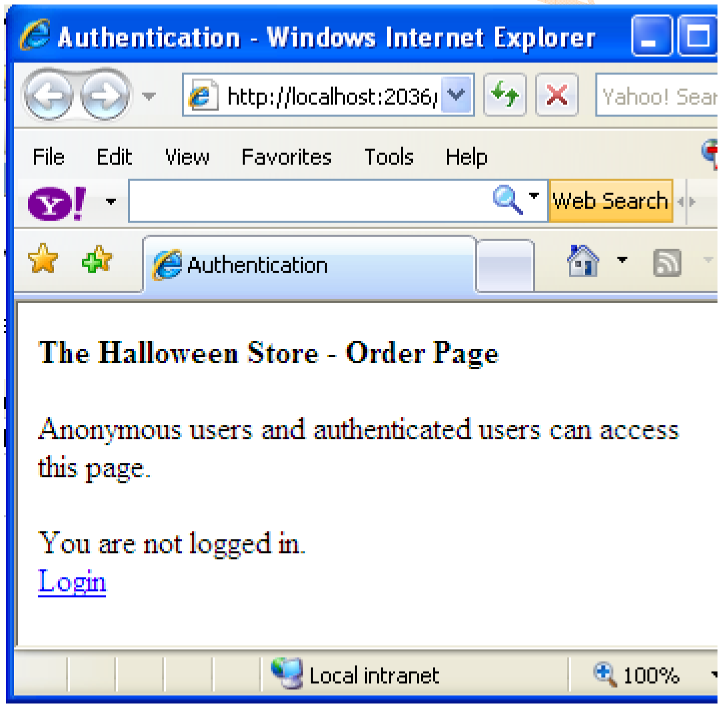
```
<asp:ChangePassword ID="ChangePassword1" runat="server"
   CancelDestinationPageUrl="MyAccount.aspx"
   ContinueDestinationPageUrl="MyAccount.aspx">
</asp:ChangePassword>
```

The *CancelDestinationPageUrl* and *ContinueDestinationPageUrl* attributes provide the URLs that are navigated to when the *Cancel* or *Continue* buttons are clicked.

## The *LoginView* control

The *LoginView* control uses two views. They contain the controls that are displayed to users who aren't logged in (*anonymous users*) or is logged in (an *authenticated user*).

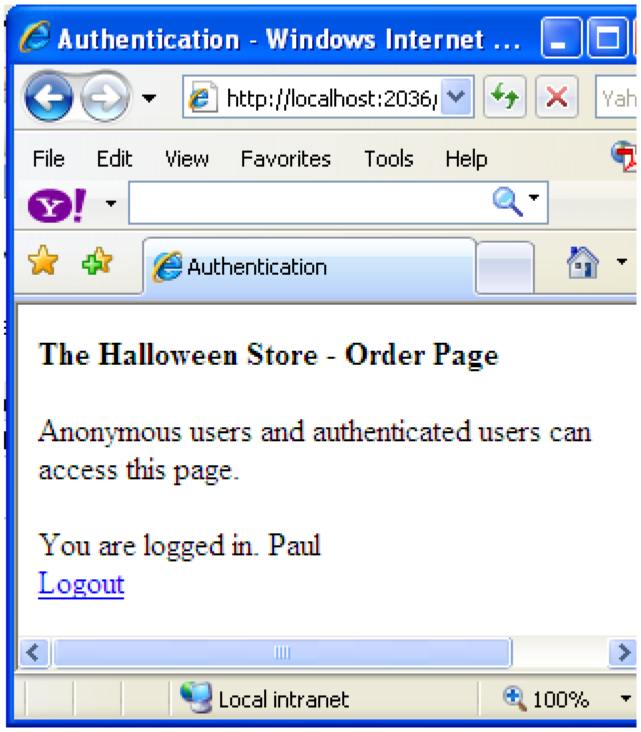
**The *LoginView* control displayed in a browser**


Authentication - Windows Internet Explorer
http://localhost:2036/
File Edit View Favorites Tools Help
Web Search
Authentication
Local intranet
100%

The Halloween Store - Order Page

Anonymous users and authenticated users can access this page.

