



Robert L. Bogue
MS MVP, MCSE, MCSA:Security
317-844-5310
Rob.Bogue@ThorProjects.com

Customize a SharePoint Login Page

Working with SharePoint can get complicated. Even seemingly basic tasks are wrapped up in so many layers of automation and support that it can be hard to sort out what you need to do to accomplish your goals. Trying to determine how to change the login page in SharePoint is no exception. For this reason you'll discover how you can modify an out-of-the-box login page to build your own, which you can customize as you desire—including associating it with custom code.

To be able to even see the login form you'll need to turn on forms-based authentication. There are several good resources that address turning on this feature, including Steve Peschka's post on the "[SharePoint Products and Technologies Team Blog](#)" and Andrew Connell's blog post, "[HOWTO: Configure a MOSS 2007 Publishing Site with Dual Authentication Providers and Anonymous Access](#)." These articles walk you through all of the details necessary to learn how to set up the secondary authentication (membership) providers and how to make them accessible. However, neither resource touches on how to modify the login page itself.

Layouts and Login

The out-of-the-box login page lives on the server as `_layouts/login.aspx` or on the file system in `\Program Files\Common Files\Microsoft Shared\Web server extensions\12\TEMPLATE\LAYOUTS`. This location makes the page an *application page*—one that lives outside of any single SharePoint site. This setup is both good and bad. It's good in that it works no matter what the site is or how the site has been provisioned. It's bad in that it doesn't use the same process for modifying a page that a regular SharePoint site page may use.

Because it's a friendly ASP.NET 2.0 application, SharePoint uses the master page functionality within ASP.NET 2.0. Within the context of a site the master pages are controlled at the site level. The master pages can be customized and updated to reflect the organization's own branding. However, being outside of the traditional site structure you're technically not able to modify the master pages that are used by the application pages in the `_layouts` directory. There are two basic approaches to solving this problem.

The first approach is to create a copy of the `LAYOUTS` directory from `\Program Files\Common Files\Microsoft Shared\Web Server Extensions\12\TEMPLATE` and place it underneath the root of the web site as a `_layouts` folder—or create the copy on the file system and define an IIS virtual directory with the name `_layouts`, which points to the copy of the files. While this approach technically works and does relieve the concern of having modified the `_layouts` directory, it inherently creates some problems—not the least of which is that it tends to break solutions packages that deploy new files in or under the `_layouts` directory. This approach also has to be provisioned manually on each front-end web server because there's no automated process for doing it. And even deleting and recreating a web application can remove this virtual directory from IIS, leaving you with the original unmodified files.

A better approach is to create subdirectories under the `LAYOUTS` directory for your files. For instance, you might create your login page under a directory called `DEVX`. You can then reference a `login.aspx` page in this directory as `_layouts/DEVX/login.aspx`. It's a bit of a longer path, but ultimately it gets you to the same result and is more supportable because service packs and upgrades should leave your files alone. Further, you can deploy the files through a solutions package because they are in the `_layouts` directory.

This approach doesn't work for every file in the `LAYOUTS` directory because many of them are referenced from SharePoint itself. However, `web.config` entries control the login page, and they can be changed to point to your new page.

For this discussion you'll use the second approach for the subdirectory (creating subdirectories under the `LAYOUTS` directory) rather than the first one because the second approach is more deployable.

Creating Code-Behind

Based on the foregoing discussion, making a copy of the *simple.master* that *login.aspx* uses, copying *login.aspx* into a subdirectory, and modifying *web.config* to reference the file in the new directory is all relatively trivial. However, just copying the files has an important limitation: you cannot use compiled code in the page. Sure, you can include code inline in the page; however, this code isn't compiled and because it's *uncompiled* it's a higher security risk than the same code in a compiled state.

In their book, *Inside Windows SharePoint Services 3.0* (Microsoft Press 2007), Ted Pattison and Daniel Larson discuss a technique in which you inherit your pages from *LayoutsPageBase* to use them in the *_layouts* directory. This great technique comes with one small problem: it won't work for the login page. *LayoutsPageBase* is designed to make sure the user is authorized to see the content. However, because you haven't authenticated who the user is yet, that authorization won't work. Luckily, *LayoutsPageBase* derives from *UnsecuredLayoutsPageBase*, which exposes a protected Boolean property, *AllowAnonymousAccess*. You can override this Boolean property to return *true* to indicate that anonymous users should have access to the page.

In addition to knowing what class to inherit from, you need to know how to create the compiled code. You can create it by producing a separate class library to contain the class that the page will inherit. In other words, the page won't inherit directly from *UnsecuredLayoutsPageBase*; it will derive from a class that you create, which derives from *UnsecuredLayoutsPageBase*. The *aspx* page will in turn inherit from the class that you've created.

The special bit of magic that makes this technique really slick is that you can create protected properties in your class for each of the user interface controls that appear on the login page. When ASP.NET compiles the page it will connect the controls declared in your custom base class with the controls in the ASPX file, provided that the ID of the control in the ASPX matches the name of the field in your custom class. In other words, you don't have to worry about how the fields get populated; ASP.NET will handle this work for you.

Assemble the Pieces

Now that you have an understanding of the basic components of how the solution fits together it's time to create your own solution that leverages these components. For the purposes of the example, you're going to create a login page that allows users to log in through their email addresses rather than through their usernames. It has become a common practice in the industry to have users log in with their email addresses rather than a username. This technique seems to be easier for users to remember, and email addresses are generally unique.

