

# Introduction to the Personal Web Site Starter Kit

Bill Evjen

Reuters

March 2005

Applies to:

Microsoft ASP.NET 2.0

Visual Web Developer 2005 Express Edition

Visual Studio 2005

**Summary:** Learn about the new Personal Web Site Starter Kit, which is an available project type in Visual Studio 2005 and Visual Web Developer 2005 Express Edition. (22 printed pages)

## Contents

[Introduction](#)

[Working with the Starter Kit Default Files](#)

[The Master Page: Default.master](#)

[The Configuration Page: Web.config](#)

[The First Content Page: Default.aspx](#)

[Including Your Resume Using Resume.aspx](#)

[Your Links to the World: Links.aspx](#)

[Picture This: Albums.aspx](#)

[Administering Your Own Albums](#)

[Registering Users: Register.aspx](#)

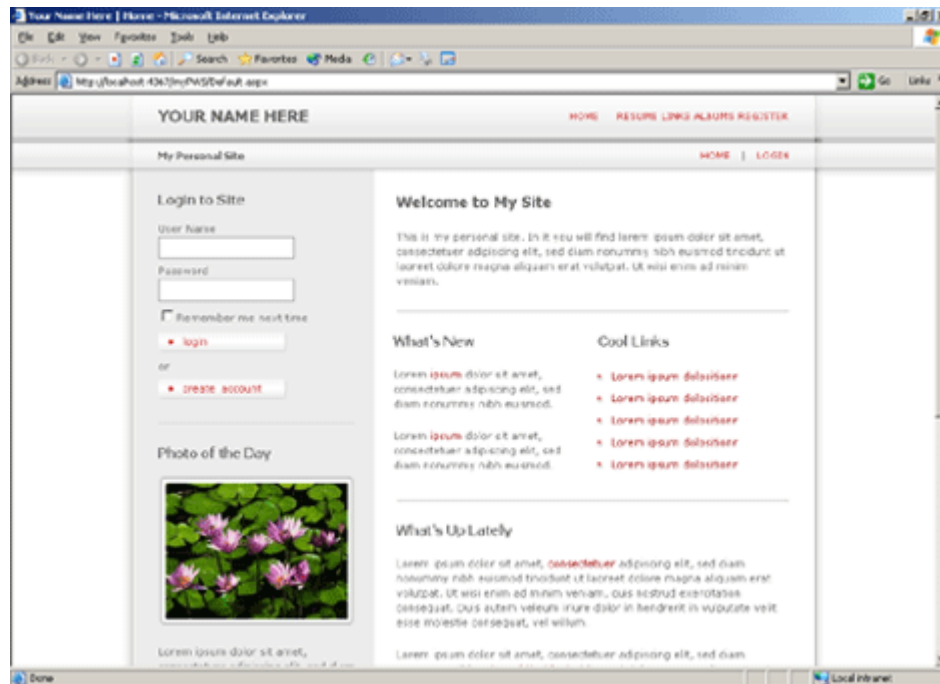
[Summary](#)

[Related Books](#)

## Introduction

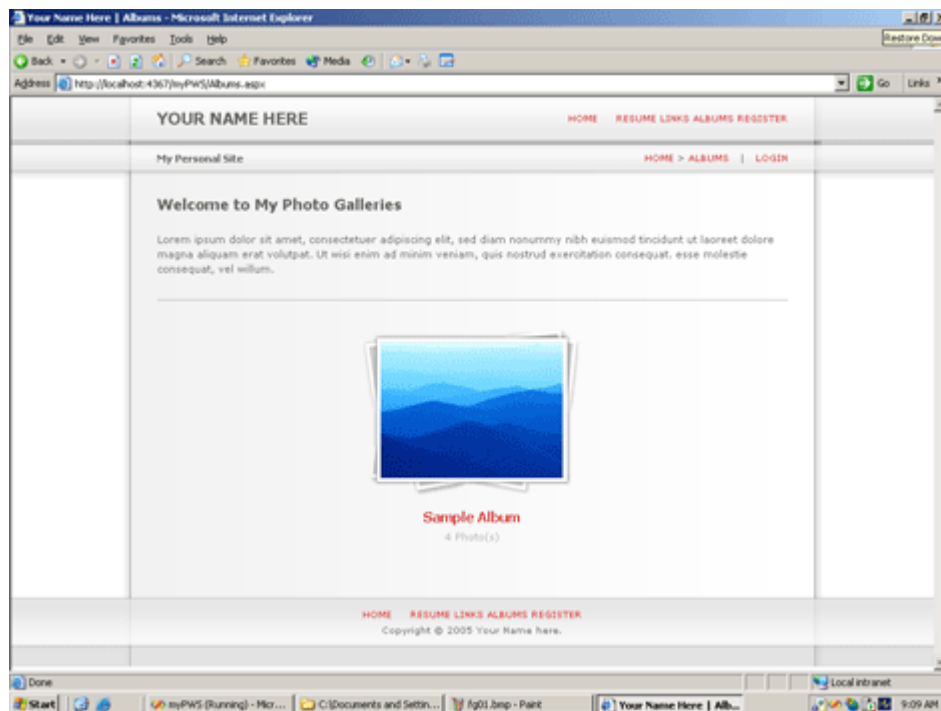
Inside Visual Web Developer 2005 Express Edition, you will find the Personal Web Site Starter Kit, ready for you to start working with right away. The Personal Web Site Starter Kit is designed to

provide you with a tool that you can use to quickly put up a worthwhile Web site. This starter kit provides you with a basic home page, a page for your resume, another to place a collection of your favorite links, and another page that enables you to publish your photos. The home page of the Personal Web Site Starter Kit is presented here in Figure 1.



**Figure 1. Personal Web Site Starter Kit home page**

The biggest focus of this starter kit is the Albums page, which allows you to publish photo albums. One nice feature of the of the Albums page is that is allows for you to publish albums for the general public, or you can create restricted access to an album and allow it to be viewed only by a select audience. The default Albums page is shown in Figure 2.



**Figure 2. Photo Gallery home page**

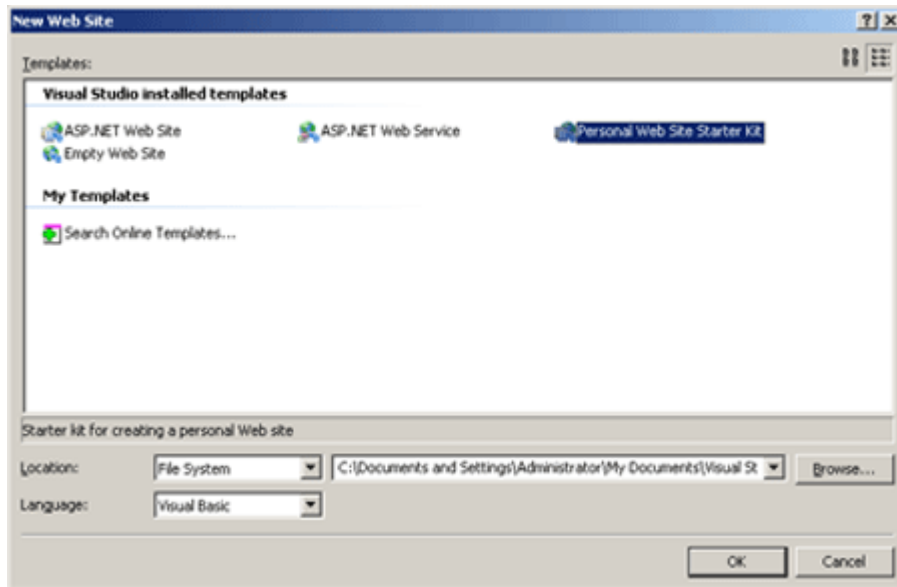
All the pages in the Personal Web Site Starter Kit are ready for you to delete the default *lorem ipsum* text and replace it with your own words. Doing this will instantly give you a personalized Web site.

The Personal Web Site Starter Kit is a great way for you to learn the fundamentals of how to build a site. You can view the code from the pages of this application, and it uses some of the strongest features from the latest release of ASP.NET. Even if you are not interested in using this starter kit for an actual production Web site, it is still a valuable resource to use for learning on how to create an application using ASP.NET 2.0.

Before we look at working with this application, let's first start by taking a look at how it is installed.

## **Working with the Starter Kit Default Files**

Installing the Personal Web Site Starter Kit is as simple as opening a project in Visual Web Developer 2005 Express Edition, since you will find that the application is one of the available projects in this IDE. To create an instance of the starter kit, click **File**, then select **New Web Site** from the Visual Studio menu. This will open the **New Web Site** dialog box, in which you can select the Personal Web Site Starter Kit, as shown in Figure 3.



**Figure 3. New Web site**

After you have selected the application, your first step will be to actually build the application and pull it up in a browser. This is so that the database file, `ASPNETDB.mdf`, is created for this particular application. This file is used for the membership and role management capabilities that are now provided by ASP.NET 2.0. When the application is first run, we are interested in having the application create a couple of roles within the role management system.

How are these roles created when the application is first run? Actually, it is pretty interesting how this occurs. Let's take a look at the application's `Global.asax` file to see how this occurs. Note that the `Global.asax` file contains an `Application_Start` function as shown below in Listing 1.

**Listing 1. Creating new roles when the application is first run from the `Global.asax`**

[Copy Code](#)

```

Sub Application_Start(ByVal sender As [Object], ByVal e
As EventArgs)
    AddHandler SiteMap.SiteMapResolve, AddressOf
Me.AppendQueryString
    If (Roles.RoleExists("Administrators") = False) Then
        Roles.CreateRole("Administrators")
    End If

```

```
If (Roles.RoleExists("Friends") = False) Then
    Roles.CreateRole("Friends")
End If
End Sub
```

From this method, you can see that a couple of `If Then` statements first check to see whether the roles of the Administrators or Friends exist in the system. This is done using the `Roles` class's `RoleExists` method. If this check is found to be `False`, then the role is created using the `CreateRole` method.

Once you pull up the application, you can then shut it down, as the next step is to create an administrator user for the application. To create an administrator for the application, click the **ASP.NET Configuration** button from the Visual Studio Solution Explorer. This will pull up the Web Site Administration tool directly in Visual Studio. You can also pull up this page by selecting **Website** and then clicking **ASP.NET Configuration** from the Visual Studio menu.

After the ASP.NET Configuration page is created and pulled up directly in Visual Studio, you should then use this tool to create a new user who represents the administrator of the site.

To create this administrative user, click the **Security** tab from the ASP.NET Configuration page. This page is shown in Figure 4.

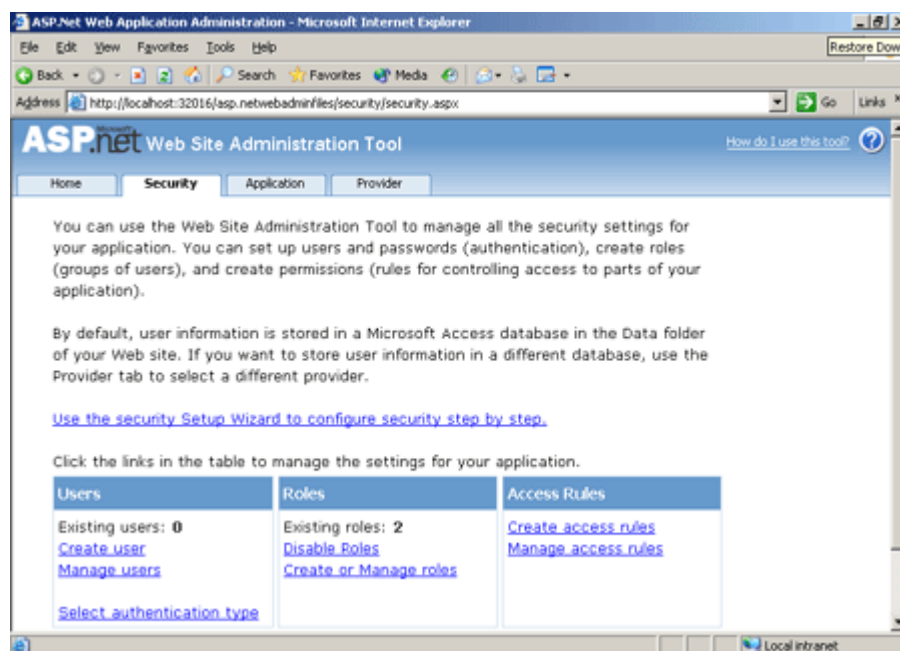


Figure 4. Administration page

From the Security page, you can create an administrator user by selecting the *Create user* link. This will pull up a page with a form for you to input all the user's information, which will be saved to the ASPNETDB.mdf file. This form is shown in Figure 5.

The screenshot shows a Microsoft Internet Explorer window titled "ASP.NET Web Application Administration - Microsoft Internet Explorer". The address bar shows "http://localhost:32016/asp.netwebadminfiles/security/users/addUser.aspx". The page has a blue header with the "ASP.NET" logo and the text "Web Site Administration Tool". Below the header are four tabs: "Home", "Security" (which is selected), "Application", and "Provider". The main content area has a blue background and contains the following text: "Add a user by entering the user's ID, password, and e-mail address on this page." and "To send the password to the user immediately, select the **Send password** check box." Below this text is a form titled "Create User" with a sub-header "Sign Up for Your New Account". The form contains the following fields: "User Name:" with the value "admin", "Password:" with masked characters "\*\*\*\*\*", "Confirm Password:" with masked characters "\*\*\*\*\*", "E-mail:" with the value "erjen@yahoo.com", "Security Question:" with the value "What is your dog's name?", and "Security Answer:" with the value "Rally". There is a "Create User" button at the bottom of the form. To the right of the form is a "Roles" section with the text "Select roles for this user:" and two checkboxes: "Administrators" (checked) and "Friends" (unchecked). At the bottom of the form, there are three checkboxes: "Send Password" (unchecked), "Autogenerate Password" (unchecked), and "Active User" (checked).

