

Insider Extra 14

Building Your Own Self-Registration System

If you need a self-registration system but your site resides on a Microsoft Windows Web server, the self-registration system that's part of Microsoft Office FrontPage 2003 won't be available to you. For historical reasons, the FrontPage Self-Registration component only works on non-Microsoft Web servers.

One alternative, particularly on intranets, is to secure your Web site using Windows logon accounts. If everyone in your organization already has a Windows logon account, there's no need for self-registration. Visitors can simply use their Windows accounts for Web site access as well.

For more information about using system logon accounts to control Web site access, refer to "Controlling Web Site Security Through FrontPage," on page 1146, and "Administering Web Settings," in Appendix P.

Building a Self-Registration System with ASP

If using system logon accounts to control Web access doesn't work for you, the use of ASP pages provides another solution. ASP pages have access to a *Session* object that remembers data pertaining to a given visitor as he or she traverses your site. What one ASP page puts in the *Session* object, another (later) ASP page can retrieve. ASP automatically creates a new *Session* object for each visitor.

One thing a *Session* object can remember quite easily is whether a given visitor is logged in. A *Session* variable might, for example, contain the visitor's e-mail address if the visitor is logged in and be empty if the visitor isn't. Each secure page in a Web site should then contain the following logic:

- If the current visitor is logged in (that is, if the *Session* object contains an e-mail address), allow the visitor to view the current page.
- If the current visitor isn't logged in (that is, if the *Session* object contains an empty e-mail address), record the current page's URL in another *Session* variable, and send the visitor to a login page.

The reason for recording the current page's URL is so that the login process can send the visitor back to that location (assuming, of course, that the login succeeds).

Microsoft Office FrontPage 2003 Inside Out

Here's the necessary code. To use this solution, rename every Web page you want to secure—giving it an .asp file extension—and then place the following code at the start of each page:

```
<%
If Session("loginemail") = "" Then
    Session("goback") = Request.ServerVariables("URL")
    Response.Redirect "seclogin.asp"
End If
%>
```

- As usual, <% and %> tags mark the start and end of a block of ASP code.
- The *If* statement determines whether the *Session* variable named *loginemail* is empty. If so:
 - 1 The code sets the *Session* variable *goback* to the URL of the current page.
 - 2 The code then redirects the Web visitor to another Web page, named *seclogin.asp*.

The best way to add this code to each page is with a server-side include (SSI). An Include Page component, for example, can't add content at the start of a Web page (that is, before the <html> tag).

For more information about SSIs, refer to the Frequently Asked Question sidebar, "What Are Server-Side Includes?" on page 753.

If the visitor succeeds in logging in, the login process saves the visitor's e-mail address in *Session("loginemail")* and then redirects the visitor to the URL in *Session("goback")*.

Figure IE14-1 shows the *seclogin.asp* page. This page offers the following four functions:

- **Login** Searches a database for the specified e-mail address. If a record with that address exists, and if the password in the database matches the one in the HTML form, the code sets *Session("loginemail")* to the given e-mail address. If *Session("goback")* contains a URL, the code jumps to that location.
- **Logout** Sets *Session("loginemail")* to an empty string.
- **New User** Jumps to a Web page where visitors can create new accounts for themselves. Figure IE14-2 illustrates this page.
- **Send Pswd** Sends an e-mail to the specified e-mail address, informing the visitor of the current password for his or her account. Note that visitors can't send someone else's password to themselves.