It might seem like changing to a login through an email address would be easy. You change the property mapped to username in the configuration of the *System.Web.Security.ActiveDirectoryMembershipProvider*, and—poof—you have your solution. However, while it's true that you would be able to make this change work for all internal users by using *UserPrincipalName*, external users wouldn't be able to log in using their email addresses because they would be able to log in only with the email address that corresponds to their account in active directory. Generally, external users are not going to think of this requirement.

[Figure 1](#) shows the principal user name on the Account tab. The authentication provider will support this address out of the box. [Figure 2](#) shows the E-mail property on the General tab that can be set to any domain, whether or not it's related to the active directory domain.

There is, however, a way to make this change work. The *ActiveDirectoryMembershipProvider* supports a standard *MembershipProvider* method called *FindUsersByEmail()*. If you add the attribute *enableSearchMethods="true"* to the configuration of the provider, you can get users' usernames and pass them into the *ValidateUser()* method to validate who they are. This approach allows users to log in with their email addresses, as specified on the General tab in active directory.

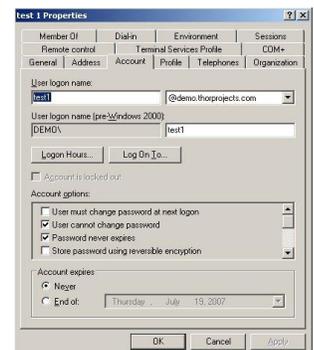


Figure 1. Principal User Name: The authentication provider offers out-of-the-box support.

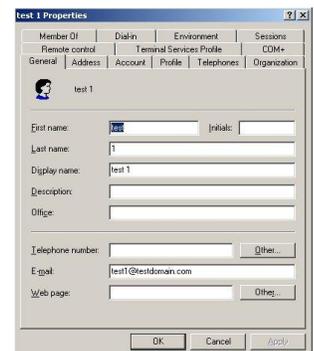


Figure 2. The Real Email Address: Any domain can be set for the E-mail property, even those not associated with active directory.

Membership Provider Configuration

Perhaps the best place to start is by briefly reviewing the settings in the *web.config* file for the web application to make sure that you have the *enableSearchMethods* attribute set as mentioned previously. This code snippet should be in the *web.config* file for the web application hosting the forms authentication:

```
<membership defaultProvider="AD">
  <providers>
    <add name="AD"
      type="System.Web.Security.ActiveDirectoryMembershipProvider,
      System.Web, Version=2.0.0.0, Culture=neutral,
      PublicKeyToken=b03f5f7f11d50a3a"
      connectionStringName="EmailConnect"
      connectionUsername="DEMO\Administrator"
      connectionPassword="123"
      attributeMapUsername="UserPrincipalName"
      enableSearchMethods="true"/>
  </providers>
</membership>
```

Note that you'll need to reference your connection string name, and you'll need to use your credentials. Make sure that the *enableSearchMethods* attribute is set.

Login Page Code-Behind

With the membership provider set up it's time to create a project to hold the code-behind for the login page. Begin by creating a new Visual Studio 2005 Class library project with an appropriate name, such as CustomLoginPage. Remove the *class1.cs* file; add a new class file, *CustomLogin.cs*; and add to that file these contents:

```
using System;
using System.Web.Security;
using System.Web.UI.WebControls;

namespace CustomLoginPage
{
  public class Login :
    Microsoft.SharePoint.WebControls.UnsecuredLayoutsPageBase
  {
    protected System.Web.UI.WebControls.Login loginBox;
    protected override bool AllowAnonymousAccess { get { return true; } }
  }

  protected override bool AllowNullWeb { get { return true; } }

  protected void Login_Click(object sender, EventArgs e)
  {
    if (AuthenticateUser(loginBox.UserName, loginBox.Password))
      return;
  }

  protected bool AuthenticateUser(string emailAddr,
    string password)
  {
    string userName = emailAddr;
    MembershipUserCollection coll =
      Membership.FindUsersByEmail(emailAddr);
    if (coll != null && coll.Count == 1)
    {
      // We're doing this to force the enumerator to give us the
      // one and only item because there is no by int indexer
    }
  }
}
```


Deploy and Run

Now you're ready to deploy the code and run it. In the deployment category there are two steps: First, deploy the CustomLoginPage assembly to the web application's *bin* directory, or alternatively to the GAC. Second, in the *web.config* file modify the *configuration/system.web/authentication/forms* node's *loginUrl* attribute to point to the new location for the *login.aspx* page.

All that's left to do is to open a web browser to the web application and try to go to a secure page—or click the login link—to display a new login page. You'll be able to log in with your e-mail address as well as your user name. Sometimes relatively simple procedures like this one can be the difference between thousands of hours writing custom authentication providers and a handful of hours spent enhancing a page in SharePoint. Which one would you prefer?

About The Author

Robert Bogue, MCSE (NT4/W2K), MCSA:Security, A+, Network+, Server+, INet+, IT Project+, E-Biz+, CDIA+, is president of Thor Projects LLC, which provides SharePoint Consulting services to clients around the country. He has contributed to more than 100 book projects and numerous other publishing projects. His latest book is The SharePoint Shepherd's Guide for End Users. (You can find out more about the book at www.SharePointShepherd.com.)

Bogue has been part of the Microsoft Most Valuable Professional (MVP) program for the past 5 years. He was most recently awarded for Microsoft Office SharePoint Server. Before that, Bogue was a Microsoft Commerce Server MVP and Microsoft Windows Servers-Networking MVP.

Bogue runs the SharePoint Users Group of Indiana (SPIN, www.spindiana.com), and he is also a member of the steering committees for the Indiana Windows Users Group and Indianapolis .NET Developer Association. In addition to speaking at local and regional events, Bogue speaks at national conferences. He blogs at www.thorprojects.com/blog, and you can reach him at Rob.Bogue@thorprojects.com.