**Figure 5. Adding an administrator**

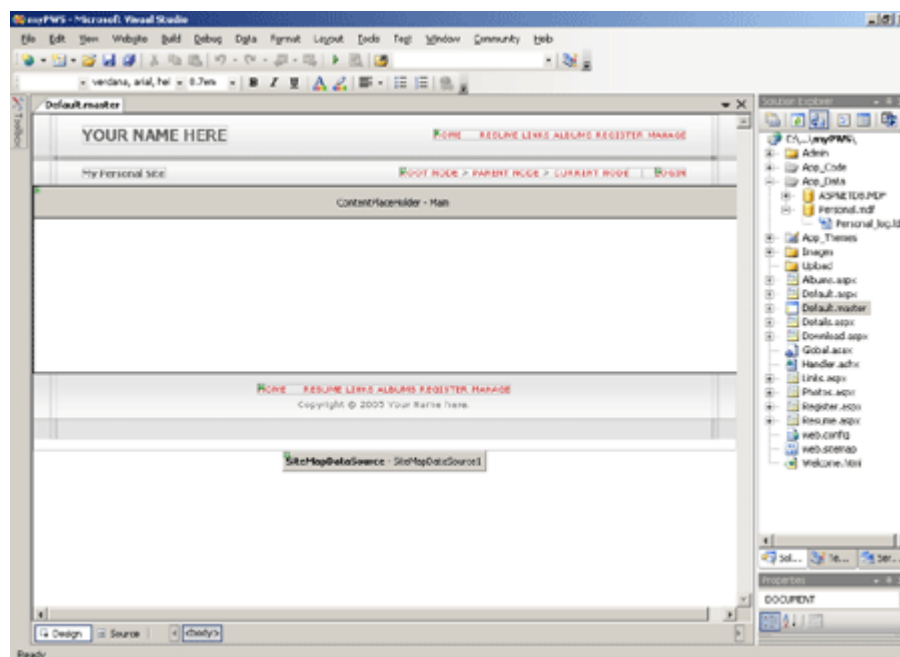
As shown in Figure 5, input the administrator's user name, password, e-mail address, and a security question and an answer for the security question. Before clicking the **Create User** button, be sure to select the roles in which the administrator belongs. For this example, check the **Administrators** check box and then click the **Create User** button to create this administrative user. If your application is going to have more than one administrator, simply create another user using the same process.

Let's next take a look at the pieces of the starter kit so you can understand exactly where you apply the customization points that are required to make this application unique.

## **The Master Page: Default.master**

The first item for this starter kit we will examine is the `Default.master` page. ASP.NET 2.0 introduces a way in which to build templated pages. This means that you can build a master template or a master page, which you can then apply to each and every page you designate. You will notice that the single master page used in the Personal Web Site Starter Kit, `Default.master`, also includes a code-behind page (`Default.master.vb` or `Default.master.cs`), as do the standard `.aspx` pages. However, since this master page is only dealing with presentation and is not concerned about any other business logic, you will find that the code-behind page is just the actual code framework of the code-behind page and doesn't include any actual code.

All the presentation for this page is contained within the `Default.master` page itself. Visual Studio does an excellent job of letting you look at the `.master` page visually. Clicking the **Design** tab in the Visual Studio IDE, you can view what the master page actually looks like. This is shown here in Figure 6.



**Figure 6. Master page**


From this figure, you can see that Visual Studio will apply this look and feel to any content page that uses this master page (I will discuss this shortly). Also, you will find a collection of HTML server and Web server controls placed on the page. These controls include the **Menu**, **SiteMapPath**, and **LoginStatus** controls. Probably the most interesting control on this page is the **ContentPlaceHolder** control.

The **ContentPlaceHolder** control is a defined area that allows for any content page that uses this particular master page to interject content into. Basically, when you construct your master pages, you are allowing content pages to use specified sections of the page. A content page will not be able to work outside the bounds of this content area. Though it is possible to include multiple content areas through the use of multiple **ContentPlaceHolder** controls placed on the master page, this example (our `Default.master` page) uses only one of these controls.

From this master page, you should modify the page either from the Design view shown in Visual Studio, or by directly changing the code of the page from the code view. What are you changing, exactly? Well, for instance, you will probably want to change the *Your Name Here* parts to the name you want to give to the site.

On the `Default.master` page, you can also see that there is a **SiteMapDataSource** control at the bottom of the page. The **SiteMapDataSource** control is one of the new data source controls that have been added to ASP.NET 2.0. This control is designed to work with any `web.sitemap` files that are contained within your application. The `web.sitemap` file is basically an XML representation of your application's page structure that can then be bound to a couple of site navigation controls that you will find at your disposal. In fact, the `Default.master` page includes a couple of these new site navigation controls, one being the **Menu** server control. This control is shown here in Listing 2.

#### Listing 2. Looking at the Menu control

 [Copy Code](#)

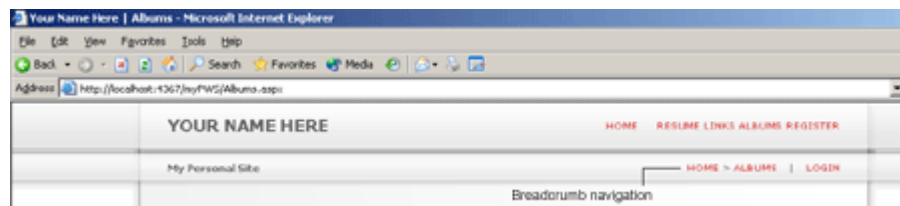
```
<asp:menu id="menua" runat="server"
  datasourceid="SiteMapDataSource1"
  cssclass="menua"
  orientation="Horizontal"
  maximumdynamicsplayscale="0"
  skipinktext=""
  staticdisplaylevel="2" />
```

From this code, you can see that the **Menu** control points to the `web.sitemap` file by way of the **SiteMapDataSource1** control on this page. This is done by using the `DataSourceId` attribute in the **Menu** control. An important attribute to pay attention to in this control is the `Orientation`



attribute, which specifies that the links should be laid out horizontally. Also, note the use of the `StaticDisplayLevels` attribute and the fact that it is only set to a value of 2, so that only the first two layers of links will be displayed (Home being the first layer and Resume, Links, and Albums being the second layer).

Besides the **Menu** control on this page, there is also the **SiteMapPath** control, which is also using the data found in the `web.sitemap` file. When generated and if the end user browses to one of the sub-pages, you can see how this control uses what is called bread-crumb navigation on the page to inform the user where in the application they reside. This is illustrated here in Figure 7.



**Figure 7. Bread-crumb navigation**

The interesting thing about this control is that it is not associated with the `web.sitemap` file through the use of the **SiteMapDataSource** control. Instead, this control directly binds itself to the `web.sitemap` file automatically. In terms of customizing this control, one attribute that is easily modifiable is the `PathSeparator` attribute, which specifies the character that is used between each of the page levels.

As far as modifying the `Default.master` page, let's stop there and move onto some of the other components of the site.

## The Configuration Page: `Web.config`

Having a `web.config` file within an ASP.NET application is nothing new; what is new is all the new features provided by ASP.NET 2.0 that have been made configurable through this XML configuration file.

You can make the biggest modification to the look and feel of your site by changing the theme of the application. Looking at the `web.config` file, you can see that the theme of any page generated is using a theme called *White*.

```
<pages styleSheetTheme="White" />
```

If you look in the Solution Explorer, you will see a folder called `App_Themes`. Inside this folder there are two themes that have been consolidated into two folders—Black and White. By default, the Personal Web Site Starter Kit will use the White theme.

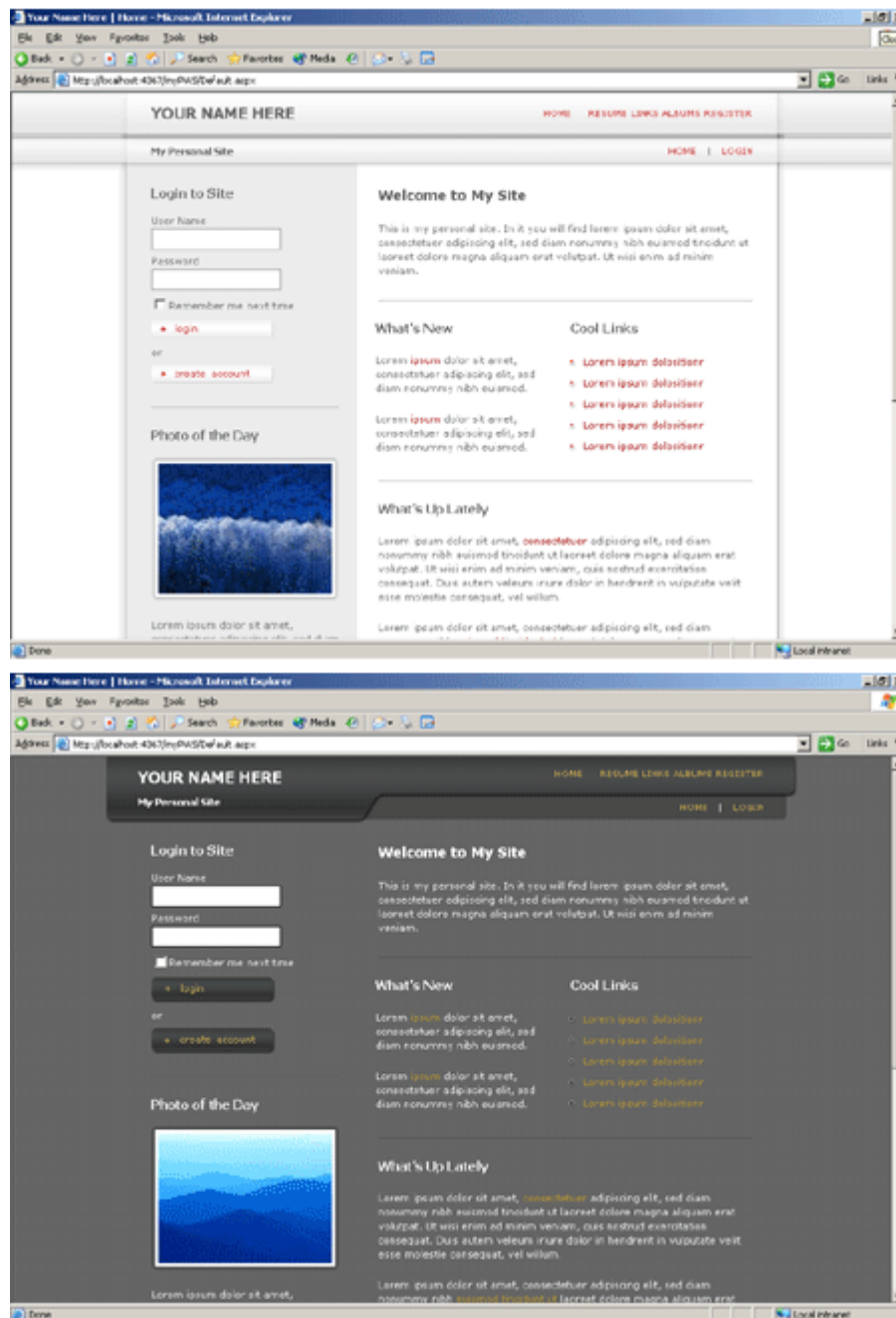
A theme contains the styling for the application. Themes can be applied at a control, page, or site level. In the case of this starter kit though, the theme is applied application-wide through the specification made in the `web.config` file. If you open up one of the theme folders, you will notice that both the Black and White themes are made up of not only `.css` files, but also `.skin` files and images. A `.skin` file defines the styling that is applied to server controls through the use of their attributes.

Changing the theme of the page is as simple as changing the value of the `styleSheetTheme` attribute in the `web.config` file as shown here:

```
<pages styleSheetTheme="Black" />
```

 [Copy Code](#)

Both the Black and White themes of the Personal Web Site Starter Kit are represented here in Figure 8.



**Figure 8. Comparing themes (click to enlarge)**

To modify the overall appearance of the application, you can either choose one of the predefined themes or you can create your own theme. To do this, simply create a new theme folder in the App\_Themes directory and create your own .css, .skin, and images. Then you can either change the theme for an individual page using the Theme attribute in the @Page directive, or you can globally change the theme in your entire application through the use of the styleSheetTheme attribute in the <pages> element of the web.config file.

Now let's take a look at the <location> section at the bottom of the web.config file.

```
<location path="Admin">
  <system.web>
    <authorization>
      <allow roles="Administrators" />
      <deny users="*" />
    </authorization>
  </system.web>
</location>
```

In most cases, developers apply settings to the `web.config` file as a means of applying a configuration setting to everything contained within the application as a whole. Note though that through the use of the `<location>` element in the `web.config` file, you can also control the configurations applied to folders or even specific pages contained within the application that are outside the realm of configurations that were applied to the application as a whole.

The section shown above is defining specific configuration settings for the Admin folder of the Personal Web Site application. This means that everything defined here is defined to each and every file contained within the Admin folder. The settings applied, through the use of the `<authorization>` element, will allow only authenticated users who are in the Administrators role to view the content. If a user is found outside of the Administrators role, then they will be locked out from viewing any of the pages contained in this folder.

### The First Content Page: Default.aspx

Looking at the `Default.aspx` page, you will notice that it isn't your standard `.aspx` page. This is a content page. This page, like most in this application, uses a master page as a template. Let's look at the basic structure of this page, as shown in Listing 3.

#### Listing 3. The structure of the Default.aspx page

```
<%@ Page Language="VB"
MasterPageFile="~/Default.master"
Title="Your Name Here | Home"
```

```
CodeFile="Default.aspx.vb" Inherits="Default.aspx" %>

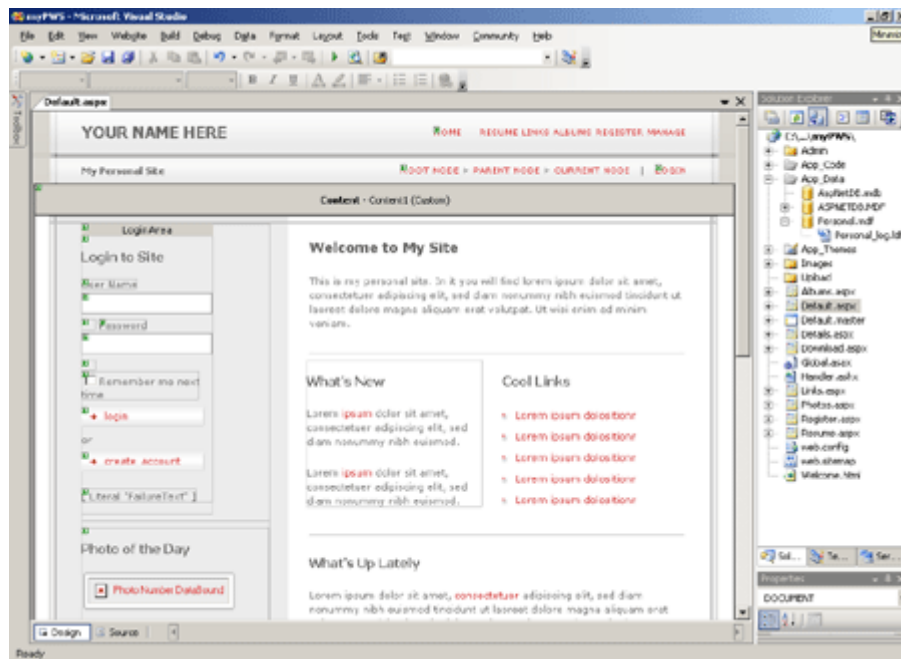
<asp:content id="Content1" contentplaceholderid="Main"
runat="server">

</asp:content>
```

All .aspx pages that are meant to be content pages use the `MasterPageFile` attribute in the `@Page` directive. From Listing 3, you can see that the `MasterPageFile` attribute points to the `Default.master` page. This is the page template from which this content page will inherit.