Building Your Own Self-Registration System

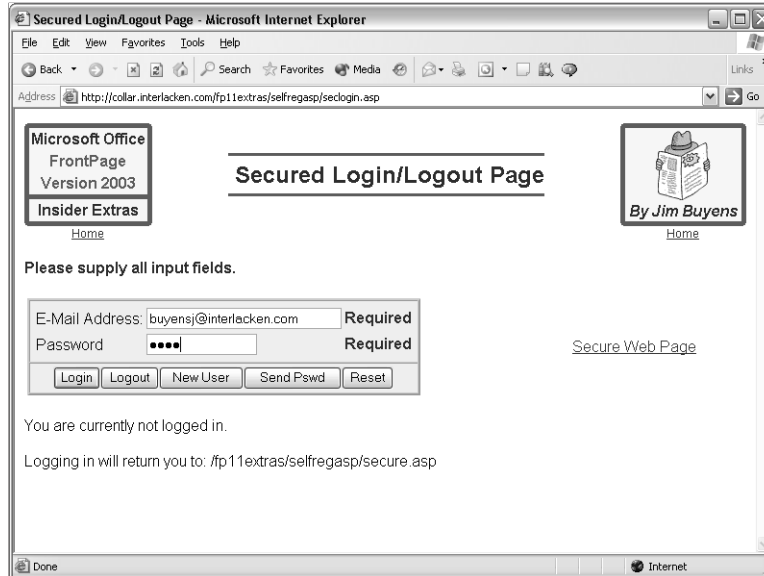


Figure IE14-1. This ASP page verifies the e-mail address and password entered against a database.

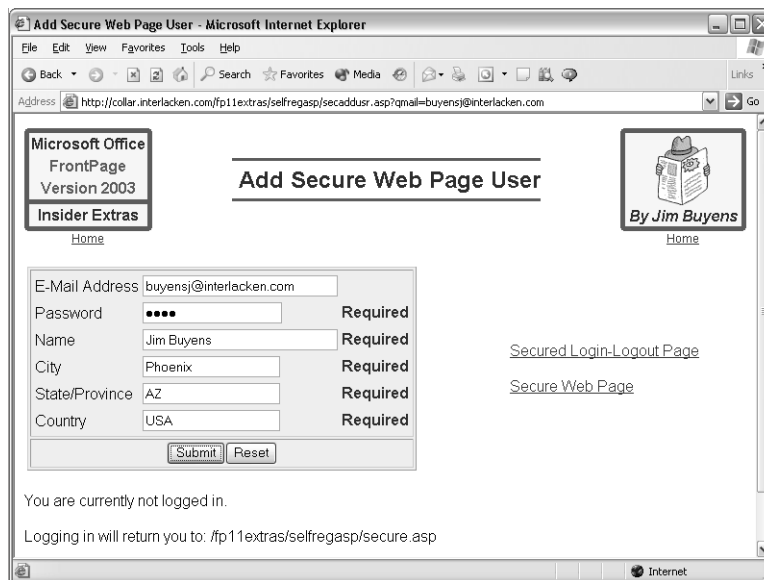


Figure IE14-2. This ASP page creates a new user account that uses the visitor's e-mail address as its key.

The code that implements these functions isn't terribly complex, but it's too long to list and explain in this book. For a closer look, open the files `selfregasp/seclogin.asp` and `selfregasp/`

Microsoft Office FrontPage 2003 Inside Out

secadduser.asp in the Insider Extra sample Web site that you installed from the companion CD. To view the HTML code, click the Code tab at the bottom of the Page view window.

If you want to use this Insider Extra as the starting point for a self-registration system on your own Web site, heed these requirements:

- Your Web server must be capable of running ASP pages. In general, this means that it must be running Microsoft Windows 2000, Windows XP Professional, or Windows Server 2003.
- The visitors.mdb database must reside in an fpdb folder located in your Web site's root folder. You can find a copy of this database in the fpdb folder of the Insider Extra sample Web site.
- The folder that contains your Web site must also be the root of an IIS application. FrontPage makes sure that this is true whenever it sets up a database connection. When you drag the visitor.mdb database to your site and FrontPage asks whether you want to create a database connection for it, answer Yes.
- The Send Pswd feature uses the *CDONTS.NewMail* object to send mail. Despite the fact that this is a standard, Microsoft-provided object, some providers don't support it. If you find yourself in this situation, you'll need to change the code so that it uses whatever mail object the provider does support.

Even if you *can* use the *CDONTS.NewMail* object, you'll need to replace the default From address nobody@deadletter.msn.com with your own. This appears in the seclogin.asp page.

Building a Self-Registration System with ASP.NET

As you might suspect, ASP.NET addresses a number of long-standing problems and feature omissions in the ASP programming model. One of these problems was the lack of any built-in support (other than Windows login accounts) for restricting access to sensitive Web pages. ASP.NET addresses this need with a feature called *forms-based authentication*.

