Programming the Web with Visual Basic .NET

LYNN TORKELSON, CONSTANCE PETERSEN, AND ZAC TORKELSON

Apress™

ISBN (pbk): 1-59059-027-9

Printed and bound in the United States of America 12345678910

Technical Reviewer: Scott Stabbert

Editorial Directors: Dan Appleman, Peter Blackburn, Gary Cornell, Jason Gilmore, Simon Hayes, Karen Watterson, John Zukowski

Managing Editor: Grace Wong

Project Manager and Development Editor: Tracy Brown Collins

Copy Editor: Tom Gillen of Gillen Editorial, Inc.

Production Editor: Kari Brooks

Compositor: Susan Glinert

Artist: Cara Brunk

Indexer: Carol Burbo

Cover Designer: Kurt Krames

Manufacturing Manager: Tom Debolski

Marketing Manager: Stephanie Rodriguez

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY, 10010 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States, phone 1-800-SPRINGER, email orders@springer-ny.com, or visit http://www.springer-ny.com.

Outside the United States, fax +49 6221 345229, email orders@springer.de, or visit http://www.springer.de.

For information on translations, please contact Apress directly at 2560 9th Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax: 510-549-5939, email info@apress.com, or visit http://www.apress.com.

The source code for this book is available to readers at http://www.apress.com in the Downloads section. You will need to answer questions pertaining to this book in order to successfully download the code.

CHAPTER 15

Creating a Web Site

THE EARLIER CHAPTERS in this book presented a wealth of ASP.NET features and tools that you can use to create effective Web sites. In this final chapter, we show how to put many of those features and tools to use to create a professional-quality Web site.

In your capacity as a Visual Basic .NET Web programmer, you may well find yourself collaborating with a professional Web designer. (We're talking about a person here, not a software program!) In all likelihood, that designer will not be using Visual Studio .NET. Don't worry about it. In this chapter, we explain how to take the HTML design created using a different tool—in this case Microsoft FrontPage, but it doesn't really matter—and implement the design in Visual Studio .NET.

Here we also address the factoring of a Web application into controls and other useful classes. As with all Visual Basic .NET programming, doing a good job of factoring improves the reliability and maintainability of your code. By encapsulating all access to a database within a particular class, for example, you eliminate redundant code and make it possible to incorporate database changes and improvements reliably and efficiently. To enhance scalability, the database classes in this chapter make extensive use of stored procedures and data readers. As we demonstrate, Visual Studio .NET makes it easy to create and use stored procedures with data readers.

In earlier chapters, we focused on securing sensitive Web information through authentication and authorization provided by IIS and ASP.NET. But your Web site can make use of a visitor's identity for other purposes as well. In this chapter, we show how to use persistent cookies to personalize a Web site.

If you create a public Web site, you'll want to make sure that potential visitors can find it. Most surfers use major search engines to locate Web sites of interest to them, and so we conclude this chapter by showing how to make your Web site accessible to search engines.

Working with a Professional Designer

To make our discussions concrete, we present a case study in this chapter that builds an ASP.NET Web application from a design created in Microsoft FrontPage. The VB Snippets Web site resembles the Web application we used to illustrate the material in the last two chapters, but it has a more professional appearance and

includes more usability features. As a matter of fact, the earlier versions of this Web application served as a prototype for the case study.

The VB Snippets case study contains many Web pages and a lot of code. In this chapter, we present some of the code, but only to illustrate points not made in earlier chapters. We encourage you to download the code and experiment with it yourself to see how all of the pages in the Web site work together. The code includes scripts to set up the databases used by VB Snippets, plus another Web application that we used to add and edit the code snippets displayed by the case study. Figure 15-1 shows a collage of the Web pages that we used to maintain the VbCode database. As this application closely resembles code presented in Chapter 13, we don't discuss it further in this chapter.