Since this is a content page, you will not need to include any of the standard HTML tags (such as the opening and closing `<html>`, `<body>`, and `<form>` tags), because these are defined in the master page and there is thus no need to repeat the tag representation. What is included is a single **Content** server control. This control binds itself to the single **ContentPlaceHolder** control that was used on the master page. The association between the two controls is done by using the `ContentPlaceHolderID` attribute within the **Content** control. In Listing 3 you can see that the value of the `ContentPlaceHolderId` attribute is `Main`, which you will find to be the value of the `ID` attribute from the **ContentPlaceHolder** control on the master page. It is within this **Content** control where the entire page's content is defined. If you end up adding additional **ContentPlaceHolder** controls on the master page, then you can also bind to these instances by adding additional **Content** controls on the content page.

The nice thing about using a content page with a master page is that you can work with it in Visual Web Developer 2005 Express Edition. Clicking the Design view of the `Default.aspx` page, you will see the following view of the page.



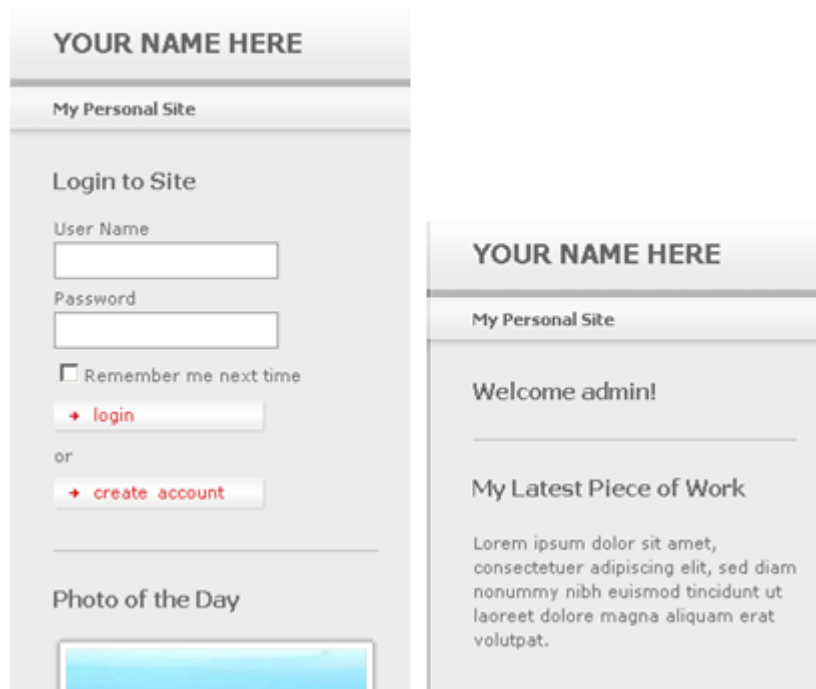
**Figure 9. Design view**

From this figure, you can see that the contents that were defined in the master page are shown in gray, while the sections that are modifiable from the content page are shown in the standard white or clear view.

The `Default.aspx` page is the home page of the application. The first thing you can do here to personalize the application is to replace all the *lorem ipsum* text with meaningful text. All the *lorem ipsum* text on this page is hard-coded text and is not data-driven, so you will just need to change this text directly in the `.aspx` page itself.

The interesting thing about this page is that it includes some of the more exciting pieces of ASP.NET 2.0. The page's left-hand column is defined at the top of the page. The first control defined here is a **LoginView** control. Since we are working with an application that will allow users to log in to the application, this page will have viewers who are considered authenticated and authorized users (meaning that they have logged into the application and were authorized for a specific role). This page will also have viewers who have not gone through the authentication and authorization process. Because of these dynamics, there will be scenarios in which you will display specific data for authenticated users while displaying other content to users who are not authenticated. The **LoginView** control allows for this kind of behavior.

The **LoginView** control has two templates, `<AnonymousTemplate>` and `<LoggedInTemplate>`. Looking at the code from `Default.aspx`, you can see that the `AnonymousTemplate` section includes a form that allows the end user to attempt to log in to the application. If the user is authenticated after logging into the application, they will then be presented with the contents provided in the `LoggedInTemplate` section. This section only contains some text and a **LoginName** control. The **LoginName** control is used simply to present the name of the authenticated user. An example of both of these views is presented in Figure 10.



**Figure 10. Unauthenticated vs. authenticated view**

Below the login form, you will find a **FormView** control, which presents a random photo from a random album. The **FormView** control will get this picture and associated data from the `Default.aspx` page's **ObjectDataSource** control (you will find this control at the bottom of the page). The code for the **ObjectDataSource** control is shown here in Listing 4.

**Listing 4. The ObjectDataSource control**

```
<asp:ObjectDataSource ID="ObjectDataSource1"
    Runat="server"
    TypeName="PhotoManager" SelectMethod="GetPhotos">
```

[Copy Code](#)

```
</asp: ObjectDataSource>
```

The **ObjectDataSource** control is meant to retrieve data from a middle-tier component. In this case, the **ObjectDataSource1** control uses the `TypeName` attribute to point to a class called `PhotoManager` and a method contained within this class called `GetPhotos`. You will actually find this class within the `App_Code` folder of your application. So, what happens is that when the `Default.aspx` page is requested, the `PhotoManager` class is invoked by the **ObjectDataSource** control, which then binds the provided information using simple binding syntax such as `<% Eval("AlbumID") %>`.

To create a thumbnail of the image on the `Default.aspx` page, a custom http handler is employed. The http handler is used from an HTML `<img>` element, as illustrated here in Listing 5.

#### Listing 5. Using the http handler for the images

```
&Size=M"  
class="photo_198" style="border: 4px solid white"  
alt='Photo Number <# Eval("PhotoID") %>' />
```

 [Copy Code](#)

From this example, you can see that the actual image is retrieved from the http handler `Handler.ashx`. You will also find this file contained within your application. Review this file as an example of how you can use your own http handlers from an ASP.NET page.

#### Including Your Resume Using `Resume.aspx`

The starter kit's resume page is another content page. Pulling up this page in the browser, you will see that this page uses the same master page that is used by the `Default.aspx` page.

The `Resume.aspx` page is a page that allows you to present your resume (also known as *curriculum vitae* or simply CV). Besides being a content page, this page simply contains a collection of static images and text, all of which are contained within the page's **Content** server control.

To customize this page by adding your resume to it, simply replace the *lorem ipsum* text with your own. For the image, place a picture of yourself in the `Images` folder of the application and point to this new picture from the page's `<img>` HTML element.



```

```

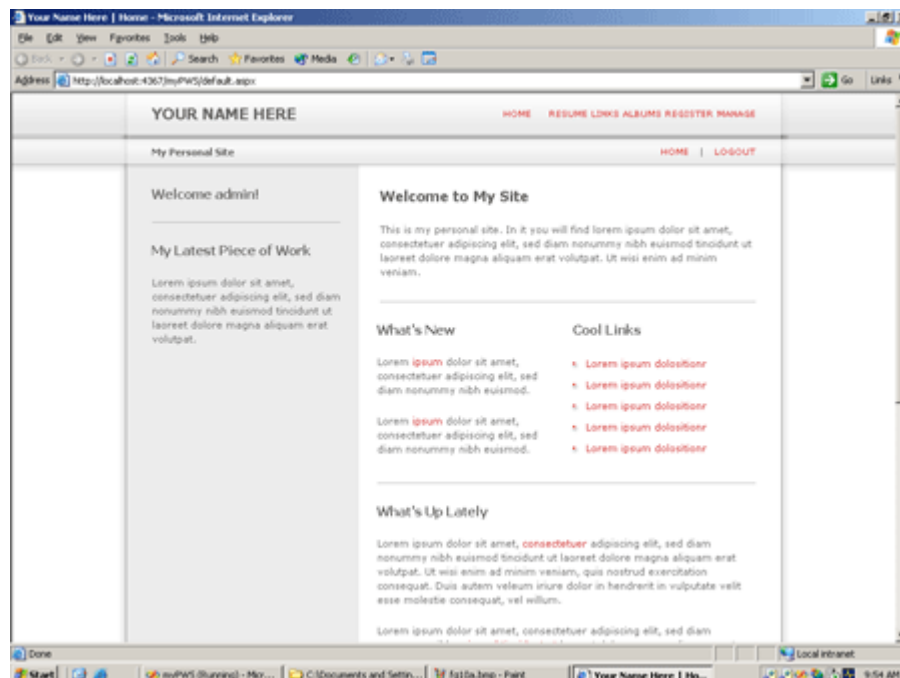
## Your Links to the World: Links.aspx

Another basic page is the `Links.aspx` page. This page is another content page that uses `Default.master`. This page is simply a list of links that are organized into categories such as Top 5, Cool Site Designs, Photo Sites, and Resources.

You can edit the `Links.aspx` page to customize the categories and the links that are presented on this page.

## Picture This: Albums.aspx

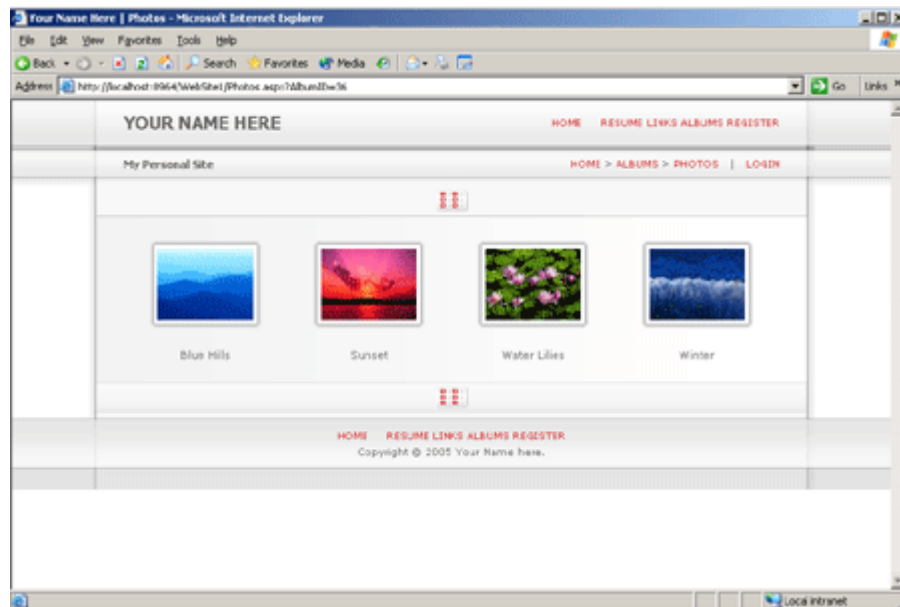
One of the big areas of focus for the Personal Web Site Starter Kit is the photo album system that it contains. Clicking the Album page will bring you to `Albums.aspx`. The default view of this page is shown here in Figure 11.



**Figure 11. Photo Gallery home page (click for larger image)**

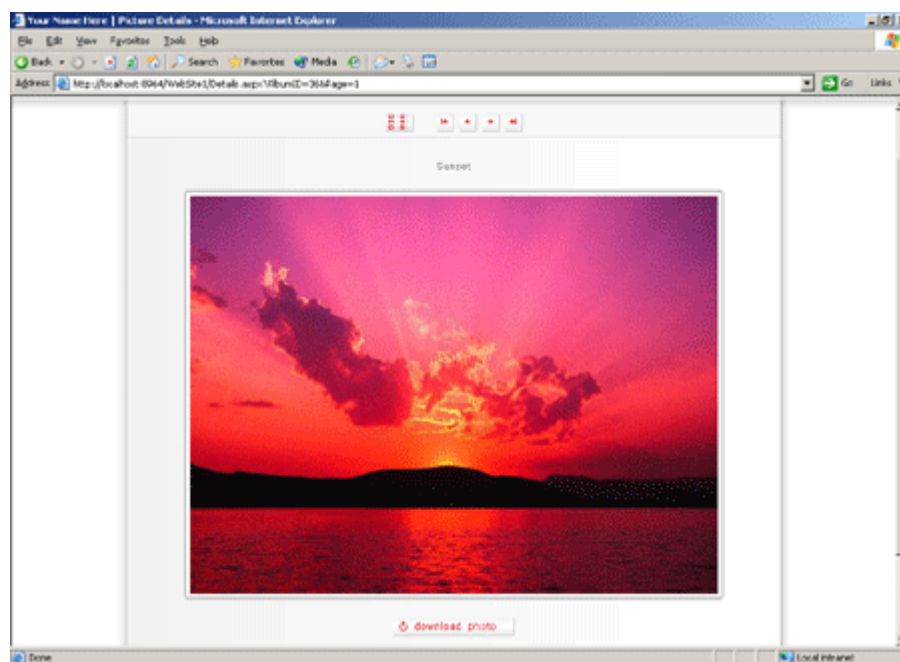
The photo album system included in this application allows you to display all your personal pictures in custom albums. Included in the default view is a single album for demonstration purposes. This

album view shows the first picture in the album as well as showing the number of photos that are included in the album. Clicking the album name (which is a hyperlink) will bring you to a page that displays the pictures contained in the album. This is illustrated here in Figure 12.



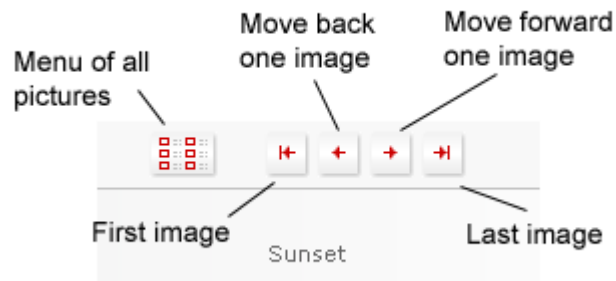
**Figure 12. Photo album**

From this figure, you can see each of the photos in the album is represented with a thumbnail image of the larger image. Also included is the name of the picture. Clicking on one of the images will open the larger image. This is shown here in Figure 13.