You are not logged in.
Login

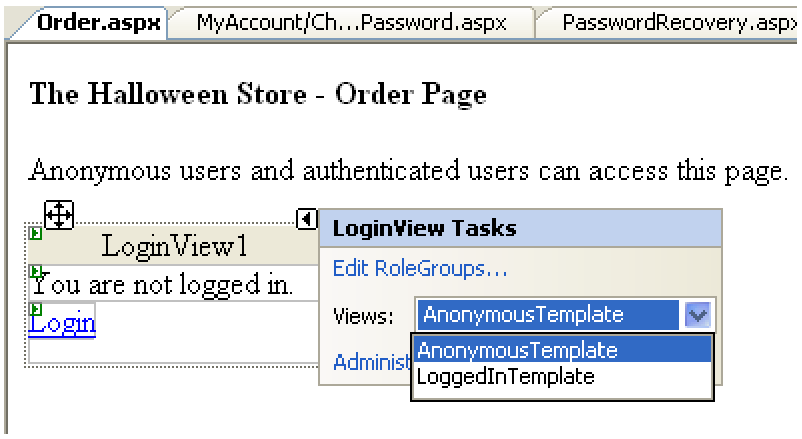

Authentication - Windows Internet ...
http://localhost:2036/
File Edit View Favorites Tools Help
Authentication
Local intranet
100%

The Halloween Store - Order Page

Anonymous users and authenticated users can access this page.

You are logged in. Paul
Logout

**The *LoginView* control in the Web Forms Designer**



The Halloween Store - Order Page

Anonymous users and authenticated users can access this page.

You are not logged in.

Login

**The aspx code for the *LoginView* control**

```
<asp:LoginView ID="LoginView1" runat="server">
  <LoggedInTemplate>
    <asp:LoginName ID="LoginName1" runat="server" BorderWidth="0px"
      FormatString="You are logged in. {0}" Width="363px" /><br />
    <asp:LoginStatus ID="LoginStatus2" runat="server" />
  </LoggedInTemplate>
  <AnonymousTemplate>
    <asp:Label ID="Label1" runat="server"
      Text="You are not logged in." Width="148px"></asp:Label><br />
    <asp:LoginStatus ID="LoginStatus1" runat="server" />
  </AnonymousTemplate>
</asp:LoginView>
```

## The *Authentication* application

### The directory structure

The figure below shows the directory structure for the *Authentication* application.

First, it shows the *Maintenance* directory, which contains the *Maintenance* page. Second, it shows the *MyAccount* directory, which contains the *MyAccount* page.

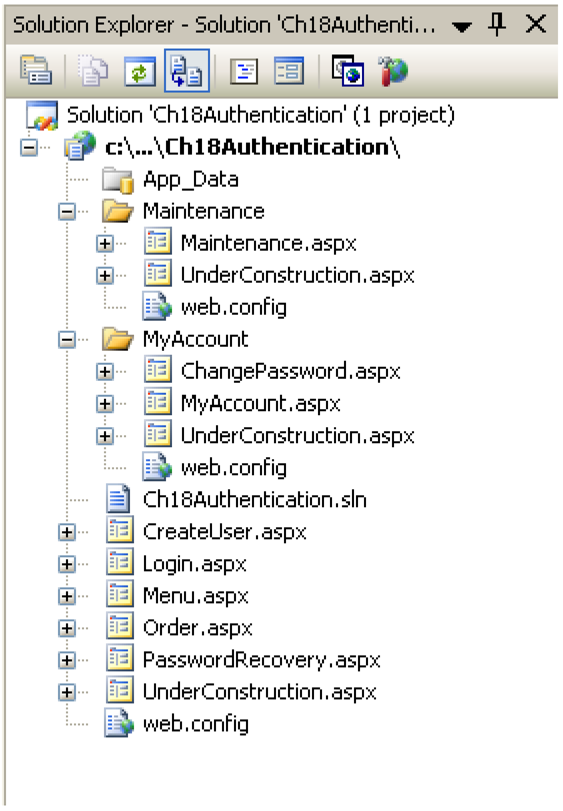
In addition, there are three web.config files for this application: one for the root directory, one for the *Maintenance* directory, and one for the *MyAccount* directory.

### The access rules

The figure below also shows the one access rule for the *MyAccount* directory. This rule denies access to anonymous users. As a result, only users who are authenticated can access the *MyAccount* directory.