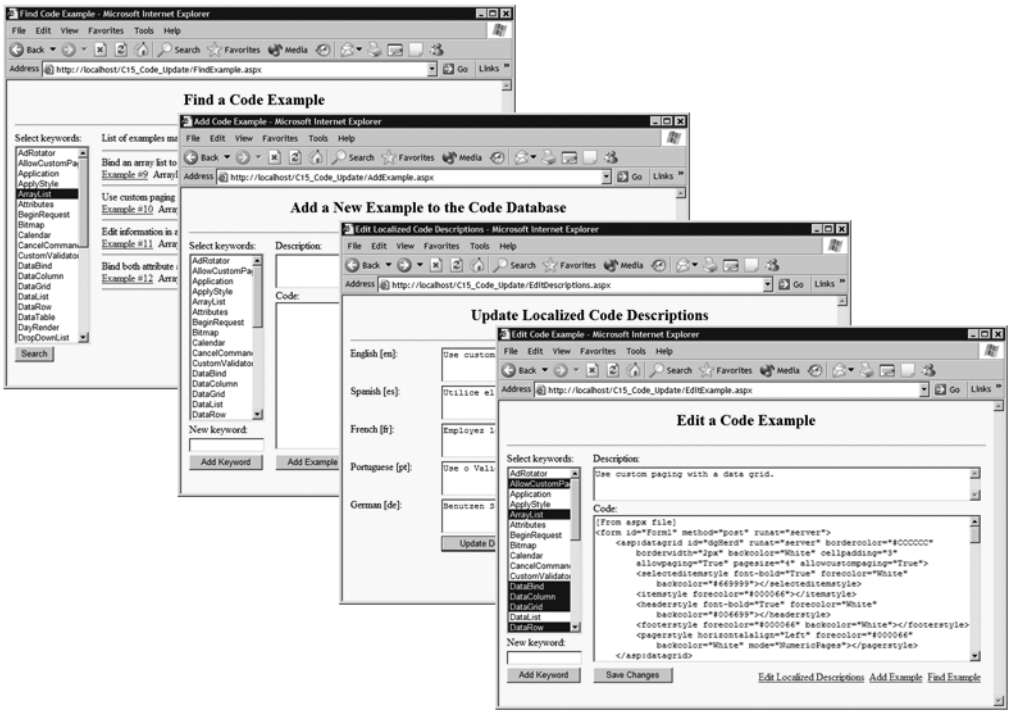

Figure 15-1. The C15_Code_Update project supports adding code snippets to the database, editing them, and adding localized descriptions.

The VbCode database stores the code snippets and localized descriptions for each. The case study supports five languages: English, Spanish, French, Portuguese, and German. Visitors to the VB Snippets Web site locate the snippets they wish to see (and copy) using keywords associated with each snippet. Figure 15-2 shows a diagram of the VbCode database in the SQL Server Enterprise Manager. All of the

accesses to this database in the VB Snippets case study occur through a single class file that's discussed later in this chapter.

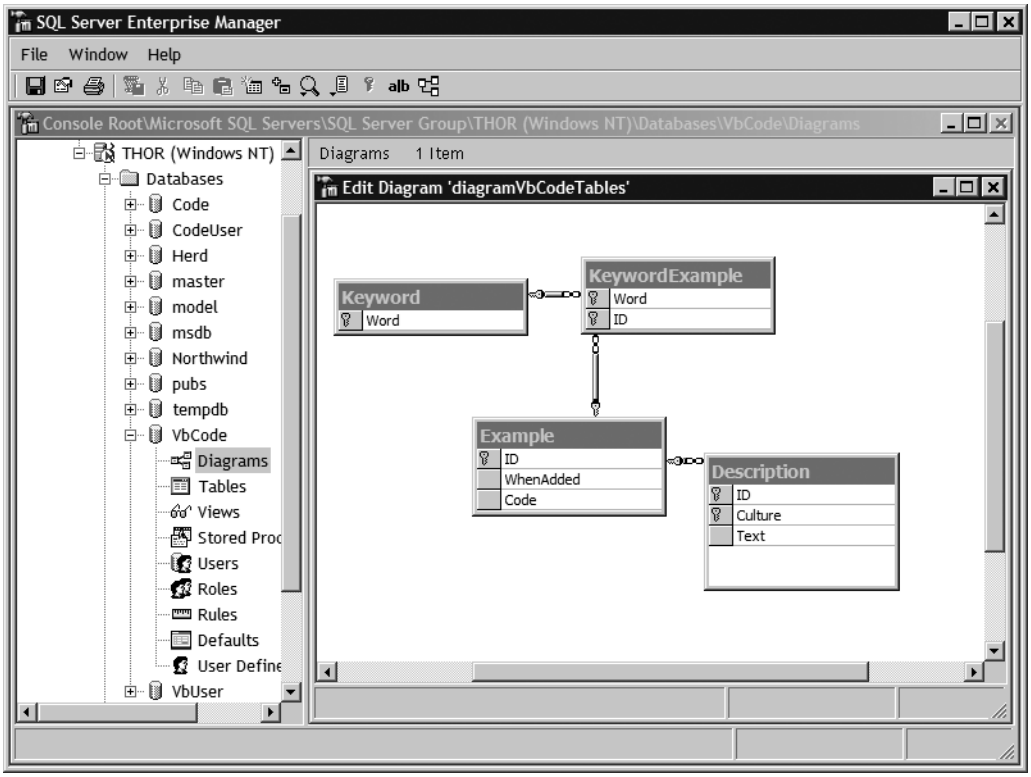

Figure 15-2. The VbCode database stores the code snippets for display by the case study.

One of the advantages of collaborating with a professional designer is that you can work concurrently. Using simple prototype Web pages, you can create and test classes that you will use when you implement the Web application after the design is completed. You can also create and populate test databases to use to test both the prototypes and the finished Web application. Overlapping the work in this manner allows you to improve the delivery time of a Web application significantly.