Forms-based authentication operates at the IIS application level. Like a FrontPage-based Web site, an IIS application includes everything within a designated, browsable folder tree except subfolders that are IIS applications in their own right. When you use FrontPage to work with ASP.NET applications, you should generally arrange for the Web site and the application to begin at the same folder.

Building Your Own Self-Registration System

Using Forms-Based Authentication

Configuring an ASP.NET application to use forms-based authentication requires no administrative access to the Web server. Here are the basic steps:

- 1 Choose a folder scheme that divides your application into protected and unprotected areas. Many such schemes are possible, but the most straightforward are these:
 - The entire application is protected except for one folder.
 - Only one folder in the application is protected.

There's nothing intrinsically wrong with having multiple protected or unprotected folders. Just do yourself a favor, and keep things simple.

The Insider Extra sample Web site is unprotected except for one folder: selfregnet/securezone.

- 2 Choose a name and location for your logon page. This needs to be an unprotected folder because the visitors who need to log on won't be logged on yet! The logon page for the Insider Extra sample Web site is selfregnet/login.aspx.
- 3 Modify or create a web.config file that activates forms-based authentication, specifies which areas to protect, and specifies the name of the logon page. More information on this step will follow shortly.
- 4 Code the logon page, taking care to save it in the location you specified in step 3. This page can prompt the visitor for whatever information you want and match it to any source you want. However, at some point, it must call a method named *FormsAuthentication.RedirectFromLoginPage*. This establishes an identity for the visitor and redirects the visitor to the protected page he or she originally tried to access.

Here are the contents of the web.config file in the Insider Extra sample Web site. The line numbers are for reference only; they don't appear in the actual file.

```

1 <configuration>
2   <appSettings>
3     <add key="visdb" value="fpdb/visitors.mdb" />
4     <add key="conffrom" value="admin@cpandl.com" />
5     <add key="confreply" value="admin@cpandl.com" />
6     <add key="smtpserv" value="smtp.cpandl.com" />
7   </appSettings>
8   <system.web>
9     <customErrors mode="Off"/>
10    <compilation debug="true"/>
11    <authentication mode="Forms">
12      <forms loginUrl="selfregnet/login.aspx"
13        name=".fp1lextras" protection="All"
14        timeout="15"/>
15    </authentication>
16    <authorization>
17      <allow users="*,?"/>
18    </authorization>
19  </system.web>

```

Microsoft Office FrontPage 2003 Inside Out

```

20     <location path="selfregnet/securezone">
21         <system.web>
22             <authorization>
23                 <deny users="?"/>
24             </authorization>
25         </system.web>
26     </location>
27 </configuration>

```

Like all web.config files, this one contains XML:

- The <configuration> and </configuration> tags form a container for the entire file.
- The <appSettings> node on lines 2 through 7 contains miscellaneous settings that control the application.
- The <system.web> node on lines 8 through 19 forms a container for application defaults. Within this container:
 - The <customErrors> tag on line 9 tells ASP.NET to display a fully detailed error page even if the current visitor is accessing the server over a network connection.
 - The <compilation> tag on line 10 makes debugging the default for all pages in this application.
 - The <authentication mode="Forms"> node on lines 11 through 15 specifies that forms-based authentication is in effect for the current application. To specify an authentication mode of *Windows*, *Passport*, or *None*, code one of these keywords instead of the keyword *Forms*. *None* is the default.

The <forms> tag on lines 12 through 14 configures forms-based authentication. Here are the attributes you can code within this tag:

- **loginUrl** Specifies the URL of a page that ASP.NET will display when an unauthenticated visitor requests a protected page. The default is default.aspx, but the Insider Extra sample Web site uses selfregnet/login.aspx.
- **Name** Specifies a name for the HTTP cookie that forms-based authentication will use. The default is .ASPXAUTH. To avoid confusion with other forms-based authentication sites on the same server, the Insider Extra sample Web site uses a cookie name of .fp11extras.
- **Protection** Controls the means that ASP.NET will use to protect the value of the authentication cookie. Two kinds of protection are available:
 - **Encryption** Means that ASP.NET will encrypt the authentication cookie. ASP.NET uses Triple-DES if it's available and the key is 48 bytes or more; otherwise, it uses DES. This ensures that Web visitors can't decipher or inspect the cookie value.