**Figure 13. Displaying a photo**

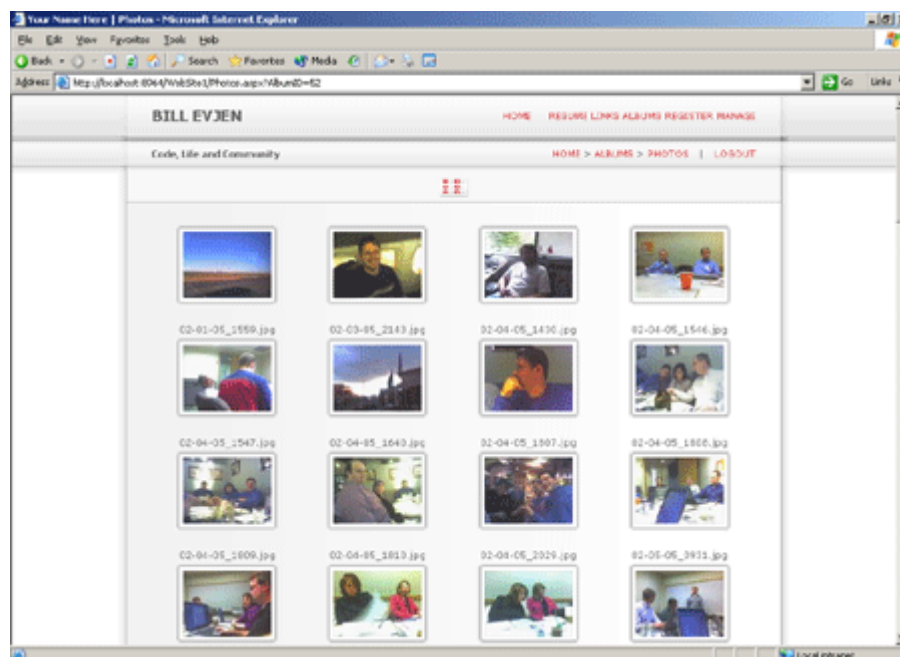
When viewing a single image, you will notice that there is a navigation system provided. This image navigation system is defined here in the following figure.



**Figure 14. Navigating the photos**

This page also includes a **download photo** button. Clicking this button will take you to a page that will show the image in its actual size. The viewer is asked to right-click the image in order to save it to disk.

A personalized Albums.aspx page is shown in Figure 15.



**Figure 15. Personalized album**

Like the home page of the site, which uses an **ObjectDataSource** control to get a random picture from a random album to display as the picture of the day, the Albums.aspx page uses an **ObjectDataSource** control to get a list of albums (plus the first picture from the album) to display.

```
<asp: ObjectDataSource ID="ObjectDataSource1"
Runat="server"
    TypeName="PhotoManager" SelectMethod="GetAlbums">
</asp: ObjectDataSource>
```

In this case, the **ObjectDataSource1** control is using the same class, `PhotoManager`, but instead is using the `GetAlbums` method to retrieve this list of albums. The `GetAlbums` method is shown here in Listing 6.

#### Listing 6. The `GetAlbums` method

```
Public Function GetAlbums() As Generic.List(Of Album)
    command.CommandText = "GetAlbums"
    command.Parameters.Add(new SqlParameter("@IsPublic",
filter))

    Dim reader As SqlDataReader = command.ExecuteReader()
    Dim list As New Generic.List(Of Album)()
    Do While (reader.Read())
        Dim temp As New Album(CInt(reader("AlbumID")), & _
            CInt(reader("NumberOfPhotos")), & _
            CStr(reader("Caption")),
CBool(reader("IsPublic")))
        list.Add(temp)
    Loop

    Return list
End Function
```

From this bit of code, you can see that the `GetAlbums` method takes no input parameters and returns a generic list of `Albums`. The `Albums` type is defined in the `ObjectTypes.vb` or `ObjectTypes.cs` file, which you will also find in the `App_Code` folder. Once each album is

retrieved from the database, some of the album details are presented through the use of some declarative binding, as shown here:

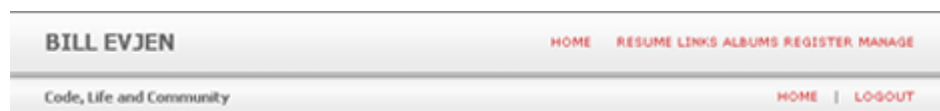
```
<h4><a href="Photos.aspx?AlbumID=<%#  
Eval ("AlbumID") %>">  
<%#  
Server.HtmlEncode(Eval ("Caption").ToString()) %></a></  
h4>  
<%# Eval ("Count") %> Photo(s)
```

From these statements, you can see that the `AlbumID` and the `Caption` are used to construct a hyperlink to the album details. Then the `Count` property is used to provide a count of the number of images that the album contains.

**Note** On a side note, the `GetAlbums` method is a nice example of the new generics capabilities provided by the .NET Framework 2.0. Generics allow you to make a strongly typed collection that allows you to avoid the process of boxing and unboxing, which is what occurs when working with collections outside of collections that are based upon generics. You will find that generic collections are easier to work with and provide better performance.

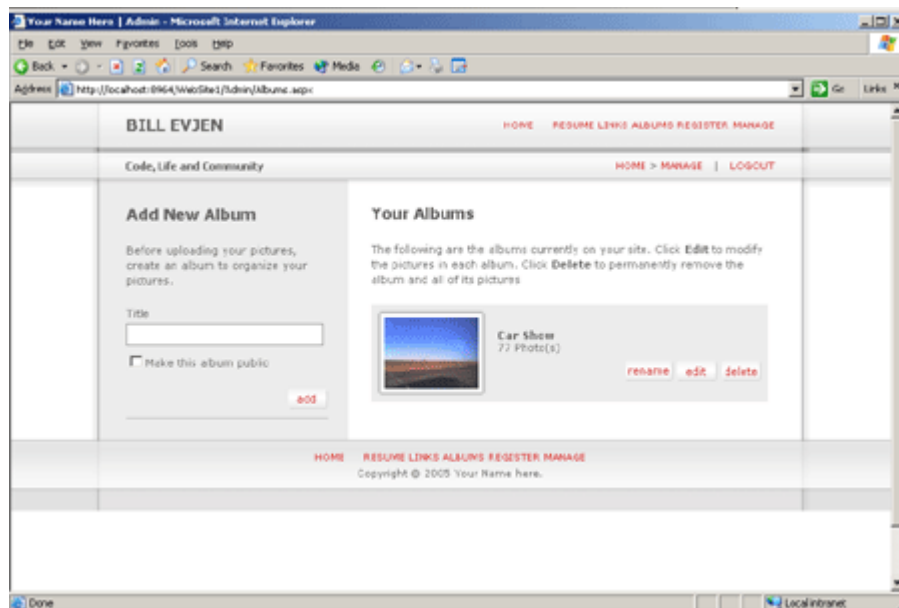
## Administering Your Own Albums

If you logged into the Personal Web Site Starter Kit as a user defined under the Administrators role, you will notice that there is an additional accessible page found in the navigation structure of the application. The name of this page is Manage and is shown here in Figure 16.



**Figure 16. Administering your albums**

Clicking on the Manage page will take you to a page found in `Admin\Albums.aspx`. This `Albums.aspx` page allows you to administer the albums that are presented through your application. This page is shown in Figure 17.

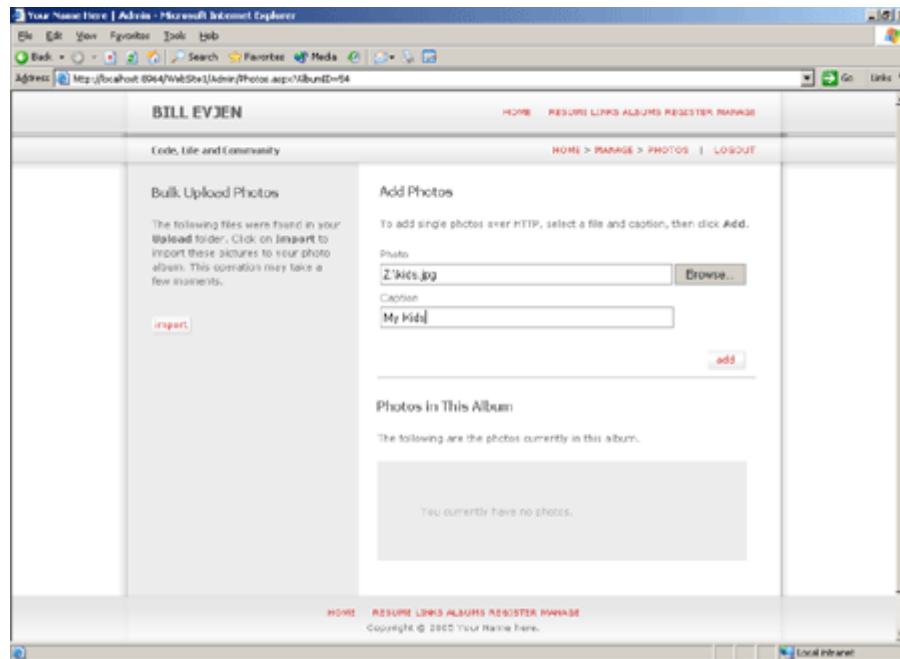


**Figure 17. Creating a new album**

From this view, you can see that the sample album is shown in the list of albums. You can rename, edit, or delete each listed album. Feel free to delete the sample album as we are going to add our own.

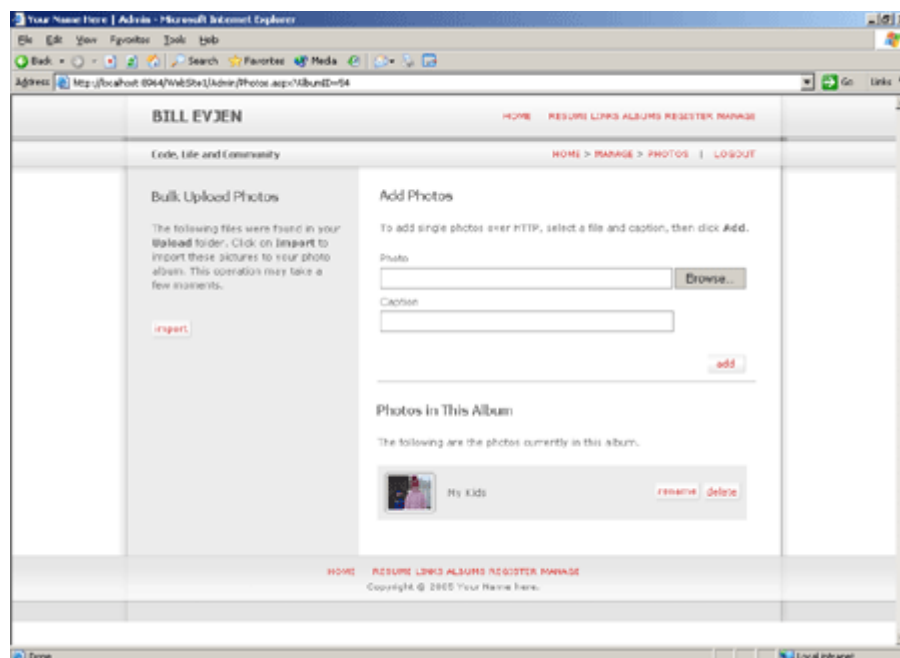
To add a new album, provide a name for the album in the text box of the **Add New Album** section of the page. You will need to decide whether or not this album is going to be a public, viewable album or if it is going to be an album that is considered private. Later, we will cover what it means for an album to be private.

To add images to your new album, click the **Edit** button from the list of albums. This will pull up a detailed view of the album (shown here in Figure 18).



**Figure 18. Setting details information (click for larger image)**

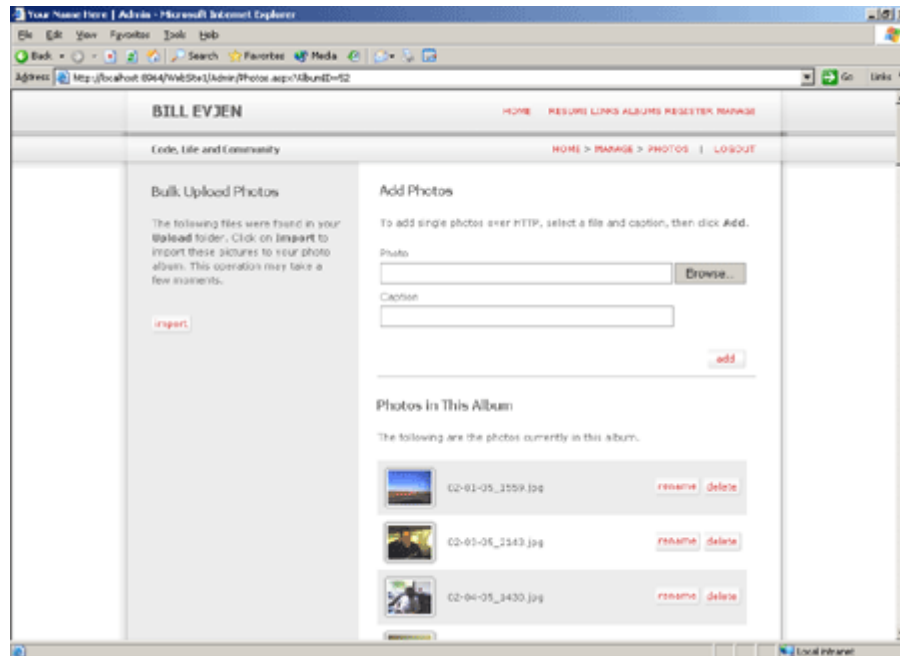
From this page, you can add a new image to the album as well as provide a name for the image that will be displayed to the viewer. After these values are provided, simply click the **Add** button and you will see the image listed in the album.



**Figure 19. Adding photos**

Not only can you add single images one at a time in this manner, but you can also add numerous images at once by copying the images to the Uploads folder that you will find in the application.

Once you have copied all the images you are interested in placing in the album in the Uploads folder, go back to the manager view of the album and click the **Import** button from the left column. This will move the images to the database for storage. After the images are moved to the database, you will see the images listed in the album page as shown in Figure 20.



**Figure 20. After adding new images**

Note that after you have moved the images from the Uploads folder to the database, you should then delete the images you placed in the Uploads folder, since clicking the **Import** button again will simply upload the images for a second time and you will have two copies of each image in your folder.

After the images are uploaded to the database and are included in your album, if you used the bulk upload you will notice that the name used for each image is simply the name of the file. You may wish to rename each of the images to something a little more meaningful. To accomplish this task, simply click the **Rename** button next to the photo and you will then be presented with a text box that enables you to rename the file. This is shown in Figure 21.



## Photos in This Album

The following are the photos currently in this album.




