7

Navigation

WHAT YOU WILL LEARN IN THIS CHAPTER:

- How to move around in your site using server controls and plain HTML
- How to address pages and other resources like images
- How to use the ASP.NET `Menu`, `TreeView`, and `SiteMapPath` navigation controls
- How to send users from one page to another programmatically

WROX.COM CODE DOWNLOADS FOR THIS CHAPTER

You can find the wrox.com code downloads for this chapter on the Download Code tab at `www.wrox.com/remtitle.cgi?isbn=1118311809`. The code is in the Chapter 7 download.

When your site contains more than a handful of pages, it's important to have a solid and clear navigation structure that enables users to find their way around your site. If you implement a good navigation system, all the disconnected web pages in your project form a complete and coherent website.

When you think about important parts of a navigation system, the first thing that you may come up with is a menu. Menus come in all sorts and sizes, ranging from simple and static HTML links to complex, fold-out menus driven by CSS or JavaScript. But there's more to navigation than menus alone. ASP.NET comes with a number of useful navigation controls that enable you to set up a navigation system in no time. These controls include the `Menu`, `TreeView`, and `SiteMapPath`, which you learn about in this chapter.

Besides visual controls like `Menu`, navigation is also about *structure*. A well-organized site is easy for your users to navigate. The `Web.sitemap` file that is used by the navigation controls helps you define the logical structure of your site.

Another important part of navigation takes place at the server. Sending a user from one page to another in Code Behind based on some condition is a very common scenario. For example, imagine an administrator entering a new CD or concert review in the Management section of the website. When the review is completed, you may want to show the administrator the full details by redirecting her to a new page.

In this chapter, you learn how to use the different navigation options at your disposal. Before you look at the built-in navigation controls, however, you need to understand the different options you have to address the resources in your site, such as ASPX pages and images.

DIFFERENT WAYS TO MOVE AROUND YOUR SITE

The most common way to let a user move from one page to another is by using the `<a>` element. This element has an `href` attribute that enables you to define the address of a page or other resource you want to link to. Between the tags you can place the content you want to link, such as text, an image, or other HTML. The following snippet shows a simple example of the `<a>` element:

```
<a href="Login.aspx">You can log in here</a>
```

With this code in a web page, after users click the text "You can log in here," they are taken to the `Login.aspx` page, which should be in the same folder as the page that contains the link.

The `<a>` element has a server-side counterpart called the `HyperLink`. It eventually ends up as an `<a>` element in the page. The `NavigateUrl` property of this control maps directly to the `href` attribute of the `<a>` element. For example, a server-side `HyperLink` in a content page such as this:

```
<asp:HyperLink runat="server" id="LoginLink" NavigateUrl="Login.aspx">
       You can log in here</asp:HyperLink>
```

produces the following HTML in the browser:

```
<a id="LoginLink" href="Login.aspx">You can log in here</a>
```

Other than the ID that is assigned by the ASP.NET run time, this code is identical to the earlier example. In both cases, the `href` attribute points to the `Login.aspx` page using a relative URL. The next topic describes the differences between relative and absolute URLs.

Understanding Absolute and Relative URLs

Key to working with links in your site is a good understanding of the different forms a *uniform resource locator* (*URL*) to a resource inside or outside your website can take. A URL is used to uniquely identify a resource in your or another website. These URLs are used in different places, including the `href` attribute of a hyperlink or a `<link>` element to point to a CSS file, the `src` attribute pointing to an image or a JavaScript source file, and the `url()` value of a CSS property. A URL can be expressed as a *relative URL* or an *absolute URL*. Both have advantages and disadvantages that you should be aware of.

Relative URLs

In the previous examples you saw a relative URL that points to another resource relative to the location where the URL is used. This means that the page containing the `<a>` element and the `Login.aspx` page should both be placed in the same folder in your site. To refer to resources in other folders you can use the following URLs. All the examples are based on a site structure shown in Figure 7-1.

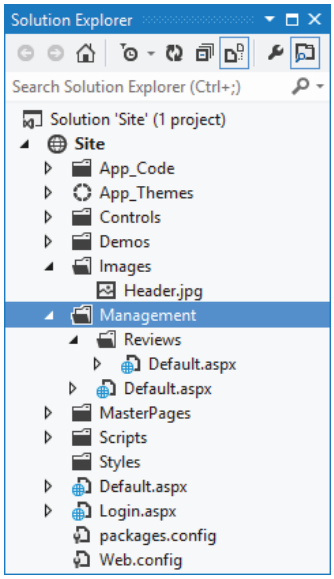

FIGURE 7-1

To link from `Login.aspx` in the root to `Default.aspx` in the `Management` folder, you can use this URL:

```
<a href="Management/Default.aspx">Management</a>
```

To refer to the image `Header.jpg` from `Default.aspx` in the `Management` folder, you can use this URL:

```
<img src="../Images/Header.jpg" />
```

The two leading periods "navigate" one folder up to the root, and then the path goes back in the `Images` folder to point to `Header.jpg`.

For a deeper folder hierarchy, you can use multiple double periods, one for each folder you want to go upward in the site hierarchy, like the following `<img>` element. You can use it to refer to the same image from pages in the `Reviews` folder, which is located under the `Management` folder:

```
<img src="../../Images/Header.jpg" />
```

One benefit of relative URLs is that you can move a set of files to another directory at the same level without breaking their internal links. However, at the same time, they make it more difficult to move files to a different level in the site hierarchy. For example, if you moved the `Login.aspx` page to a separate folder like `Members`, the link to the `Management` folder would break. The new `Members` folder doesn't have `Management` as its subfolder, so `Management/Default.aspx` is no longer a valid link.