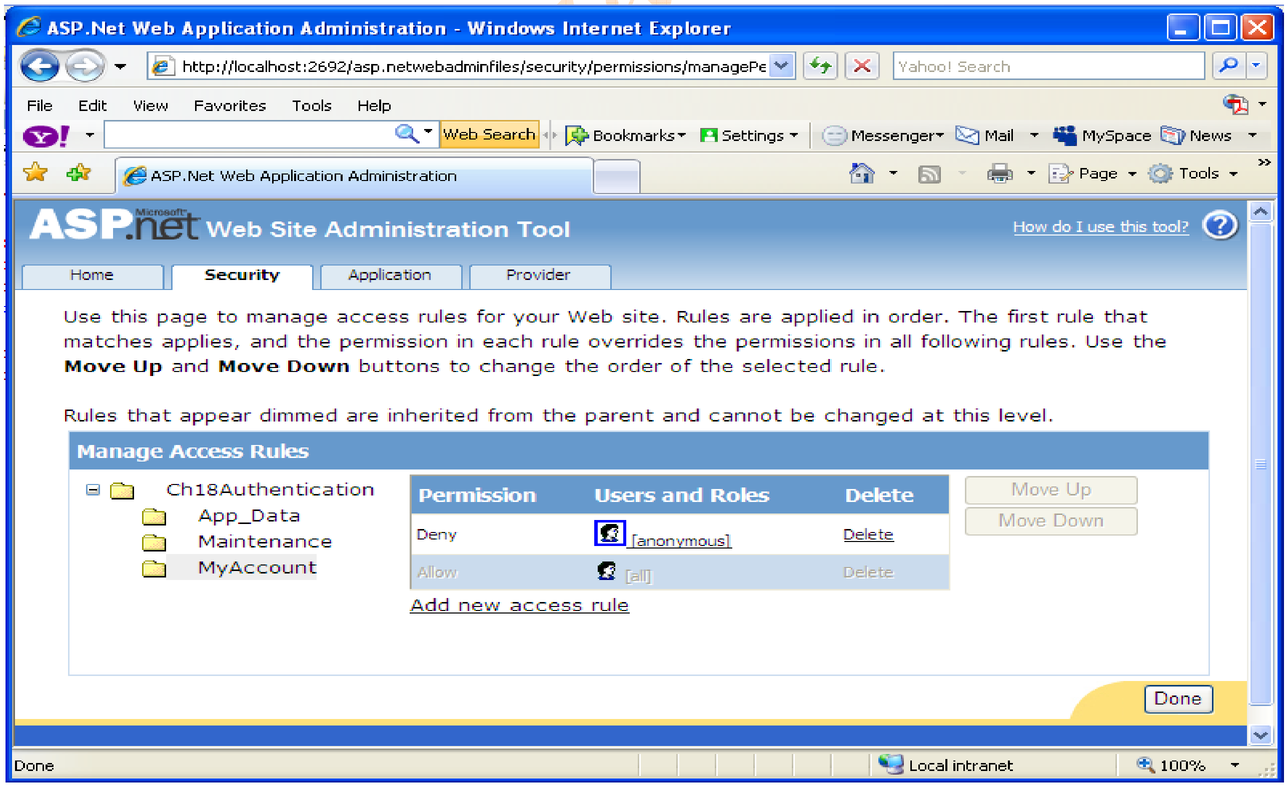
The *Maintenance* directory, on the other hand, contains two access rules. The first rule denies access to all users. Then, the second rule allows access to authenticated users who are associated with the *Admin* role.

## The directory structure for the Authentication application



```
Solution 'Ch18Authentication' (1 project)
c:\...\Ch18Authentication\
    App_Data
    Maintenance
        Maintenance.aspx
        UnderConstruction.aspx
        web.config
    MyAccount
        ChangePassword.aspx
        MyAccount.aspx
        UnderConstruction.aspx
        web.config
    Ch18Authentication.sln
    CreateUser.aspx
    Login.aspx
    Menu.aspx
    Order.aspx
    PasswordRecovery.aspx
    UnderConstruction.aspx
    web.config
```

## The access rules for the *My Account* directory


ASP.Net Web Application Administration - Windows Internet Explorer
http://localhost:2692/asp.netwebadminfiles/security/permissions/managePe
Yahoo! Search
File Edit View Favorites Tools Help
Web Search
Bookmarks
Settings
Messenger
Mail
MySpace
News
ASP.Net Web Application Administration
Page
Tools
ASP.net Web Site Administration Tool
How do I use this tool?
Home
Security
Application
Provider
Done
Local intranet
100%

Use this page to manage access rules for your Web site. Rules are applied in order. The first rule that matches applies, and the permission in each rule overrides the permissions in all following rules. Use the **Move Up** and **Move Down** buttons to change the order of the selected rule.

Rules that appear dimmed are inherited from the parent and cannot be changed at this level.