Note Data Encryption Standard (DES) is a widely-used method of data encryption developed in the 1970s by the US government National Bureau of Standards and National Security Agency. The longer the encryption key, the longer it takes to break this form of encryption.

Building Your Own Self-Registration System

- Validation.** Means that ASP.NET will calculate and append a message authentication code (which is a kind of checksum) to the outgoing authentication cookie. ASP.NET then repeats the calculation for incoming cookie values and compares the two results. If the two checksums match, it's extremely likely that no tampering or corruption has occurred.

By coding the values shown in Table IE14-1, you can specify none, one, or both kinds of protection. The default, and the value that the Insider Extra sample Web site uses, is All.

Table IE14-1. Protection Options for ASP.NET Forms-Based Authentication

Protection value	Uses encryption	Uses validation
All	Yes	Yes
Encryption	Yes	No
Validation	No	Yes
None	No	No

- Timeout** Specifies the approximate number of whole minutes that the authentication cookies will remain valid. A value of 15, for example, means that the cookie will expire after about 15 minutes of inactivity from the visitor. This is the value that the Insider Extra sample Web site uses.
- Path** The path that the application specifies on authentication cookies. The default value is a slash (/). This is because most browsers are case-sensitive and won't send back cookies if there's a path case mismatch.

Avoiding the Path to Cookies

If your application resides at the URL path `/anywhere/`, you might think that forms-based authentication would use a cookie path of `/anywhere/` as well. That way, only your application would receive the authentication cookie.

Because cookie paths are case-sensitive, however, the browser wouldn't return the cookie with any request that specifies a folder name such as `/AnyWhere/`, `/aNYwHERE/`, or `/ANYWHERE/`. Not finding the cookie, ASP.NET would redirect the visitor to the logon URL.

Using a slash (/) for the cookie path avoids case issues, but at a price. Specifically, this path tells the browser to send your application's authentication cookie with every request to your Web server. This involves a bit of extra transmission to and from the browser, but more importantly, it means that two applications on the same server—both using the cookie path `/` and the same cookie name—might end up fighting over control of the same cookie. This is a good reason to specify a custom cookie name for every application.

Microsoft Office FrontPage 2003 Inside Out

The sample web.config file contains the following `<authorization>` node in lines 16 through 18:

```
<authorization>
  <allow users="*,?"/>
</authorization>
```

Because this `<authorization>` node appears within the `<system.web>` section, it specifies the default access rules for the entire application.

An `<authorization>` node can contain any number of `<allow>` and `<deny>` tags. These tags allow access and deny access, respectively. Within each `<allow>` or `<deny>` tag, you can specify any of the following attributes:

- **users** A comma-delimited list of users who will receive the specified permission. A question mark (?) means anonymous users, and an asterisk (*) means all users.
- **roles** A comma-delimited list of roles that will receive the specified permission. In Windows-based authentication, roles correspond to Microsoft Windows NT or Active Directory groups. Forms-based authentication, however, makes no use of roles.
- **verbs.** A comma-delimited list of HTTP transmission methods that should have the specified permission. ASP.NET recognizes the verbs *GET*, *HEAD*, *POST*, and *DEBUG*.

If you specify any users other than ? and *, you must define their user names and passwords elsewhere in the web.config file. For example, to define a user named Morris with a password of chair, you would add these lines. (However, this isn't something you need to do for visitors who log on through a custom logon page.)

```
<authentication ... >
  <forms ... >
    <credentials passwordFormat="Clear">
      <user name="Morris" password="chair" />
    </credentials>
  </forms>
</authentication>
```

To specify access rules that differ from the default for one page or folder tree, add a `<location>` node such as the following after the `</system.web>` end tag for the application. These lines come straight from the previous listing of the Insider Extra Web site's web.config file.

```
20   <location path="selfregnet/securezone">
21     <system.web>
22       <authorization>
23         <deny users="?"/>
24       </authorization>
25     </system.web>
26   </location>
```

The path attribute in the `<location>` tag can specify either a folder name or a file name. The `<system.web>` and `<authorization>` nodes are required. As before, `<deny>` and `<allow>` tags

Building Your Own Self-Registration System

in the `<authorization>` node specify the access rules. In this example, ASP.NET will deny access to anonymous users.