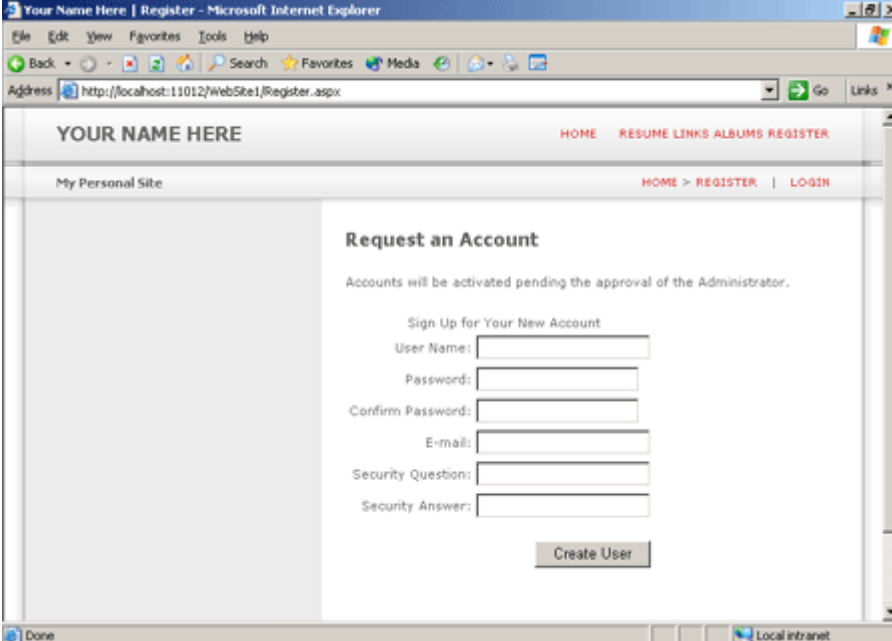
	<input type="text" value="02-01-05_1559.jpg"/>	<input type="button" value="save"/> <input type="button" value="cancel"/>
	02-03-05_2143.jpg	<input type="button" value="rename"/> <input type="button" value="delete"/>
	02-04-05_1430.jpg	<input type="button" value="rename"/> <input type="button" value="delete"/>

Figure 21. Renaming a photo

## Registering Users: Register.aspx

Clicking on the Register link in the navigation of the application, viewers will be directed to a page that will allow them to register for the application. This page can also be accessed when a viewer clicks the **Create Account** button on the home page. The Register.aspx page is presented in Figure 22.



YOUR NAME HERE

HOME RESUME LINKS ALBUMS REGISTER

My Personal Site

HOME > REGISTER | LOGIN

### Request an Account

Accounts will be activated pending the approval of the Administrator.

Sign Up for Your New Account

User Name:

Password:

Confirm Password:

E-mail:

Security Question:

Security Answer:

Create User

Figure 22. Requesting a new account

What are users actually registering for? The viewer who has registered will find that once the registration is complete they will not have any new access capabilities. Instead, they will first have to be activated for more privileges by a user in the Administrator role.

There are two roles in the Personal Web Site Starter Kit—Administrators and Friends—both of which warrant a little more definition.

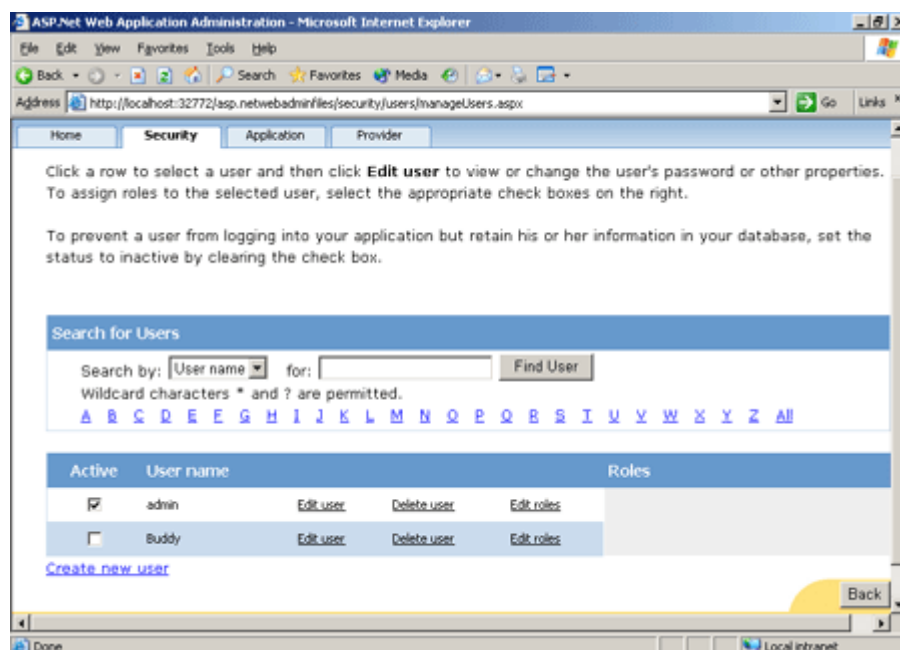
An individual who is considered an administrator will be able to manage the application. This means that a user in this role will be able to create new albums and photos, delete albums and photos, rename items, and decide whether any of these items are public or private. Administrators also have the ability to manage the application's registered users. Administrators can add and delete users from the Administrator role as well as add or delete users from the Friends role.

A user who is defined as being in the Friends role will not be able to manage any of the items that are mentioned above. Instead, users defined in the Friends role will be able to view the albums that are earmarked as private.

Albums that are earmarked as private are viewable only by users who are in the Administrators or Friends role. Unauthenticated users will not be able to view any albums that are not marked public.

When users register for your personal web site they will be placed in the list of users found in the ASP.NET Web Site Administration Tool. You can get to this tool by clicking the *ASP.NET Web Site Administration Tool* link from the Users and Settings section of the Manage page in the application. The other way to pull up this page is to click the appropriate button in the Visual Studio Solution Explorer or by selecting **Website**, then selecting **ASP.NET Configuration**.

When this configuration tool opens, select the **Security** tab. From the security page, select the *Manage users* link. You will then be presented with a list of registered users. An example of this is presented in Figure 23.



**Figure 23. Listing users**

From this page, you can edit or delete users. Right away, you can see whether the user you are working with is considered active in the application. *Active* means that their login (their defined username and password) will work in order to get authenticated and authorized for the application. Checking the check box next to the user will make them active. Clearing (unchecking) the checkbox will turn off the user's active status.

If someone has registered for your application, they are essentially asking to be considered to be in the Administrators or the Friends role. If you are interested in assigning them to one of these roles, click on the *Edit roles* link next to their name and select the appropriate role. If they are then considered an active user and in the role of Friends, the next time they log into the application, they will be able to see all the albums that are earmarked as private, which unauthenticated users will be unable to do.

## Summary

The Personal Web Site Starter Kit is both valuable and fun for a several reasons. First, it allows you to quickly and easily create a Web site that contains some personal information about yourself. Second and more importantly, this starter kit shows off some of the new capabilities found in ASP.NET 2.0 as well as some new features provided by the underlying .NET Framework 2.0.

From generics to new controls, there is a lot to learn from this application. Some of the more important items to pay attention to are the new membership and role management systems that ASP.NET 2.0 provides. These new security systems allow for you to easily manage access to your application and to not only authenticate users in a general fashion, but to also place the authenticated user in a particular role that will have different access rights.

The idea of this starter kit is not to use it as is, but instead to customize it heavily. You needn't limit yourself to just changing the *lorem ipsum* text, but instead you should look at how to add additional roles, privileges, new pages, and new capabilities. Explore further possibilities in my article [Extending the Personal Web Site Starter Kit](#). Have fun and happy coding!

## Related Books

- **Bill Evjen**, [ASP.NET 2.0 Beta Preview](#)

## About the author

Bill Evjen is an active proponent of .NET Framework technologies and community-based learning initiatives for .NET. He is a technical director for Reuters, the international news and financial services company ([www.reuters.com](http://www.reuters.com)) based in St. Louis, Missouri. Bill is the founder and executive director of INETA ([www.ineta.org](http://www.ineta.org)), the International .NET Association, which represents more than 100,000 members worldwide. Bill is also an author and speaker and has written such books as *ASP.NET Professional Secrets*, *XML Web Services for ASP.NET*, *Web Services Enhancements*, and the *Visual Basic .NET Bible* (all from John Wiley at <http://www.wiley.com/>).

[Manage Your Profile](#) | [Legal](#) | [Contact Us](#) | [MSDN Flash Newsletter](#)

© 2007 Microsoft Corporation. All rights reserved. [Terms of Use](#) | [Trademarks](#) | [Privacy Statement](#)

# Extending the Personal Web Site Starter Kit

Bill Evjen

Reuters

April 2005

Applies to:

Microsoft ASP.NET 2.0

Visual Studio 2005

Visual Web Developer 2005 Express Edition

SQL Server Express Edition

**Summary:** Learn how to extend the new Personal Web Site Starter Kit, which is an available project type in Visual Web Developer 2005 Express Edition. (21 printed pages)

## Contents

[Introduction](#)

[First, Personalize Your Site](#)

[Extending the Administration Abilities](#)

[Conclusion](#)

[Related Books](#)

[About the Author](#)

## Introduction

Inside Visual Studio 2005 and Visual Web Developer 2005 Express Edition, you will find an ASP.NET project type labeled as *Personal Web Site Starter Kit*. This project type provides you with a basic Web site to modify as your own. I provide an overview of the starter kit in my article [Introducing the Personal Web Site Starter Kit](#). If you need an introduction to the starter kit, then I recommend that you review this previous article first.

The Personal Web Site Starter Kit includes a home page, a resume page, a page that allows you to list your favorite links, and a photo album. The main piece of the starter kit is focused on the photo album and the administrator pages built around this functionality, which allows you to upload, modify, and delete photos as well as entire albums.

The starter kit is a great example of how to use some of the new and exciting features provided by ASP.NET 2.0 in an application. This starter kit utilizes master pages (the ability to provide your application with a master template), new configuration capabilities, XML definitions of the site structure (.sitemap files), and more.

This article will take a look at how to extend some of these features, which is the explicit purpose of the starter kit. It is called a starter kit for a reason! You should consider this starter kit as a starting point to use and build a truly unique application for yourself. In the end, you might not even end up with what is referred to as a personal Web site; but instead, it might be something else entirely.

## **First, Personalize Your Site**


The first thing you will want to do is to go through the site and completely personalize the pages you are going to keep in the application. This means changes to the master page and any of the content pages contained within the application.

Besides that, one important item you should be aware of for this application is that it doesn't take advantage of any sort of .NET caching capabilities. This means that each page is not stored in the system cache to reuse in order to save the cost of regenerating the page.

In most cases, you are most likely going to be presenting data on your personal Web site that isn't going to be changing frequently. Therefore, to get better performance from your application, you can specify that your ASP.NET pages should be cached for a specific period of time.

However, since this application makes use of the new capability of using master pages and content pages, there are now two pages that are combined when invoked to create a single ASP.NET page. Since this is the case, where exactly would you apply the caching?

When using master pages, you would apply caching to the content page, not to the master page. When using the `OutputCache` page directive, you need to place this in the content page for your page for any caching to occur. This is shown here:

 [Copy Code](#)

```
<%@ OutputCache Duration="60" VaryByParam="None" %>
```

Placing this page directive in any of the content pages of your application will cause both the contents of the content page and the associated master page to be cached by the system (remember that it is a single page at this point) for 60 seconds.

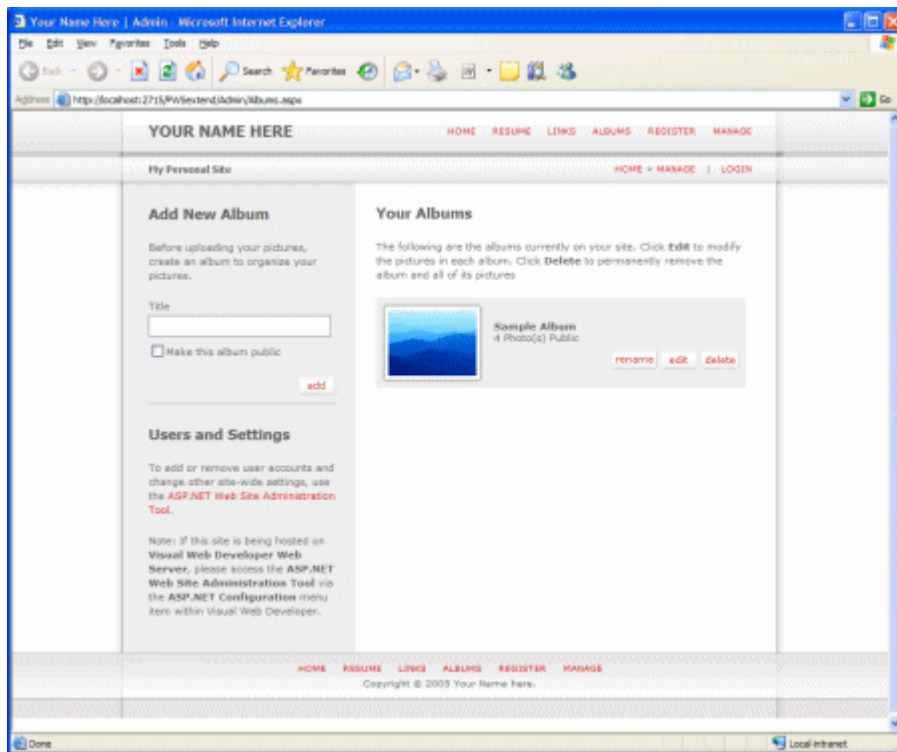
## Extending the Administration Abilities

The rest of this article will look at extending various parts of the application by examining the administrator's abilities to remotely manage the content of the application.

Presently, if you are logged in as an administrator for the application, you will have the ability to create, delete, or modify photo albums that are presented through the application. The rest of the application's content, mainly the textual content and the provided "cool links," are just hard-coded text contained in the content pages themselves. We will take a look at placing some of these items in a SQL Express database as well as making this content remotely manageable through the browser, as are the photo albums.

### Adding a Page to the Application

The first step that we will take is to add some additional pages to the overall application. Presently, when you log into the application as an administrator and click on the Manage link, you are brought to the `Albums.aspx` page shown here in Figure 1.



**Figure 1. Albums administration page**

As you can see, the main administration page allows you to create and manage photo albums as well the ability to add and modify the user database, which really is simply a link to the `webadmin.axd` http handler (this will run only for the local user).

In order to add the ability for the site administrator to manage some site content from this page in addition to the photo albums presented, we are going to have to make some changes here. You could just add these new management capabilities to the page presented here in Figure 1, but instead of that, let's add a couple of additional management pages just to make it a little more interesting.

For our example, let's cause the Manage link to go to a simple page that allows for further navigation to manage the photo albums, another page to manage users, and a third page to manage site content.

To create these new pages, let's first make the necessary changes to the `web.sitemap` file. The original `web.sitemap` file is presented in Listing 1.

#### **Listing 1: The original web.sitemap file**

 [Copy Code](#)

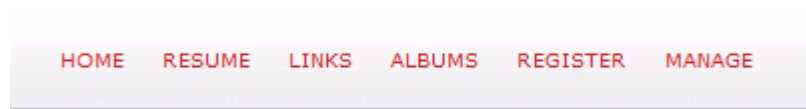


```

<?xml version="1.0" encoding="utf-8" ?>
<siteMap>
  <siteMapNode title="Home" url="Default.aspx">
    <siteMapNode title="Resume" url="Resume.aspx" />
    <siteMapNode title="Links" url="Links.aspx" />
    <siteMapNode title="Albums" url="Albums.aspx" >
      <siteMapNode title="Photos" url="Photos.aspx" >
        <siteMapNode title="Details" url="Details.aspx" />
      </siteMapNode>
    </siteMapNode>
  </siteMapNode>
  <siteMapNode title="Register" url="Register.aspx" />
  <siteMapNode title="Manage" url="Admin/Albums.aspx" >
    <siteMapNode title="Photos" url="Admin/Photos.aspx" >
      <siteMapNode title="Details" url="Admin/Details.aspx" />
    </siteMapNode>
  </siteMapNode>
</siteMap>

```


A sitemap is an XML description of your application's structure. This definition is then used by various site navigation server controls that are made available to you. For instance, the Personal Web Site Starter Kit uses the `Menu` server control at the top of the application in a horizontal view. This control simply reads the content of the `web.sitemap` file through a `SiteMapDataSource` server control in order to generate its results. This menu is presented here in Figure 2.