Like programming, Web site design is an intense, exacting occupation. And, again like programming, the design of a good Web site takes time and effort. Fortunately, Visual Studio .NET makes it possible to prototype a new Web application while its design is in progress. The prototyping period is an excellent time to establish a good working relationship with the professional Web site designer on your team.

Establishing a Working Relationship

Because of the newness of ASP.NET, Visual Basic .NET programmers and Web site designers are still formulating their working relationships. Each organization will, no doubt, create procedures to reflect its unique requirements. Over time, we predict, Web design tools will integrate nicely with Visual Studio .NET, but you don't have to wait for that day. As we demonstrate in this case study, you can quite productively build an ASP.NET Web application from a design that was created using a tool other than Visual Studio .NET.

Why won't your professional Web site designer be using Visual Studio .NET? Although Visual Studio .NET offers a fine development environment for programming, other tools have been created specifically for the purpose of creating the HTML, images, and JavaScripts used for Web sites. Until Visual Studio .NET matches those capabilities, Web site designers will—understandably—use the tools that are familiar to them. Even when Visual Studio .NET *does* match those capabilities, many Web designers will stick with the tool they know rather than switch to something new.

Irrespective of their choice of tools, however, good Web designers provide great value to a Web development project. A good Web designer combines an artistic sense with a hard-earned knowledge of the variations in popular Web browsers and is well versed in usability issues. Because it is surprisingly difficult to create HTML that displays attractively in many browsers, good Web designers work meticulously, sometimes to the point of inserting single pixels, to make their designs look right in as many browsers as possible.

Just as it is very difficult to create a good cross-browser design, it is correspondingly easy to mess it up. Even things that should not, in theory, make any difference to a browser sometimes do. As programmers, we are used to seeing code nicely indented, and the authors of this book like to see HTML nicely indented also. All of the HTML examples in this book use indentation to help reveal the nested structure. Nevertheless, "fixing" HTML by indenting it nicely sometimes causes it to display incorrectly in one or more browsers. Perhaps the biggest step a programmer can take in creating a good working relationship with a professional Web designer is to *respect the design delivered*: avoid making any unnecessary changes to the HTML, and consult with the designer on all changes.

Delivering a Web Design

Organizations vary in the amount and nature of the paperwork required for each phase of application development. In writing this book we have the luxury of sidestepping paperwork issues to cut to the essentials. From a programmer's

perspective, you can expect a Web designer to deliver the following design products:

- HTML pages (HTM or HTML files)
- CSS style sheet (CSS file)
- images (GIF and JPG files)
- JavaScript (JS file)

The VB Snippets case study makes use of all of these design products. Figure 15-3 shows a collage of seven of the thirty HTML pages delivered by the designer for VB Snippets. Let us quickly mention that several of the design pages show different states of particular pages, so not all thirty represent different Web pages to be developed. For example, the index_logged_in.htm page contains the design of the index.htm page after a visitor has logged in to the Web application.

> **NOTE** *When you download the Chapter_15 project, you receive all of the design files as well. You can find the original HTM, CSS, and JS files in the Chapter_15\design folder. The Chapter_15\images folder contains the images received from the designer.*

The disadvantage of receiving HTML files in this form is, of course, that you need to do some cutting and pasting after creating the corresponding ASPX files in Visual Studio .NET. Because you will be creating reusable controls for many portions of a Web design, however, the amount of this cutting and pasting is limited enough not to be tedious. In the meantime, the delivered design remains a constant reference. When anomalies appear during development (and some will), you can compare the source HTML sent to the browser by your code to the HTML delivered by the Web designer to determine the differences.

For Web applications other than the very smallest, you will also encounter situations that will require design updates. A frequent cause for this is that some actual string data turns out to be longer than the designer expected. In that case, you should show the designer how this affects the appearance of the Web page, and allow the Web designer to make any required changes to the HTML originally submitted. If you've established a good working relationship, the Web designer will also help you locate, in difficult cases, the specific HTML problems that cause the appearance of a Web page in a browser to differ from the design.