If a visitor doesn't have access to a given page or folder (such as the `selfregnet/securezone` folder in the Insider Extra sample Web site), ASP.NET redirects the visitor to the page you specified in the `loginUrl` attribute of the `<forms>` tag. If this page verifies—through any method you care to code—that the visitor should have access, it calls the `FormsAuthentication.RedirectFromLoginPage` method, as shown here:

```
FormsAuthentication.RedirectFromLoginPage( _
    username, CreatePersistentCookie)
```

Here's how ASP.NET interprets the parameters for this method:

- **username** The name of the user for purposes of cookie authentication. This doesn't need be an account name defined anywhere else.
- **createPersistentCookie** If `True`, specifies that ASP.NET should issue a durable authentication cookie (that is, one that the browser will save on the visitor's hard disk). This saves the visitor from having to log on again days or weeks in the future, but incurs the risk of someone else stealing a copy of the cookie.

If you set this parameter to `False`, the cookie will disappear when the visitor closes all browser windows. This is usually the best choice.

Designing Login Pages for Forms Authentication

Given the configuration in the `web.config` file, ASP.NET will intercept any attempt (by a non-authenticated visitor) to browse an `.aspx` page within the `selfregnet/securezone` folder tree and will instead send the visitor the `selfregnet/login.aspx` page, shown in Figure IE14-3.

Normally, the visitor fills in an e-mail address and password and then clicks Login. The `login.aspx` page then looks for the given e-mail address in a table named `visitors` that resides in the `/fpdb/visitors/mdb` database. It then determines whether all the following conditions are true:

- The e-mail address exists in the database.
- The password that the visitor entered matches the password in the database.
- The visitor has completed a verification process to confirm that the e-mail address is genuine.

If all these conditions are true, the `login.aspx` page calls the `FormsAuthentication.RedirectFromLoginPage` method, passing the visitor's e-mail address as the user name value and passing `False` for persistent cookies. The `FormsAuthentication.RedirectFromLoginPage` method then displays the page that the visitor originally requested.

Microsoft Office FrontPage 2003 Inside Out

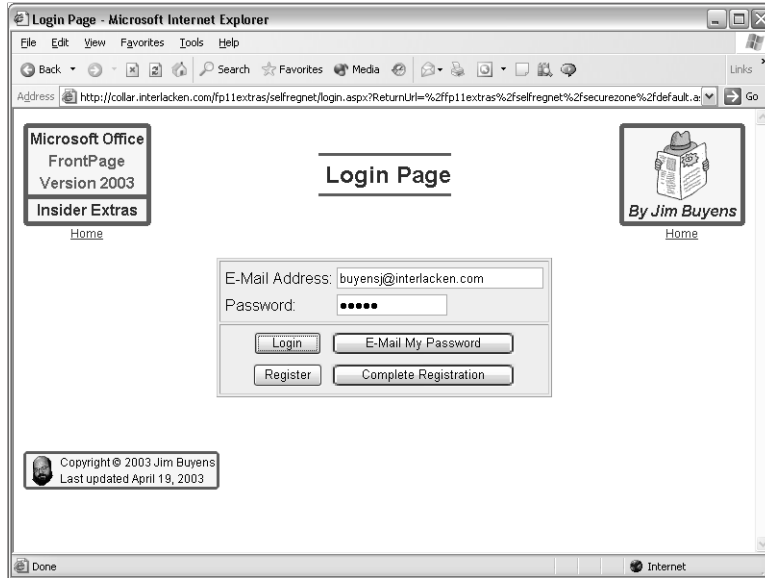


Figure IE14-3. This page from the Insider Extra sample Web site appears whenever a non-authenticated visitor tries to browse a protected Web page.

Tip Provide a default.aspx Page for Every Application

If a visitor browses the login.aspx page directly, the *FormsAuthentication.RedirectFromLoginPage* method redirects the visitor to a default.aspx page in the application's root folder, even if no such page exists. This is a good reason to provide a default.aspx page in the root folder of every ASPNET application that uses forms-based authentication.