**Figure 2. The new menu**

If you look at the original `.sitemap` file shown in Listing 1, you can see that the `Manage` link is defined to use the URL `Admin/Albums.aspx`. For our example, let's change around the `Admin` section so that we can break out some of the management capabilities onto separate pages. In the end, you should have a `web.sitemap` file as presented here in Listing 2.

**Listing 2: The modified web.sitemap**

 [Copy Code](#)

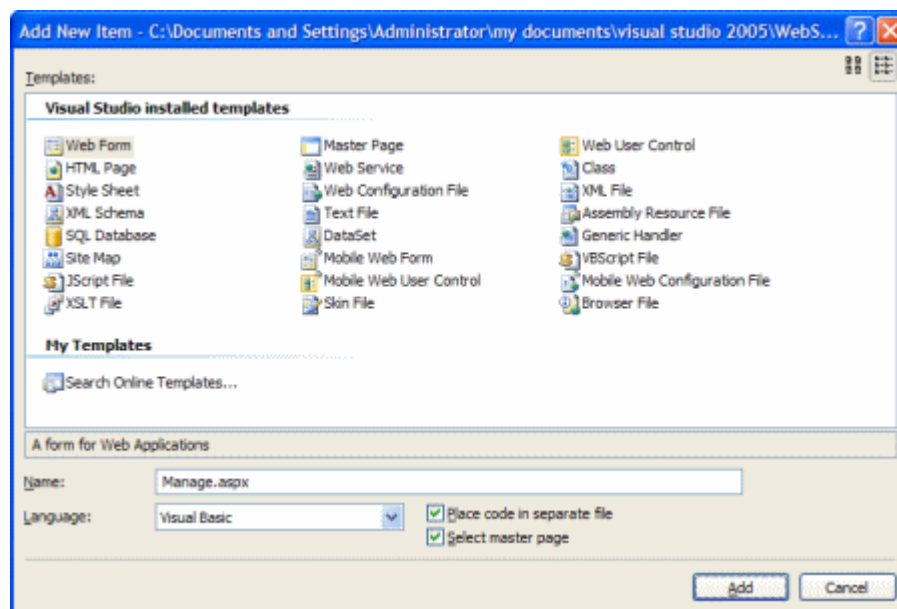
```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap>
  <siteMapNode title="Home" url="Default.aspx">
    <siteMapNode title="Resume" url="Resume.aspx" />
    <siteMapNode title="Links" url="Links.aspx" />
    <siteMapNode title="Albums" url="Albums.aspx" >
      <siteMapNode title="Photos" url="Photos.aspx" >
        <siteMapNode title="Details" url="Details.aspx" />
      </siteMapNode>
    </siteMapNode>
    <siteMapNode title="Register" url="Register.aspx" />
    <siteMapNode title="Manage" url="Admin/Manage.aspx" >
      <siteMapNode title="Albums" url="Admin/Albums.aspx" >
        <siteMapNode title="Photos" url="Admin/Photos.aspx" >
          <siteMapNode title="Details" url="Admin/Details.aspx"
        />
        </siteMapNode>
      <siteMapNode
        title="Site Content"
url="Admin/Content.aspx" />
      <siteMapNode title="Manage Users" url="Admin/Users.aspx"
    />
    </siteMapNode>
  </siteMapNode>
</siteMap>
```

From this example, you can see that adding additional pages to the application's navigation system is simply a matter of adding some additional XML nodes to the `web.sitemap` document. In this case, we changed the destination of the Manage link (it now points to `Admin/Manage.aspx`) and

we added two new pages: `Content.aspx` and `Users.aspx`. Now let's look at creating some of these pages.

### Manage.aspx

Let's create a new page for the Manage link in the navigation system: `Manage.aspx`. In creating `Manage.aspx`, remember that we are going to want to create this ASP.NET page so that it is located in the Admin folder (as seen in the Visual Studio Solution Explorer). To do this, right-click the Admin folder in the Solution Explorer and select **Add New Item**. This will pull up the **Add New Item** dialog box, which will allow you to select a new Web form. This is shown here in Figure 3.



**Figure 3. Adding a new content page**

In this dialog box, note that we are interested in creating a *content page*. A content page works with a master page and we have specified this by selecting the **Select master page** check box. Doing this and clicking **Add** will pull up another dialog that allows you to select the master page to use for this particular content page. In this case, there is only one master page to choose from: `Default.master`.

This content page—`Manage.aspx`—will simply contain three image buttons. The code for this page is presented here in Listing 3.

### Listing 3: `Manage.aspx`

```
<%@ Page Language="VB" MasterPageFile="~/Default.master"
    AutoEventWireup="false" CodeFile="Manage.aspx.vb"
    Inherits="Manage"
    title="Manage Your Personal Web Site" %>

<asp:Content ID="Content1" ContentPlaceHolderID="Main"
    Runat="Server">

    <div class="shim column"></div>

    <div class="page" id="admin-albums">

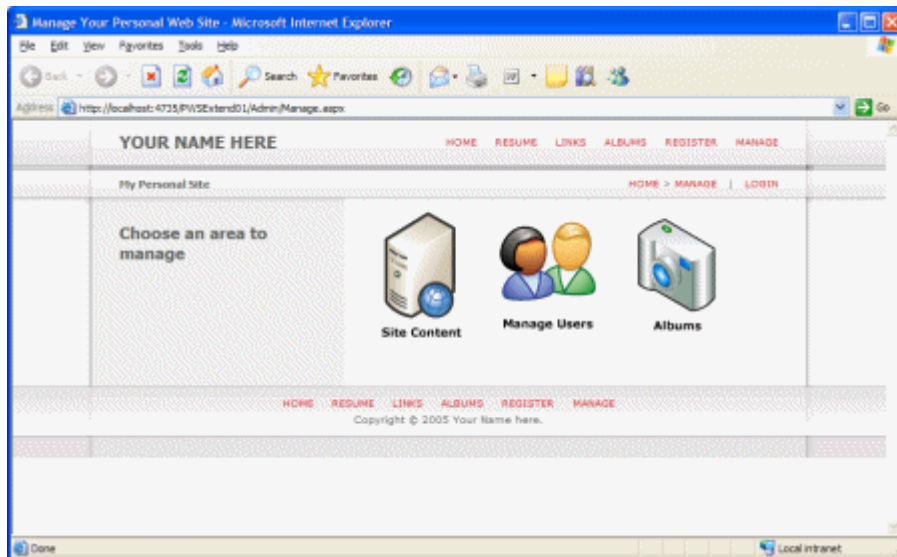
        <div id="sidebar">
            <h3>Choose an area to manage</h3>
        </div>

        <div id="content">
            <table width="90%"><tr>
                <td><asp:ImageButton ID="ImageButton1" runat="server"
                    ImageUrl=".. /Images/button_sitecontent.gif"
                    PostBackUrl="Content.aspx" /></td>
                <td><asp:ImageButton ID="ImageButton2" runat="server"
                    ImageUrl=".. /Images/button_manageusers.gif"
                    PostBackUrl="Users.aspx" /></td>
                <td><asp:ImageButton ID="ImageButton3" runat="server"
                    ImageUrl=".. /Images/button_albums.gif"
                    PostBackUrl="Albums.aspx" /></td>
            </tr></table>
        </div>

    </div>
```

```
</asp: Content >
```

Since this is a content page, you can see that it points to the master page to use the @Page directive's `MasterPageFile` attribute. Also, this page is rather simple. It uses the same CSS classes as the `Albums.aspx` page found in the Admin folder and simply contains a table holding three image buttons that each point to a different administration page. Running this page in the browser produces the graphic shown here in Figure 4.



**Figure 4. The new Administration home page**

Now that the `Manage.aspx` page is in place, the next step we will take is to create the `Users.aspx` page. I explain how in the next section.


### **Users.aspx**

This page will be rather simple to create. If you look back at Figure 1, you will notice that there is a small section on the page that is dedicated to allowing you to manage your users and the user's settings (shown in the lower left-hand corner of the page). Since we want to separate the administration functionality a bit, we will end up pulling this section from the `Albums.aspx` page and then placing it in the `Users.aspx` page instead.

The first step is to create the `Users.aspx` page. Just like the `Manage.aspx` page, create the `Users.aspx` page inside of the Admin folder and make sure that it is a content page using the `Default.master` page as its master template.

The code to use for `Users.aspx` is shown here in Listing 4.

#### Listing 4: Users.aspx

 [Copy Code](#)

```
<%@ Page Language="VB" MasterPageFile="~/Default.master"
    AutoEventWireup="false" CodeFile="Users.aspx.vb"
    Inherits="Users"
    title="Manage Users" %>

<asp:Content ID="Content1" ContentPlaceHolderID="Main"
    Runat="Server">

    <div class="shim column"></div>

    <div class="page" id="admin-albums">

        <div id="sidebar">
            <h3>Users and Settings</h3>
        </div>

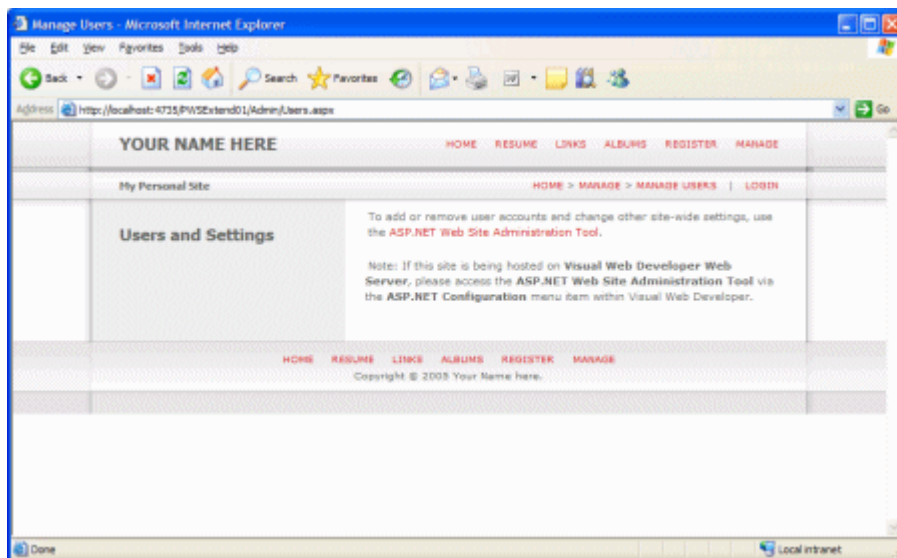
        <div id="content">
            <p>To add or remove user accounts and change other site-wide
                settings, use the <a href="webadmin.axd">ASP.NET Web Site
                Administration Tool</a>. </p>
            <p>Note: If this site is being hosted on <b>Visual Web
                Developer
                Web Server</b>, please
                access the <b>ASP.NET Web Site Administration Tool</b> via
                the
                <b>ASP.NET Configuration</b>
                menu item within Visual Web Developer. </p>
        </div>
```

</di v>

</asp: Content>

This page is very much like the `Manage.aspx` page. It uses the same CSS styling and has some simple text and a link to the `webadmin.axd` http handler. Again, it is important to note that you will be able to invoke this http handler only as a local user on the server.

Figure 5 illustrates what is produced by the `Users.aspx` page.



**Figure 5. The new User Administration page**

The next and final step is to create an administration page that will allow us to manage the content of the home page.

## Building a Simple Content Management System

If you look at the home page for the Personal Web Site Starter Kit, you will notice that much of the text it contains is hard-coded in the actual page. Though this works, let's look at changing this structure so that the content is retrieved from a SQL Server 2005 Express database instead. We will also look at allowing the site administrator (someone under the Administrators role) to update this information directly from an admin page that we will later create (`Content.aspx`).

The first step we will take to achieve this is to modify the SQL Express file, `Personal.mdf`. We will use `Personal.mdf` to store the home page content in addition to the photo album

information for which it is already used. It is important to note that this is just one example. You could also build this content management system to use an XML file instead of the SQL Express file.

### Modifying the Personal.mdf File

From the Visual Web Developer Solution Explorer, expand the App\_Data folder. You will notice that there are two SQL Server Express files in this folder, ASPNETDB.mdf and Personal.mdf.

ASPNETDB.mdf is used by ASP.NET to store values for the various systems that are part of ASP.NET such as the new membership and role management systems. All the application roles and the user's credentials are stored in this database file.

The Personal.mdf file is used to store the photo album information and even the photos themselves (they are contained within the Photos table). Our first step is to add a new table to this database. This new table is where we will store the text that will appear on the home page of the Personal Web Site Starter Kit.

To create a new table in the SQL Server Express file, choose the Database Explorer tab in Visual Web Developer and expand the Data Connections root node. Expand the Tables folder for the Personal.mdf database presented by expanding Personal.mdf, then Tables in the Database Explorer. You will see that this database contains two tables called Albums and Photos. To add an additional table, right-click the Tables folder and select **Add New Table** from the list of options.

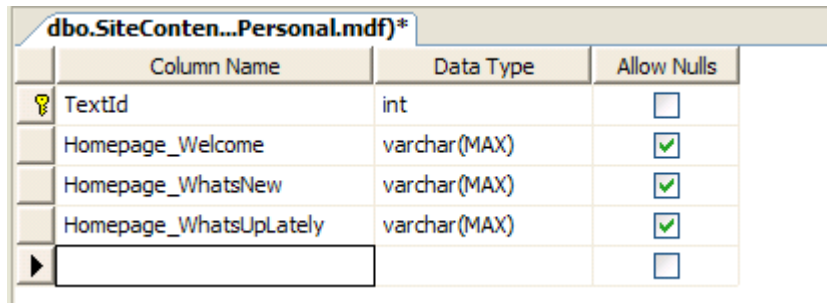
In this table, we are interested in adding a few simple columns. Following is a list of columns and their associated data types to add to the table:

Column Name	Data Type
<a href="#">Copy Code</a> TextId	<a href="#">Copy Code</a> int
<a href="#">Copy Code</a> Homepage_Welcome	<a href="#">Copy Code</a> varchar (MAX)
<a href="#">Copy Code</a> Homepage_WhatsNew	<a href="#">Copy Code</a> varchar (MAX)
<a href="#">Copy Code</a> Homepage_WhatsUpLate ly	<a href="#">Copy Code</a> varchar (MAX)



After these columns are in place, right-click the `TextId` column and set this column to be a primary key. Doing this will allow us to later update the other columns as the update will require a way to identify the row that needs updating (even though we are using just one row in the table).