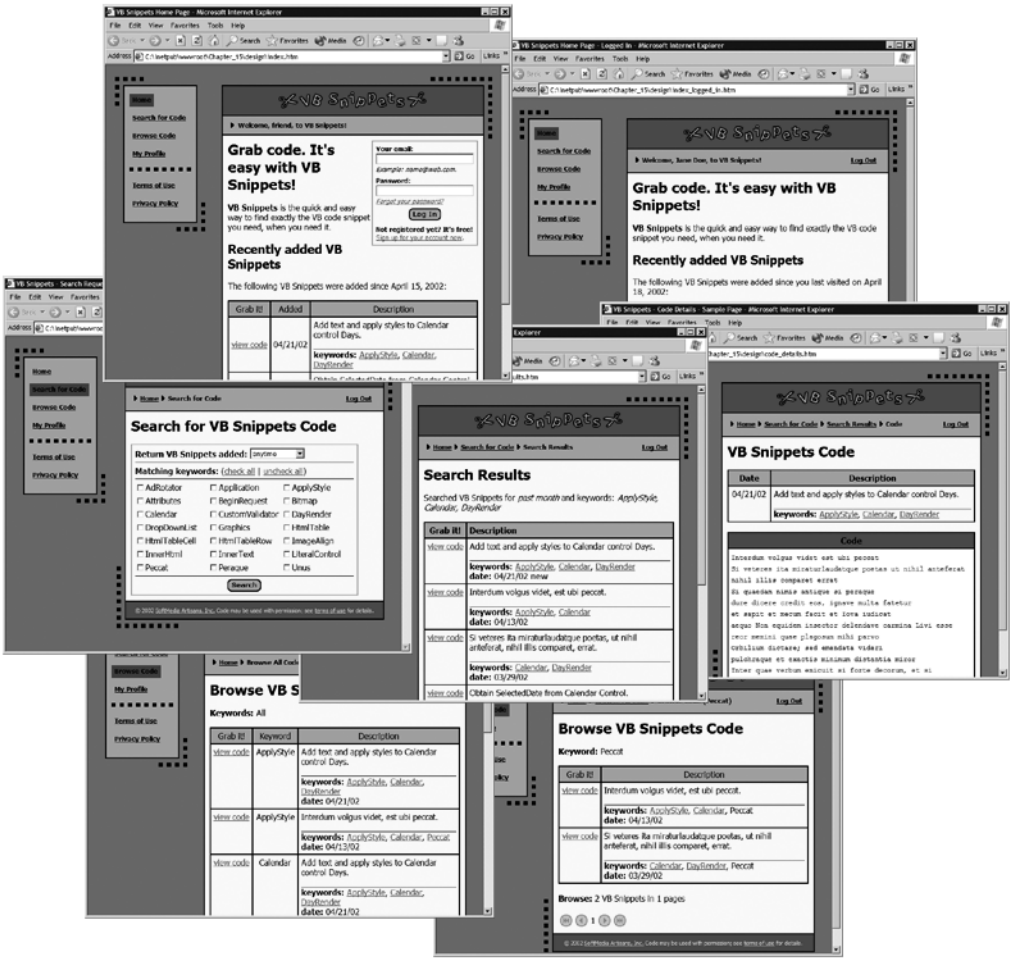

Figure 15-3. The Web design HTML specifies the appearance and interaction among the pages of the Web site.

Converting a Web Design into a Web Application

Just as the Web designer is responsible for specifying the overall appearance of and interaction among the pages of a Web site, the programmer is responsible for the server-side implementation of that design. The implementation covers a lot of ground, including the following details:

- ASP.NET Web pages (ASPX files)
- user controls (ASCX files)

- custom controls (DLL files)
- code, including reusable classes (VB files)
- database queries (SQL or stored procedures)
- resources (RESX files)
- configuration (CONFIG files)
- deployment (DLL files and SETUP.EXE)

For a successful Web application, all of these pieces must work together flawlessly, must be scalable, and must be easy to maintain. Our aim with this book has been to show you how to use Visual Studio .NET and Visual Basic .NET to accomplish exactly that. The VB Snippets case study puts these tools to work to convert a Web design into an ASP.NET application.

When collaborating with a professional Web designer, you will already have created many of these pieces in prototype form before you ever receive the finished design. Once the design has been delivered, you need to complete the development of the Web application.

Factoring a Web Design for Implementation

Take another look at Figure 15-3. Many design elements repeat, some with variations, from page to page. The left side of each Web page displays a navigation menu with items representing the homepage and the second-level pages in the site. The large rectangle on the right side of each Web page contains four distinct sections. A graphical logo appears at the top. Underneath the logo is a "breadcrumb" section that allows a visitor to return to pages higher in the hierarchy from the current page. (Appendix A provides more information about providing breadcrumb links to enhance a Web site's usability.) Underneath the breadcrumb section is the main content section, which varies greatly from page to page (but it is the only section to do so). The bottom section displays a standard footer containing copyright and business information.

Figure 15-4 shows an ASPX page in Visual Studio .NET that distills the basic structure of all the HTML pages delivered by the Web designer. A table containing just one row of two cells defines the overall left and right structure of each page.

The three user controls in Figure 15-4—`smaNav`, `smaLogo`, and `smaFoot`—appear in every Web page in the case study. The custom control `BreadCrumbs` appears in every Web page except for the homepage (index.aspx), which is at the top of the hierarchy and needs no backwards navigation to parent levels of the hierarchy. We

present the code for each of these controls in this chapter. We also describe other controls with more-limited use, such as a control that displays localized graphic buttons. Figure 15-5 shows the basic page in Visual Studio .NET after a Build and Browse. As you can see, the four controls on the page duplicate the appearance of the HTML received from the Web designer.

Figure 15-4. The aa_basic.aspx Web page defines the structure of all the ASPX pages in the VB Snippets case study.