To overcome this problem, you can use root-based relative URLs.

Root-Based Relative URLs

Root-based relative URLs always start with a leading forward slash to indicate the root of the site. If you take the link to the `Management` folder again, its root-based version looks like this:

```
<a href="/Management/Default.aspx">Management</a>
```

Note the leading forward slash in front of the `Management` folder to indicate the root of the website. This link is unambiguous. It always points to the `Default.aspx` file in the `Management` folder in the root. With this link, moving the `Login.aspx` page to a subfolder doesn't break it; it still points to the exact same file.

Relative URLs in Server-Side Controls

With ASP.NET Server Controls, you have another option at your disposal to refer to resources in your website: You can use the tilde (~) character to point to the current root of the site. This is especially useful when you run your website as a separate *application folder* under the main website. This would be the case if your main site ran under `www.PlanetWrox.com/Site` rather than under `www.PlanetWrox.com`, for example. To see what that means, consider this image that uses the tilde in its `ImageUrl`:

```
<asp:Image ID="Image1" runat="server" ImageUrl="~/Images/Header.jpg" />
```

When you use an application folder such as `Site`, the image is searched for at `/Site/Images/Header.jpg`. If you reconfigure the site to run without an application folder, the image is looked for at `/Images/Header.jpg` without requiring you to change any code.

You can also use the ~ syntax on regular HTML elements, provided you add the `runat` attribute. This way, the path is processed at the server and then returned to the client. The following example shows a plain HTML link that links to a page in the `Management` folder:

```
<a href="~/Management/Default.aspx" runat="server">Management</a>
```

Previous versions of Visual Studio set up the built-in web server as an application folder, making the use of the tilde a much needed option. However, the new IIS Express that now ships with VS 2012 does not use an application folder by default. So, when you start up the web server by requesting a page, its address will be something like `http://localhost:59898/` and not `http://localhost:59898/Site/`. If you still see the `Site` part in the URL, you may be running the older built-in web server instead. If that's the case, you can switch to using IIS Express by right-clicking the site in the Solution Explorer and then choosing Use IIS Express. The remainder of this book assumes you're using IIS Express and do not have an application folder in the URL for your site.

Absolute URLs

In contrast to relative URLs that refer to a resource from a document or site root perspective, you can also use absolute URLs that refer to a resource by its full path. So instead of directly referring to an image and optionally specifying a folder, you include the full name of the domain and protocol information (the `http://` prefix). Here's an example that refers to the Wrox logo at the Wrox Programmer to Programmer site (`http://p2p.wrox.com`), where you go for questions about this and other Wrox books or for general questions regarding programming:

```
<img src="http://p2p.wrox.com/images/header/wrox_logo.gif" />
```

Absolute URLs are required if you want to refer to a resource outside your own website. With such a URL, the `http://` prefix is important. If you leave it out, the browser will look for a folder called `p2p.wrox.com` inside your *own* website.

Absolute URLs are unambiguous. They always refer to a fixed location, which helps you to make sure you're always referring to the exact same resource, no matter where the source document is located. This may make you think that they are ideal to use everywhere—including references to

resources within your own site—but that's not the case. The extra protocol and domain information adds to the size of the page in the browser, making it unnecessarily slower to download. But more important, it creates difficulties if you're changing your domain name, or if you want to reuse some functionality in a different website. For example, if you previously had your site running on `www.mydomain.com` but you're moving it to `www.someotherdomain.com`, you will need to update all the absolute URLs in the entire website.

You will also have trouble with absolute URLs during development. Quite often, you test your website on a URL such as `http://localhost`. If you were to point all your images to that URL, they would all break as soon as you put your site on a production domain like `www.PlanetWrox.com`.

In short, use absolute URLs with caution. You always need them when referring to resources outside your website, but you should give preference to relative URLs within your own projects wherever possible.

Understanding Default Documents

In the context of URLs you should also know about *default documents*. When you browse to a site like `www.domainname.com` you magically see a page appear. How does this work? Each web server has so-called default documents, a list of document names that can be served to a browser when no explicit document name is supplied. So, when you browse to `www.domainname.com`, the web server scans the directory requested (the root folder in this example) and processes the first file from its default documents list it finds on disk. In most ASP.NET scenarios, the web server is set up to use `Default.aspx` as the default document. So, when you browse to `www.domainname.com` on an ASP.NET web server, you are actually served the page `www.domainname.com/Default.aspx`.

In the links you create, you should generally leave out `Default.aspx` when it isn't needed. It decreases the page size, but more important, it makes it easier for your users to type the address.

Now that you have seen how you can use URLs to point to documents and other files, it's time to look at some higher-level controls that make use of these URLs: the ASP.NET navigation controls.

USING THE NAVIGATION CONTROLS

ASP.NET 4.5 offers three useful navigation tools: `SiteMapPath`, `TreeView`, and `Menu`. Figure 7-2 shows basic examples of the three navigation controls, without any styling applied.

The `SiteMapPath` on the left shows the user the path to the current page. This helps if users want to go up one or more levels in the site hierarchy. It also helps them to understand where they are. The `TreeView` can display the structure of your site and enables you to expand and collapse the different nodes; in Figure 7-2 the entire tree is expanded. The `Menu` control on the right initially only displays the Home menu item. However, as soon as you move the mouse over the menu item, a submenu appears. In Figure 7-2 one of these child elements is the Reviews item, which in turn has child elements itself.

Similar to index.htm or index.html on other web servers.