In the end, your table definition should appear as shown in Figure 6.

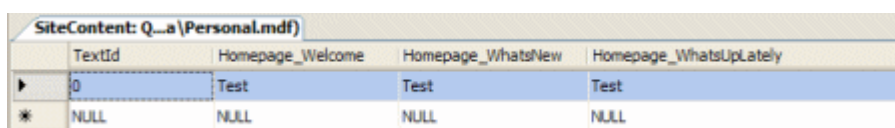


	Column Name	Data Type	Allow Nulls
🔑	TextId	int	<input type="checkbox"/>
	Homepage_Welcome	varchar(MAX)	<input checked="" type="checkbox"/>
	Homepage_WhatsNew	varchar(MAX)	<input checked="" type="checkbox"/>
	Homepage_WhatsUpLately	varchar(MAX)	<input checked="" type="checkbox"/>
▶			<input type="checkbox"/>

**Figure 6. Table definition**

Next, save the table and you will then be presented with a dialog box that asks you to name the table. Name the table `SiteContent`. After the table is named and saved, it will then appear in the Database Explorer with the other two tables. Next, right-click the newly created table `SiteContent` and select **Show Table Data** from the list of options. This will show a list of columns plus the data contained within each (which will be just NULL values at this point).

From here, you can enter in the content of your page, but since we will be building a page to do this later, let's just enter a numeric 0 in the `TextId` column and the string "Test" in each of the other columns. This is illustrated here in Figure 7.



	TextId	Homepage_Welcome	Homepage_WhatsNew	Homepage_WhatsUpLately
▶	0	Test	Test	Test
*	NULL	NULL	NULL	NULL

**Figure 7. Adding content to the new table**


Now that the `Personal.mdf` file contains the additional table that will drive the home page of the application, let's next re-code the home page to work from this database table.

### Re-coding the `Default.aspx` page to work from the `Personal.mdf` file

Pull up the `Default.aspx` page in Visual Studio and scroll to the bottom of the page. Here at the bottom is an `ObjectDataSource` control. This control drives the photo that appears on the sidebar of the page. ASP.NET 2.0 provides a number of new data controls that work to pull data


from different data sources as well as perform operations such as inserting, deleting, and updating data in these data stores. Looking in the Toolbox of Visual Studio, you will notice that in addition to the `ObjectDataSource` control used in the Personal Web Site Starter Kit, there are also data source controls for working with SQL databases, Microsoft Access, XML sources, and sitemaps.

The `ObjectDataSource` control used on the `Default.aspx` page is shown here:

 [Copy Code](#)

```
<asp: ObjectDataSource ID="ObjectDataSource1" Runat="server"
    TypeName="PhotoManager" SelectMethod="GetPhotos">
</asp: ObjectDataSource>
```

You would use the `ObjectDataSource` control when you are interested in using a traditional three-tiered model when it comes to working with data that is presented on your page. From this example, you can see that the photos are retrieved through the class `PhotoManager` (specified by the `TypeName` attribute) using the `GetPhotos` method (specified by the `SelectMethod` attribute). You will find the `PhotoManager.vb` or `.cs` class file in the `App_Code` folder of your project. Contained within this class, you will then find the `GetPhotos` method as shown here (in Visual Basic):

 [Copy Code](#)

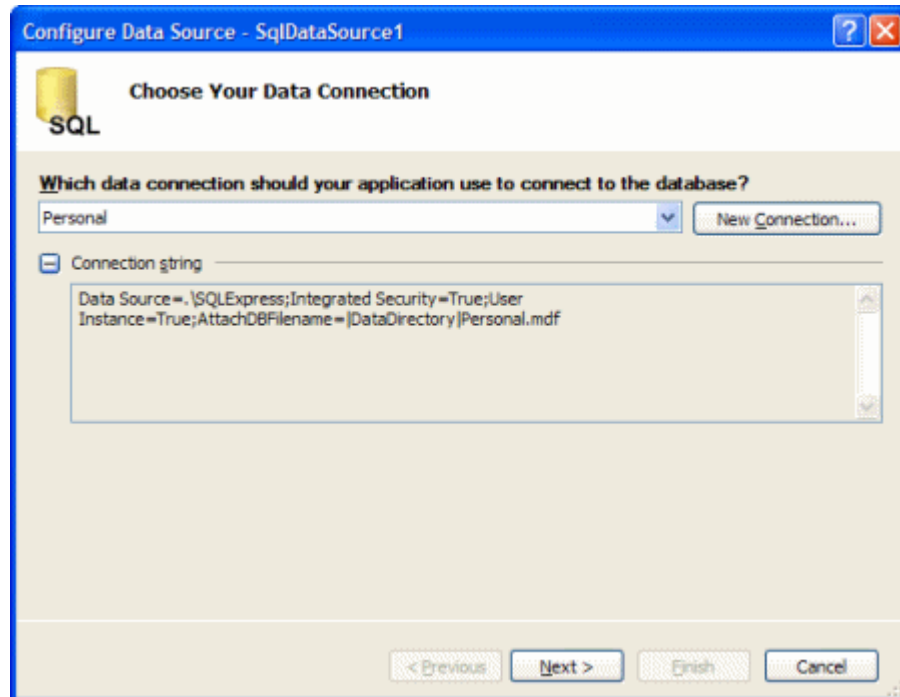
```
Public Function GetPhotos() As Generic.List(Of Photo)
    Return GetPhotos(GetRandomAlbumID())
End Function
```

In changing the `Default.aspx` page so that it will now display textual content as retrieved from the `Personal.mdf` file, you could certainly add another class to the `App_Code` folder that deals with getting this information and then use another `ObjectDataSource` control to invoke a `Select` method in that class. Constructing the site content retrieval in this manner will allow you to extend the data model that the application is already using. However, for the sake of learning, let's instead look at using a `SqlDataSource` control to pull the site content data.

### Setting up the `SqlDataSource` control

To set up a `SqlDataSource` control that will be used to pull the site content data from the `Personal.mdf` file, open up the `Default.aspx` page in Visual Studio in the Design mode (the Design tab is at the bottom of the document window). Once the page is open in the Design

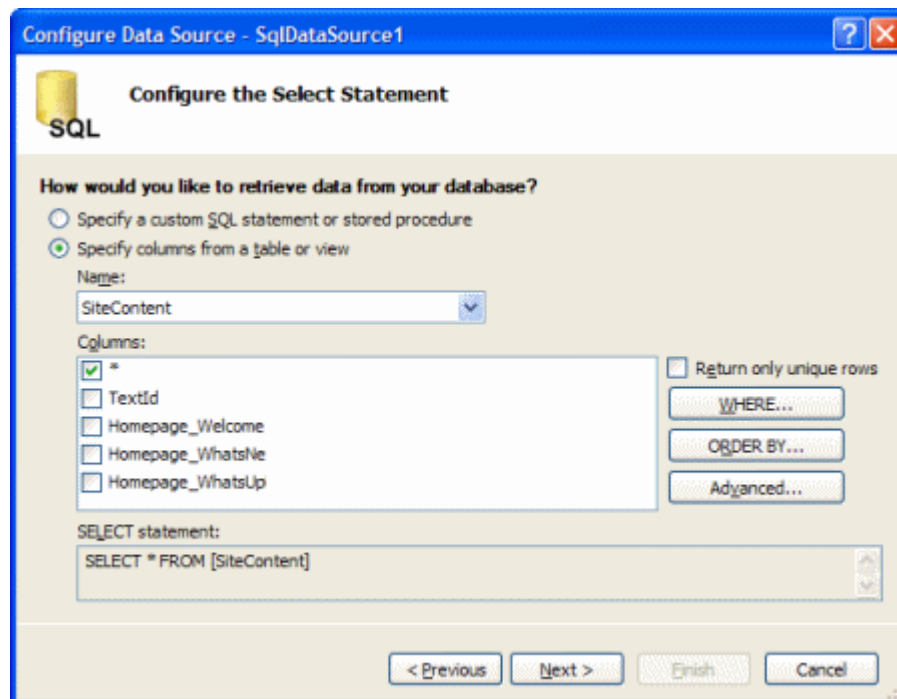
mode, drag a `SqlDataSource` control onto the design surface so that it is positioned next to the `ObjectDataSource` control at the bottom of the page. Highlighting the `SqlDataSource` control and clicking the green arrow that will appear will allow you to open the control's smart tag. From here, you will then be able to click on the [Configure Data Source](#) link. This will launch a wizard that will allow you to configure the `SqlDataSource` control to get at the site content data found in the `Personal.mdf` file. The first page of this wizard is shown in Figure 8.



**Figure 8. Configuring a Data Source Wizard**

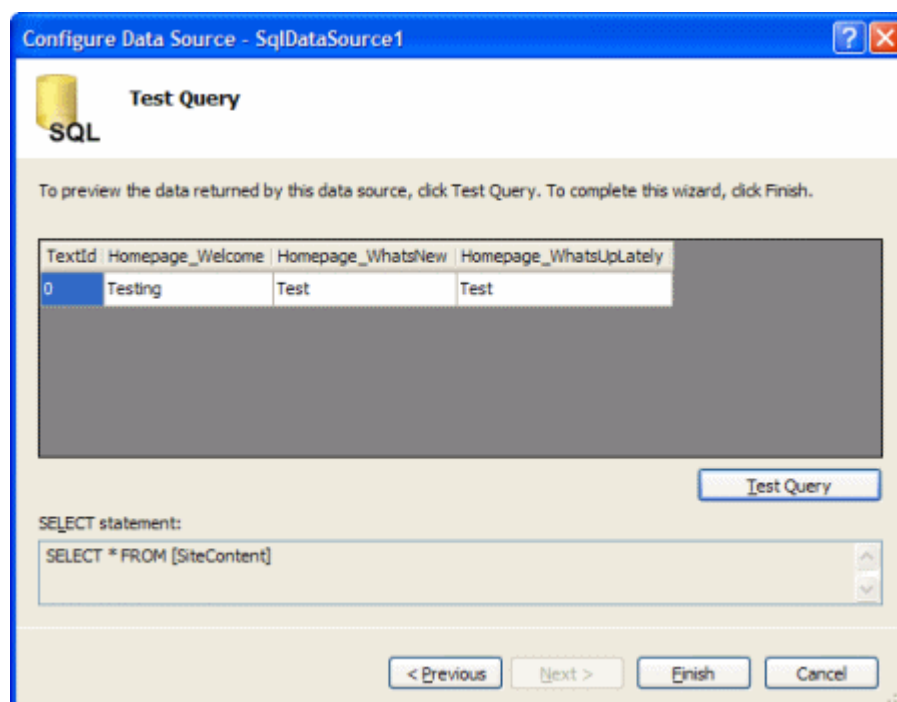
As the wizard asks you to configure the `SqlDataSource` control, you can see that you first need to establish a connection to the `Personal.mdf` file. Since there is already a defined connection in the `web.config` file, you will see the option of **Personal** in the drop-down list. Select **Personal** and then click the **Next** button.

The next screen in the wizard allows you to specify the table you want to work with. In this case, select `SiteContent` from the drop-down list. Since we want to deal with everything contained in this table, select the asterisk (\*), a wildcard character that indicates that you are interested in everything this table offers. This is shown here in Figure 9.



**Figure 9. Configuring a Data Source: Selecting the table**


You are now ready to move onto the next screen in the wizard, which allows you to test the connection to the `Personal.mdf` database you just created. Clicking the **Test Query** button should produce the results illustrated here in Figure 10.



**Figure 10. Configuring a Data Source: Previewing the Data**

When you are done, click **Finish**. Listing 5 shows the code of the `SqlDataSource` control that Visual Studio will generate from the instructions provided to the wizard.

**Listing 5: The `SqlDataSource` control code as generated by Visual Studio**

 [Copy Code](#)


```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="<%"$ ConnectionStrings:Personal %">
    SelectCommand="SELECT * FROM [SiteContent]">
</asp:SqlDataSource>
```

Now that the `SqlDataSource` control is in place, let's move on to the next step of modifying the `Default.aspx` page to show the content that will come from this `datasource` control.

**Modifying the `Default.aspx` page to show the content coming from the `SqlDataSource` control**

Looking at the source code for the `Default.aspx` page, you will notice that the page is a content page that uses the only master page available to it in the application: `Default.master`. This master page exposes a single area for the content page to use. This area is then defined in the content page through the use of a single `Content` server control located on the page. Inside of the `Content` server control, you will see that the content page is divided up into a few distinct sections, all of which are defined by `<div>` elements. A framework of this is shown here in Listing 6.

**Listing 6: Looking at the structure of the `Default.aspx` page**

 [Copy Code](#)

```
<%@ Page Language="VB" MasterPageFile="~/Default.master"
    Title="Your Name Here | Home"
    CodeFile="Default.aspx.vb" Inherits="Default.aspx" %>

<asp:Content ID="Content1" ContentPlaceHolderID="Main"
    runat="server">

    <div class="shim column"></div>
```

```

<div class="page" id="home">
  <div id="sidebar">
    <!-- Content removed for clarity -->
  </div>
  <div id="content">
    <!-- Content removed for clarity -->
  </div>
</div>
</asp:content>

```

From this code framework, you can see that the actual page is divided up into two sections: the sidebar content (defined using `<div id="sidebar">`) and the main part of the page (using `<div id="content">`). Since we are interested in driving the content contained in the `<div id="content">` section of the page, let's focus on that part of the page. To accomplish this, we are going to place a `DataList` server control in this section of the page and pull out the values we need using `Eval` statements. This is illustrated in Listing 7.

**Listing 7: Using a DataList control to drive the content on the page**

```

<div id="content">
  <asp:DataList ID="DataList1" runat="server"
    DataSourceID="SqlDataSource1">
    <ItemTemplate>
      <h3>Welcome to My Site</h3>
      <p><%# Eval("Homepage_Welcome") %></p>
      <hr />
      <div id="whatsnew">
        <h4>What's New</h4>
        <p><%# Eval("Homepage_WhatsNew") %></p>
      </div>
      <div id="coollinks">
        <h4>Cool Links</h4>

```

 [Copy Code](#)

```

        <ul class="link">
        <li><a href="#">Lorem ipsum dolositi onr</a></li>
        <li><a href="#">Lorem ipsum dolositi onr</a></li>
        <li><a href="#">Lorem ipsum dolositi onr</a></li>
        <li><a href="#">Lorem ipsum dolositi onr</a></li>
        <li><a href="#">Lorem ipsum dolositi onr</a></li>
        </ul>

    </div>

    <hr />

    <h4>What' s Up Lately </h4>

    <p><%# Eval ("Homepage_WhatsUpLately") %></p>

</ItemTemplate>

</asp: DataLi st>
</div>

```

From the example illustrated here in Listing 7, you can see that the `DataList` server control binds itself to the `SqlDataSource1` control by using the `DataSourceId` attribute.

```

<asp: DataLi st                                ID="DataLi st1"
runat="server" DataSourceID="SqlDataSource1">
</asp:DataList>

```

[Copy Code](#)

The `DataList` control provides a number of template types that can be used for display purposes. Examples include a header template, footer template, item template, separator template, and more. Since we are interested in displaying a single row from the `SiteContent` table, we therefore need to employ only the `<ItemTemplate>` node and nothing more. When the `DataList` control is then rendered, it will display the contents of the `<ItemTemplate>` section once for each row contained in the table in which it is bound.