The VB Snippets case study also makes use of four class files. The `CLocalization` class, introduced in Chapter 13, encapsulates common functionality needed to localize each Web page. We do not discuss it further here. Two classes—`CVbUser` and `CVbCode`—encapsulate the code that's needed to access the two databases used by the application. We discuss portions of the `CVbCode` class in this chapter, and you can download and examine the full code for both. The fourth class, `CVisitor`, encapsulates information about the current visitor that is needed to personalize some of the displays in VB Snippets. As shown later in this chapter, the information in `CVisitor` comes from persistent cookies as well as the VbUser database.

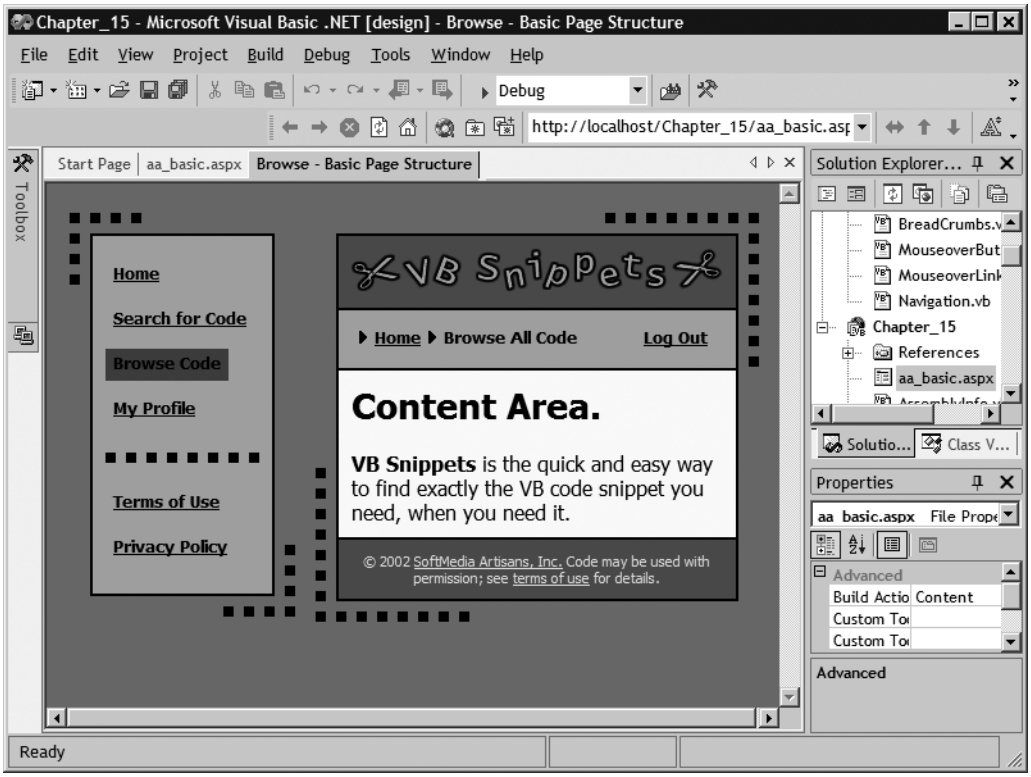

Figure 15-5. The controls on the aa_basic.aspx Web page encapsulate design elements that recur throughout the Web site.

By factoring a Web application in this manner, you achieve the benefits of object-oriented programming. The classes may be instantiated and used throughout the Web application. When changes are needed, they are isolated within the appropriate class. Once tested, the code in a class works properly no matter what other class uses it.

Implementing a Web Design

The coherent look and feel of a professionally designed Web site arises from the common design elements, which lend themselves to the creation of controls. The main content area of each page, however, is tailored to the specific information provided there. When working with a professional designer, you can implement a Web application via the following steps:

1. Create class files during the prototyping phase. You will need to modify and extend the classes during implementation, of course, but this gives you a running start.

2. Isolate common design elements into controls and develop the controls.

3. Create a basic structure page containing the design elements that are common to all of the Web pages in the application.

4. For each unique Web page in the application, create a separate ASPX page that matches the basic structure page.

5. Set the properties of each modifiable control to meet the requirements of the specific Web page being developed.

6. Paste the HTML from the content area delivered by the Web designer into the content area of the ASPX page.

7. Localize the strings in the content area.

8. Substitute server controls for all HTML elements in the Web page that you will need to handle at the server. You may convert existing HTML tags by adding an `id` attribute and `runat="server"`, or you may substitute ASP.NET Web server controls that generate equivalent HTML.

9. Write and test the server code behind the Web page.

You can determine the common design elements by inspection, as we already showed for the VB Snippets case study. Some of the resulting controls will belong in the basic structure page as shown in Figures 15-4 and 15-5. Once you create the basic structure page, you are ready to tackle localizing and programming the main content area of each page.