A visitor who forgets his or her password can click the E-Mail My Password button. The login.aspx page responds by sending the visitor a message containing the current password. This method provides some confidence that only the holder of the e-mail account can learn the password.

If the visitor doesn't have an account, he or she can click the Register button to create one. This button displays the selfregnet/register.aspx page, shown in Figure IE14-4.

Building Your Own Self-Registration System

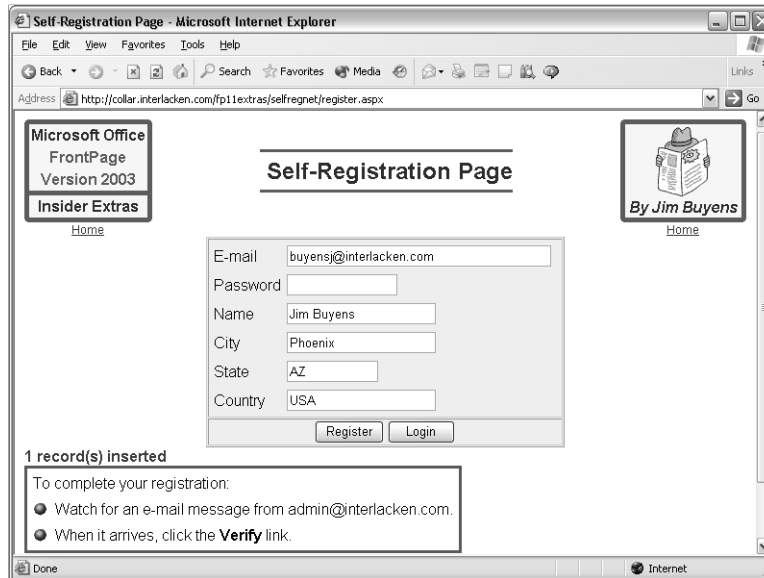


Figure IE14-4. Visitors can use this page to create their own login accounts. However, such accounts require verification.

To register, the visitor enters his or her e-mail address, desired password, name, city, state, and country and then clicks the Register button. The register.aspx page verifies that the given e-mail address isn't in use and displays an error message if it is. Otherwise, the page adds a record to the *visitors* table of the visitors.mdb database. This record contains the information that the visitor entered, the current date, and two special fields:

- **confnumber** A random number between 1 million and 10 million
- **confirmed** A True/False field that the register.aspx page initializes to *False*.

The idea is to send the visitor an e-mail message that contains a hyperlink back to the site, with the visitor's e-mail address and confirmation number as query string variables. Figure IE14-5 shows such a message.

Notice that the visitor must receive the e-mail message to get the link containing the random 7-digit number. A visitor who wants to use a false e-mail address might figure out the link format but has a 1 in 10 million chance of guessing the confirmation number.

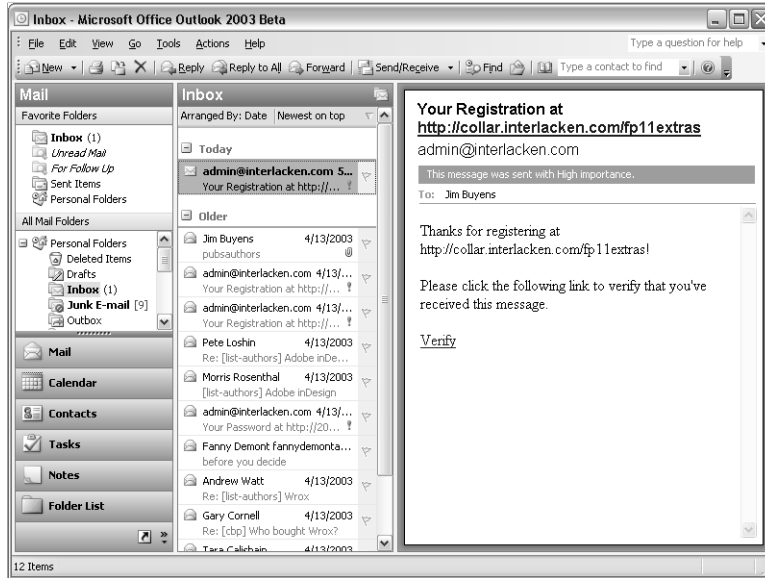


Figure IE14-5. Visitors who self-register receive a message like this and must click the Verify link before using their account to access Web pages.