Inside the `<ItemTemplate>` section, we can then get at the contents for a particular column of the `SiteContent` table through the use of a data-binding expression `Eval`. In ASP.NET 2.0, the data binding has been simplified and can now be as simple as:

```

<%# Eval ("Homepage_Welcome") %>

```

[Copy Code](#)

This statement will cause the contents contained in the table column Homepage\_Welcome to be displayed in its place.

Save the Default.aspx page and run it in the browser. You should now see the results as shown in Figure 11.

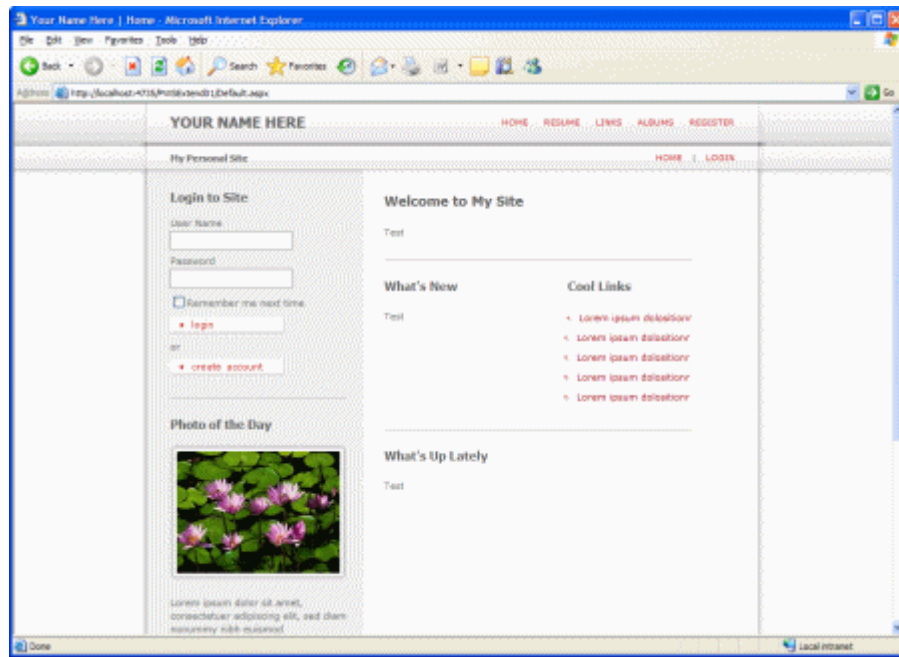


Figure 11. The new dynamic home page

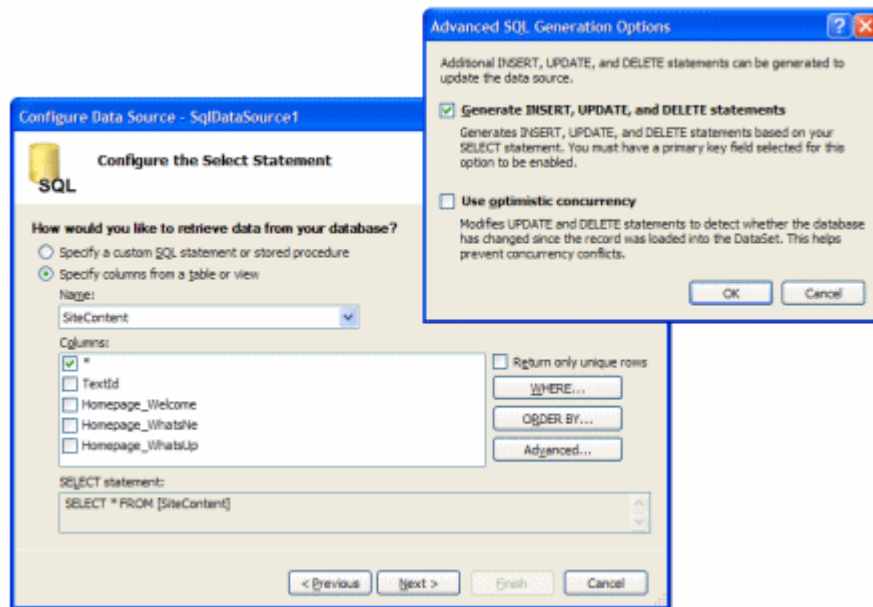
### Creating an administrator page to update the page content

Our final step is to create a page for the Site Content link of the Manage.aspx page. This new page will allow administrators to update the content that appears on the Default.aspx page. For this step, create a new content page called Content.aspx that uses the Default.master page as a template.

This page will require you to place a `SqlDataSource` control on it. The configuration of this `SqlDataSource` control will be a bit different than that of the Default.aspx page. Since we are going to be also dealing with the updating of the data on this administration page, we are therefore interested in having this `SqlDataSource` control perform the proper updates to the data contained in the SiteContent table. Therefore, as you work through the `SqlDataSource` control's configuration wizard, when you are working from the Select statement page of the wizard, click the **Advanced** button in the dialog box and select the first check box to build



a `SqlDataSource` control that will allow for the updating of the data. These dialogs are shown here in Figure 12.



**Figure 12. Automatically generating SQL statements**

After working through the wizard, your `SqlDataSource` control should appear as shown in Listing 8.

**Listing 8: The `SqlDataSource` control code as generated by Visual Studio**

[Copy Code](#)

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="<%$ ConnectionStrings:Personal %>"
    DeleteCommand="DELETE FROM [SiteContent] WHERE [TextId] =
        @original_TextId"
    InsertCommand="INSERT INTO [SiteContent] ([TextId],
        [Homepage_Welcome], [Homepage_WhatsNew],
        [Homepage_WhatsUpLately])
        VALUES (@TextId, @Homepage_Welcome, @Homepage_WhatsNew,
        @Homepage_WhatsUpLately) "
    SelectCommand="SELECT * FROM [SiteContent] "
    UpdateCommand="UPDATE [SiteContent] SET [Homepage_Welcome] =
        @Homepage_Welcome, [Homepage_WhatsNew] = @Homepage_WhatsNew,
```

```

[Homepage_WhatsUpLatel y] = @Homepage_WhatsUpLatel y WHERE [TextId]
=
@ori gi nal _TextId">
    <DeleteParameters>
        <asp: Parameter Name="ori gi nal _TextId" Type="Int32" />
    </DeleteParameters>
    <UpdateParameters>
        <asp: Parameter Name="Homepage_Wel come" Type="Stri ng" />
        <asp: Parameter Name="Homepage_WhatsNew" Type="Stri ng" />
        <asp: Parameter
            Name="Homepage_WhatsUpLatel y"
Type="Stri ng" />
        <asp: Parameter Name="ori gi nal _TextId" Type="Int32" />
    </UpdateParameters>
    <InsertParameters>
        <asp: Parameter Name="TextId" Type="Int32" />
        <asp: Parameter Name="Homepage_Wel come" Type="Stri ng" />
        <asp: Parameter Name="Homepage_WhatsNew" Type="Stri ng" />
        <asp: Parameter
            Name="Homepage_WhatsUpLatel y"
Type="Stri ng" />
    </InsertParameters>
</asp: Sql DataSource>

```

Of course, we are interested only in reading (using the `Select` statement) and updating (using the `Update` statement) the data in the `SiteContent` table. Since we are not interested in deleting or inserting new rows of data into the `SiteContent` table, you can feel free to delete the `UpdateCommand` and `InsertCommand` attributes and their values from the `SqlDataSource` control. You can also feel free to delete the `<DeleteParameters>` and the `<InsertParameters>` sections as well.

Now that the `SqlDataSource` control is in place, let's add a `FormView` control to the page to drive all the displaying and updating of the content to the `SiteContent` table. Drag and drop a `FormView` control onto the `Content.aspx` page.



```

</asp: TextBox></p>
<asp: LinkButton ID="UpdateButton" runat="server"
CausesValidation="True" CommandName="Update" Text="Update">
</asp: LinkButton>
<asp: LinkButton ID="UpdateCancelButton" runat="server"
CausesValidation="False" CommandName="Cancel" Text="Cancel">
</asp: LinkButton>
</EditItemTemplate>
<ItemTemplate>
<p><b>TextId: </b>
<asp: Label ID="TextIdLabel" runat="server"
Text='<%# Eval("TextId") %>'></asp: Label></p>
<p><b>Homepage_Welcome: </b><br />
<asp: Label ID="Homepage_WelcomeLabel" runat="server"
Text='<%# Bind("Homepage_Welcome") %>'>
</asp: Label></p>
<p><b>Homepage_WhatsNew: </b><br />
<asp: Label ID="Homepage_WhatsNewLabel" runat="server"
Text='<%# Bind("Homepage_WhatsNew") %>'>
</asp: Label></p>
<p><b>Homepage_WhatsUpLately: </b><br />
<asp: Label ID="Homepage_WhatsUpLatelyLabel" runat="server"
Text='<%# Bind("Homepage_WhatsUpLately") %>'>
</asp: Label></p>
<asp: LinkButton ID="EditButton" runat="server"
CausesValidation="False" CommandName="Edit" Text="Edit">
</asp: LinkButton>
</ItemTemplate>
</asp: FormView>

```

From this code listing, you can see that we added a few break lines and changed the edit template so that instead of using standard `TextBox` controls, we are now using multilined textboxes.

Running these pages as an administrator will produce the results shown in Figure 13.

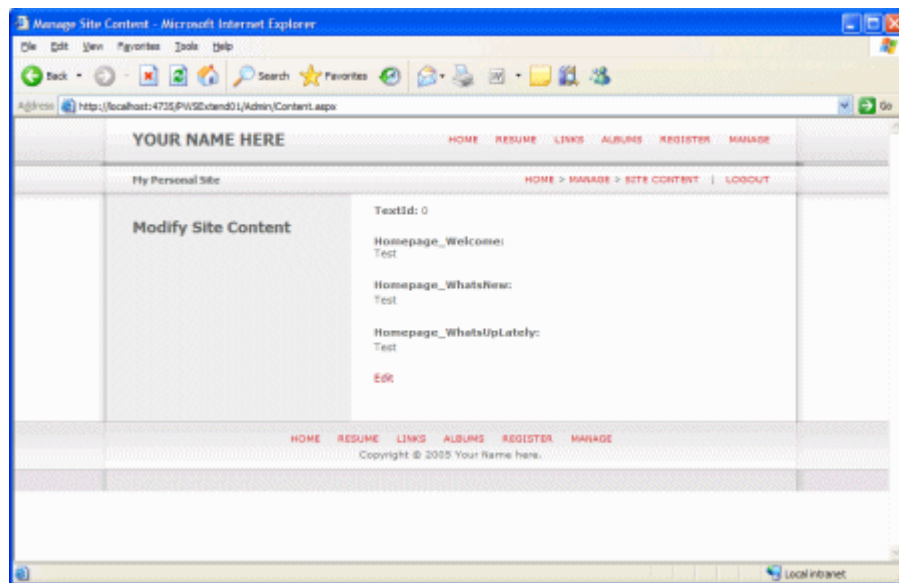


Figure 13. The edit content page

Clicking the **Edit** button will allow you to edit the contents of each of these sections. This is illustrated in Figure 14.

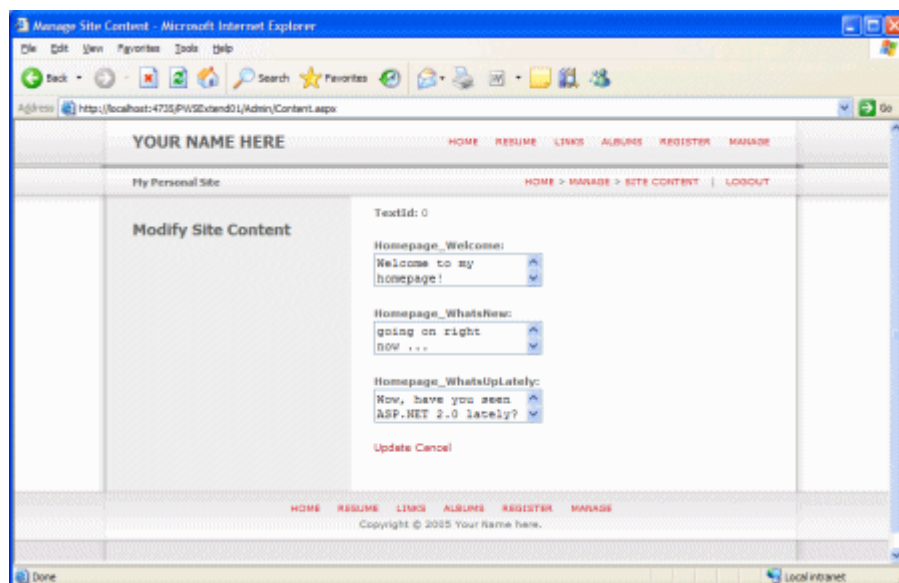


Figure 14. Changing the home page content

Now the information placed inside of these controls will be updated into the SiteContent table, which in turn will be output to the Default.aspx page. Now the content of the Default.aspx page is remotely manageable.

## Conclusion

This article showed you how to work with the foundation of the Personal Web Site Starter Kit and expand upon it for your own personal use. As you can see, it is rather easy to start incorporating your own features into this simple-to-use starter kit.

Have fun and happy coding!

## Related Books

- **Bill Evjen**, [ASP.NET 2.0 Beta Preview](#)

## About the Author

Bill Evjen is an active proponent of .NET Framework technologies and community-based learning initiatives for the .NET Framework. He is a technical director for Reuters, the international news and financial services company ([www.reuters.com](http://www.reuters.com)) based in St. Louis, Missouri. Bill is the founder and executive director of INETA ([www.ineta.org](http://www.ineta.org)), the International .NET Association, which represents more than 100,000 members worldwide. Bill is also an author and speaker and has written such books as *ASP.NET Professional Secrets*, *XML Web Services for ASP.NET*, *Web Services Enhancements*, and the *Visual Basic .NET Bible* (all from John Wiley at [www.wiley.com](http://www.wiley.com)).

[Manage Your Profile](#) | [Legal](#) | [Contact Us](#) | [MSDN Flash Newsletter](#)

© 2007 Microsoft Corporation. All rights reserved. [Terms of Use](#) | [Trademarks](#) | [Privacy Statement](#)