Localizing a Web Page Design

We covered the localization of Web pages in Chapter 13. In a nutshell, you create separate resource files for each language that your Web site will support. Your code accesses the localized strings through a `System.Resources.ResourceManager` instance declared like this:

```
Protected rm As ResourceManager
```

In the `Page_Load` procedure, you instantiate the `ResourceManager` class with a statement such as the following:

```
rm = New ResourceManager("Chapter_15.strings", GetType(index).Assembly)
```

The ASPX file inherits the code behind specified in its `Page` directive and uses the protected `ResourceManager` instance to supply the correct values for all localized strings. After you paste the content area from the design into the corresponding ASPX file, you simply go through the HTML looking for strings such as

```
<span class="lblform">Password:</span>
```

Replace each string with the corresponding localization element, like this:

```
<span class="lblform"><%=rm.GetString("Password")%></span>
```

As we explained in Chapter 13, localizing strings in this manner requires the addition of an RESX file to your application for each language that your Web application will support. However, there can be benefits to localization even if you don't plan your Web application to be multilingual. Should the phrasing or terms used in a Web site change, whether because of marketing or other considerations, you can makethe necessary corrections in the RESX file without having to hunt them down in the HTML for all the Web pages in the application. Figures 15-6 and 15-7 show screenshots of the VB Snippets homepage, index.aspx, in English and German.

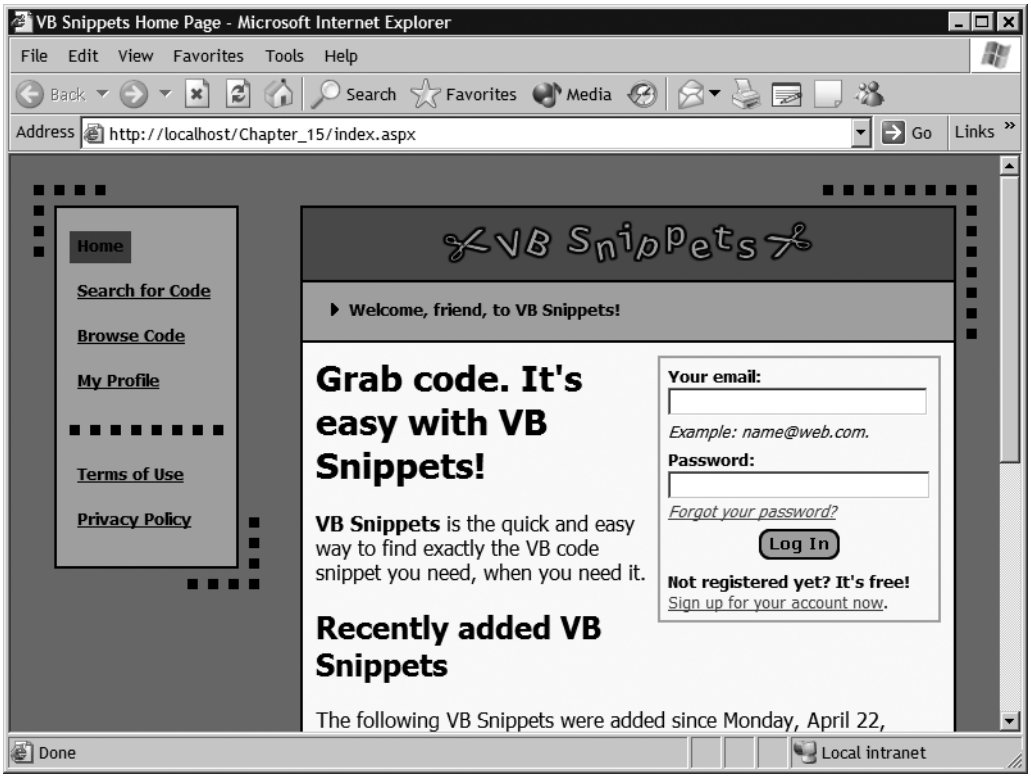

Figure 15-6. The localized index.aspx page obtains its strings via a `ResourceManager` *instance rather than hard-coded HTML.*

One thing you won't be expected to do as a programmer is to provide the translations yourself. For the VB Snippets case study, we obtained the translations from the Internet to illustrate the mechanics of localization, but for a real application we would employ a professional translator.

Note that the controls on the page have also been localized, including the graphical Log In button. As you will see shortly, the localization of controls works just like the localization of Web pages.