**Manage Access Rules**

Ch18Authentication
- App_Data
- Maintenance
- MyAccount

| Permission | Users and Roles | Delete |
|---|---|---|
| Deny | [anonymous] | Delete |
| Allow | [all] | Delete |

Add new access rule

Move Up
Move Down

## The *web.config* files

To enable forms-based authentication, you can use the *Mode* attribute of the *Authentication* element. This attribute is set to “Windows” by default to use Windows authentication, but you can set it to “Forms” to enable forms-based authentication.

### The *web.config* files for the Authentication application

### For the root directory

```
<?xml version="1.0"?>
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
 <connectionStrings>
  <remove name="LocalSqlServer" />
  <add name="LocalSqlServer"
    connectionString="Data Source=VPOUGATCHEV;
                  Initial Catalog=HalloweenASPNETDB;
                  Integrated Security=True;
                  MultipleActiveResultSets=False;
                  Packet Size=4096;
                  Application Name=&quot;
                  Microsoft SQL Server Management Studio Express&quot;"
    providerName="System.Data.SqlClient" />
 </connectionStrings>
 <system.web>
                  <authentication mode="Forms"/>
                  <roleManager enabled="true"/>
                  <compilation debug="true"/>
 </system.web>

</configuration>
```

### For the *MyAccount* directory

```
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
  <system.web>
    <authorization>
      <deny users="?" />
    </authorization>
  </system.web>
</configuration>
```

**For the *Maintenance* directory**

```
<?xml version="1.0" encoding="utf-8"?>
<configuration xmlns="http://schemas.microsoft.com/.NetConfiguration/v2.0">
  <system.web>
    <authorization>
      <allow roles="admin" />
      <deny users="*" />
    </authorization>
  </system.web>
</configuration>
```

***Wildcard* specifications in the users attribute**

| Wildcard | Description |
|---|---|
| * | All users, whether or not they have been autheticated |
| ? | All unauthenticated users |

## Using a code to work with authentication

ASP.NET provides three new classes that make to write a code easy:

1. *Membership*
2. *MembershipUser*
3. *Roles*

All these classes are stored in the *System.Web.security* namespace.

If you need to work with user data, you can use the *Membership* class as shown below:

| Method | Description |
|---|---|
| *ValidateUser(username, password)* | Returns a *Boolean* value that indicates whether the user is authenticated and authorized |
| *CreateUser(username, password)* | Creates a user with the specified username and password |
| *CreateUser(username, password, email)* | Creates a user with the specified username, password, and e-mail address |
| *GetUser()* | Returns a *MembershipUser* object that contains data about the currently logged-on user. Updates the last activity date-time stamp. |
| *GetUser(username)* | Returns a *MembershipUser* object that contains data about the user with the specified username. Updates the last activity date-time stamp. |

|  |  |
|---|---|
| *GetUserNameByEmail(email)* | Returns a string that contains the username that corresponds with the specified e-mail address. |
| *GetAllUsers()* | Returns a *MembershipUserCollection* object that contains *MembershipUser* object |
| *DeleteUser(username)* | Deletes the user with the specified username |
| *UpdateUser(memvershipUser)* | Updates the user with the data that's stored in the specified *MembershipUser* object |

**A statement that checks if a user is valid**

```
bool validUser;
validUser = Membership.ValidateUser(txtUserName.Text, txtPassword.Text)
```

**A statement that creates a usert**

```
try
{
        Membership.CreateUser(txtUserName.Text, txtPassword.Text);
}
catch(MembershipCreateUserException ex)
{
        lblStatus.Text = ex.Message;
}
```

**A statement that gets the current user and updates the last activity timestamp**

```
MembershipUser user = Membership.GetUser();
```

**A statement that deletes a user**

```
Membership.DeleteUser(txtUsername.Text);
```

## The *MembershipUser* class

You typically use a method of the *Membership* class to retrieve a *MembershipUser* object from the data store that's used for authentication. Then, you can use the properties and methods of the *MembershipUser* object to get or modify the data that's stored in this object.

For example, you might want to use the *Email* property to get the e-mail address from the *MembershipUser* object. Or, you might want to set the *IsApproved* property to *False* if the user is unsuccessful at logging in a given number of times. Then, the user would not be allowed to log in until the password question was answered correctly, at which time the *IsApproved* property could be reset to True.

You might also want to use the *ChangePassword* method to change the password for the object. When you use this and other methods from the *MembershipUser* objcet, it updates the data in the

*MembershipUser* object and it updates the data in the membership data store. As a result, you don't need to use the *UpdateUser* method of the *Membership* class to update the data in the data store.