Inside Out

No identification system is perfect

Verifying that a Web visitor can receive e-mail at a registration address provides some confidence that the address is genuine, but it's hardly foolproof. For example, a visitor can obtain a free Hotmail or Yahoo account, use it to register on your site, and then never use the free address again. Nevertheless, some verification is better than none.

Clicking the Verify link in Figure IE14-5 loads the page shown in Figure IE14-6. A quick check of the Address bar in this screen shot reveals the link format.

The verify.aspx page verifies whether all of the following conditions are true:

- The *visitors* table in the visitors.mdb database contains a record for the given e-mail address.
- The confirmation number in the query string matches the *confnumber* field in the database.
- The *confirmed* field in the *visitors* table in the visitors.mdb database is *False*.

Building Your Own Self-Registration System

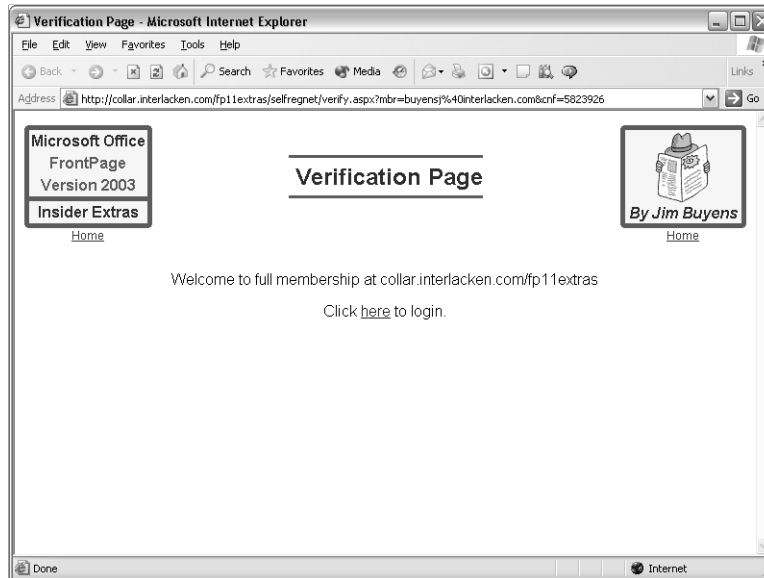


Figure IE14-6. Visitors who successfully complete registration receive this page.

If they are, it changes the *confirmed* field to *True* and displays the message shown in Figure IE14-6. Otherwise, it displays an error message. In either case, the visitor can click the Click Here To Login link and return to the login.aspx page.

The function of the Complete Registration button in the login.aspx page (Figure IE14-3) should now be clear. If a visitor tries to log in, gets the message *Registration never completed for E-Mail Address*, and can't find the original verification message, he or she can click the Complete Registration button to get a new verification message.

The code in the login.aspx, register.aspx, and verify.aspx pages isn't terribly complex, but it's too long to list in a discussion of this type. To view the code, open the page in FrontPage, and switch to Code view. The files *getvisconstr.vb* and *mailers.vb* contain code that appears in more than one Web page. The statements that include these files are:

```
<script runat="server" src="mailers.vb"></script>
<script runat="server" src="GetVisConStr.vb"></script>
```

If you decide to use these pages in your own site, you'll no doubt want to change the page headers, page footers, fonts, and colors. You should also change the settings shown below, which appear on lines 3 through 6 in the web.config file.

```
2 <appSettings>
3 <add key="visdb" value="fpdb/visitors.mdb" />
4 <add key="conffrom" value="admin@cpandl.com" />
5 <add key="confreply" value="admin@cpandl.com" />
6 <add key="smtpserv" value="smtp.cpandl.com" />
7 </appSettings>
```

Microsoft Office FrontPage 2003 Inside Out

Here are the settings:

- *visdb* Specifies the location of the visitors.mdb file, relative to the application root.
- *conffrom* Specifies the From e-mail address.
- *confreply* Specifies the Reply-To e-mail address.
- *smtpserv* Specifies the name of the SMTP mail server to use for sending confirmation and password notification messages.