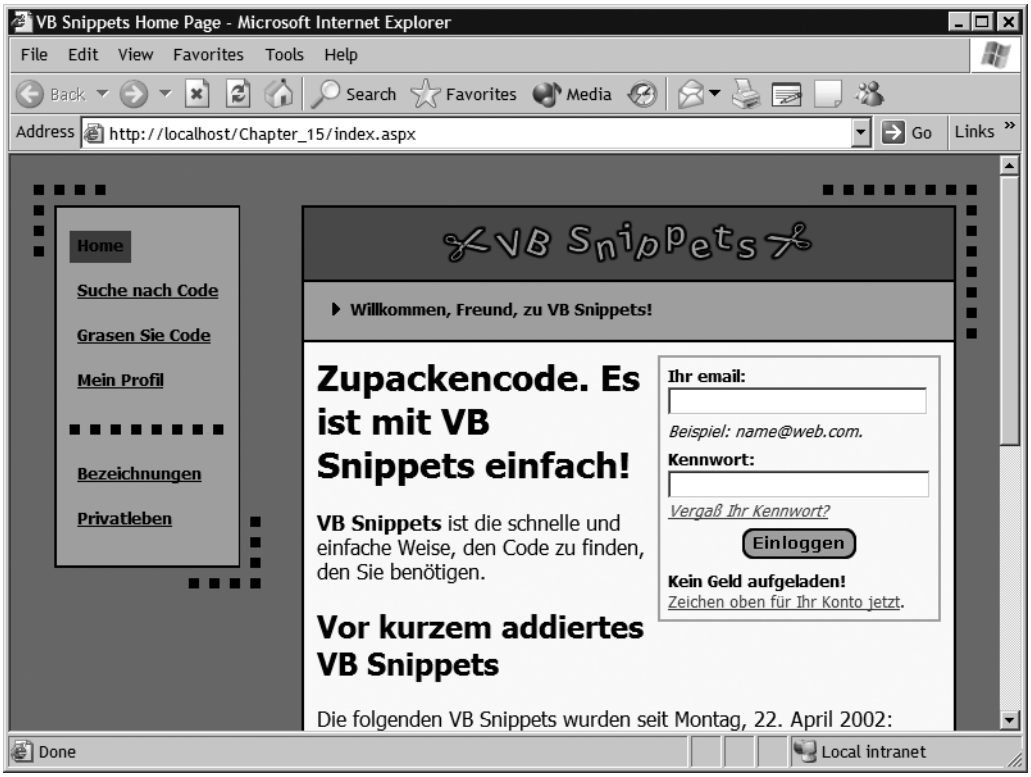

Figure 15-7. When a visitor's browser specifies German as the preferred language, the index.aspx page responds accordingly.

Modifying the Structure of a Web Page in Code

The VB Snippets application does not require a visitor to log in to view the information available. Instead, the application invites the visitor to register and log in to receive personalized information. For example, the registered visitor can identify keywords of interest and request email notifications when code snippets associated with those keywords are added to the database. VB Snippets also keeps track of the last visit by a logged-in visitor and uses that information to call attention to code snippets that were added to the database since his or her last visit. Once registered with VB Snippets, the visitor can log in to the application and receive personalized displays and update preferences on his or her profile.

After a visitor logs in, however, the rectangle in the homepage that accepts login information becomes superfluous. The Web designer recognized this by providing two design versions of the homepage: index.htm and index_logged_in.htm. The VB Snippets case study shows how to modify the appearance of a page to adapt to such circumstances. Figure 15-8 shows a screenshot of a registered visitor's log in entries on the VB Snippets homepage. Figure 15-9 shows the top portion of the homepage after the visitor completes the login procedure.