**Common properties of the *MembershipUser* class**

| **Property** | **Description** |
|---|---|
| *Uesrname* | The username for the membership user |
| *Email* | The email address for the membership user |
| *PasswordQuestion* | The password question for the membership user |
| *CreationDate* | The date and time when the user was added to the membership data store |
| *LastLoginDate* | The date and time when the membership user was last authenticated |
| *LastActivityDate* | The date and time when the membership user was last authenticated or accessed the application |
| *IsApproved* | A Boolean value that indicates whether the membership user can be authenticated |

**Common methods of the MembershipUser class**

| **Method** | **Description** |
|---|---|
| *GetPassword()* | Gets the password for the membership user from the membership data store |
| *ChangePassword(oldPassword, newPassword)* | Updates the password for the membership user in the membership data store |
| *ResetPassword()* | Resets a user's password to a new, automatically generated password |
| *ChangePasswordQuestionAndAnswer (password, question, answer)* | Updates the password question and answer for the membership user in the membership datastore |

**Code that changes a user's password**

```
 MembershipUser user = Membership.GetUser();
try
{
        user.ChangePassword(txtOldPassword.Text, txtNewPassword.Text);
}
catch(Exception ex)
```

```
{
        lblStatus.Text = "Error changing password: " + ex.Message;
}
```

## Using the *FormsAuthentication* class

If you don't want to use the *Login* control, you can create a login form that includes text boxes that allow the user to enter a username and password. Then, tiy can write code that authenticates the user and redredirects the user from the login page to the page that was originally requested. To do that, you can use the *ValidateUser* method of the *Membership* class with the *RedirectFromLoginPage* method of the *FormsAuthentication* class as shown below.

**Common methods of the FormsAuthentication class**

| Method | Description |
|---|---|
| *RedirectFromLoginPage(username, createPersistentCookie)* | Issues an authentication ticket for the user and redirects the browser to the page it was attemting to access when the login page was displayed. If the *createPersistentCookie* argument is *true*, the cookie that contains the authentication ticket is persistent across browser restarts. Otherwise, the cookie is only available for the current session. |
| *SignOut()* | Logs the user off by removing the cookie that contains the authentication ticket |

**A statement that redirects the browser to the originally requested page**

```
if (Membership.ValidateUser(txtUserName.Text, txtPassword.Text))
        FormsAuthentication.RedirectFromLoginPage(txtUserName.Text, false);
else
        lblStatus.Text = "Invalid user! Try again.";
```

**A statement that logs a user off**

```
FormsAuthentication.SignOut();
```

## Using the *Roles* class

The *Roles* class, shown below, works similarly to the *Membership* class. First, all methods of the *Roles* class are static methods. Second, you can use the *Roles* class to read data from and write data to the data store that's specified by the role provider for the application.

Although you can use the *Roles* class to add and delete roles from the roles data store, you probably won't need to do that. However, you might need to provide some custom code that works with roles in another way. To illustrate, let's say you've used the Web Site Administrator Tool to define a role named *Premium*. Then, when uesrs pay a small fee, they can access the *Premium* pages of the web site. In that case, you might need to write code that adds the current user to the *Premium* role after the user has paid for the primium subscription. To do that, you can use the *AddUserToRole* method of the *Roles* class as shown below.

**Common methods of the *Roles* class**

| Methods | Description |
|---|---|
| *CreateRole(rolename)* | Adds a new role to the data store |
| *GetAllRoles()* | Gets a string array containing all the roles for the application |
| *AddUserToRole(username, rolename)* | Adds the specified user to the specified role |
| *GetRolesForUser(username)* | Gets a string array containing the roles that a user is in |
| *GetUserInRoles(rolename)* | Gets a string array containing users in the specified role |
| *IsUserInRole(username, rolename)* | Gets a Boolean value indicating whether a user is in the specified role |
| *RemoveUserFromRole(username, rolename)* | Removes the specified user from the specified role |
| *DeleteRole(rolename)* | Removes a role from the data store |

**Code that creates a role**

```
try
{
        Roles,CreateRole(txtRoleName.Text);
}
catch(Exception ex)
{
        arrorMessage = “Error creating role: “ + ex.Message;
}
```

**Code that gets all roles in the system**

```
string[] roles = Roles.GetAllRoles();
```

**A statement that adds a user to a role**

```
Roles,AddUserToRole(currentUserName, “Premium”);
```

**A statement that removes a user from a role**

*Roles.RemoveUserFromRole(txtUsername.Text, cboRoles.SelectedValue);*