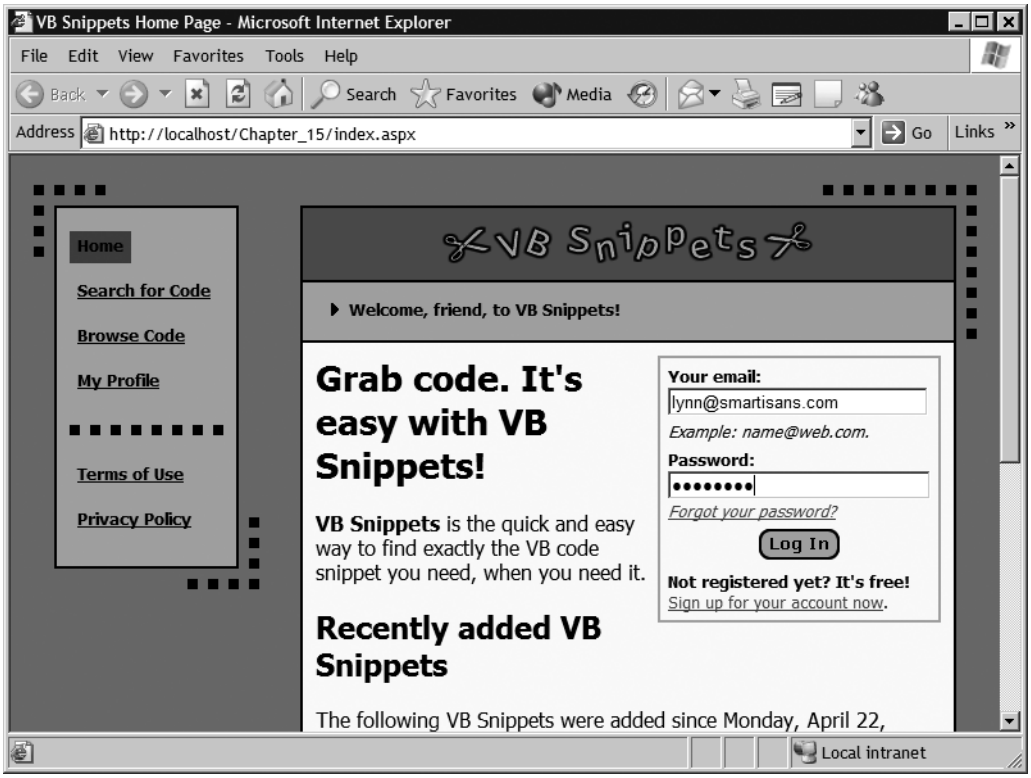

Figure 15-8. The index.aspx page provides a convenient area for a registered visitor to log in.

The version of index.aspx received by a logged-in visitor does not display the no-longer-needed login area. It does however, greet the visitor by name and display the word *new* with each snippet added to the database since this visitor's last visit. As shown later in this chapter, some of the information used to identify a logged-in visitor comes from a persistent cookie stored at the client. The visitor can remove the cookie at any time by clicking the Log Out link that appears in the breadcrumb section of the page.

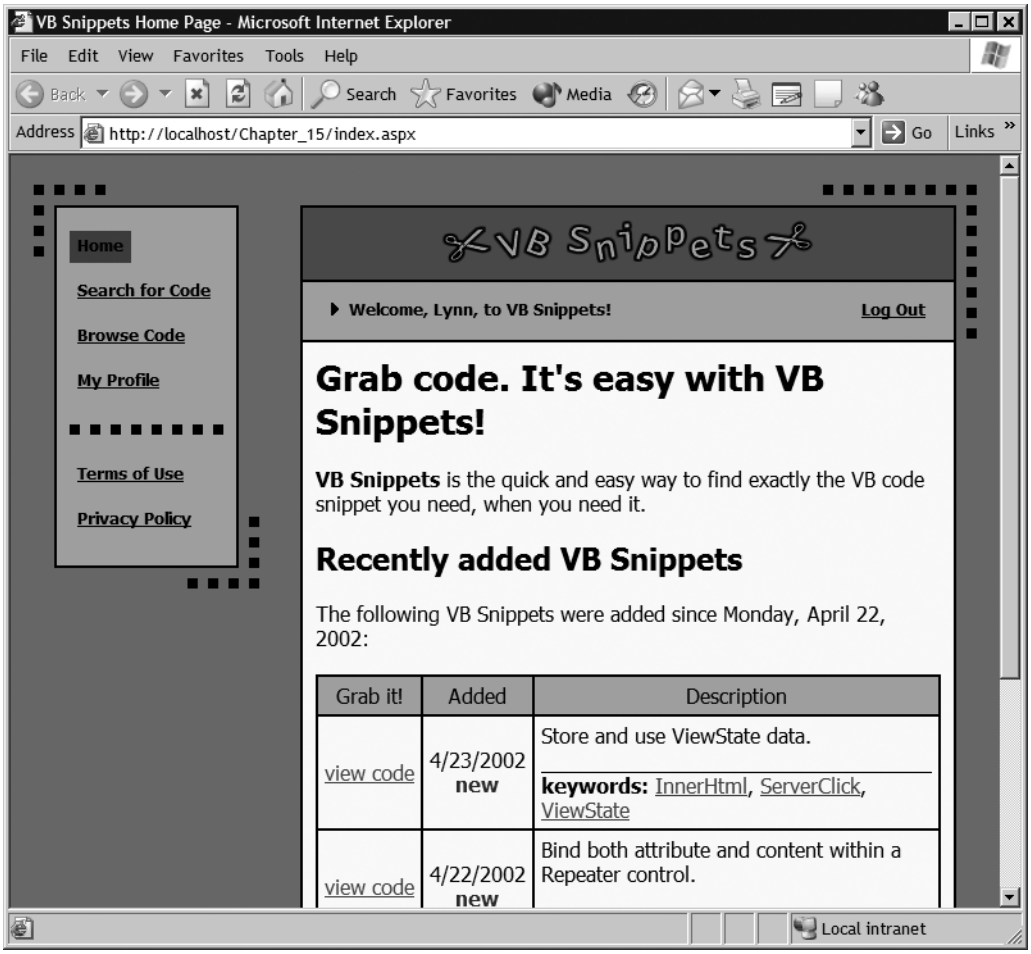

Figure 15-9. A logged-in visitor receives a personalized version of the index.aspx page.

The `visible` property of server controls comes in handy for modifying the appearance of a Web page in circumstances such as these. When the `visible` property is `False`, ASP.NET renders no HTML for the control. You can't get more invisible than that! In the present example, all of the HTML for the login area is contained within a table element with a start tag originally designed as follows:

```
<table class="tform" cellSpacing="6" cellPadding="0" align="right" border="0">
```

By adding `id="tblLogin"` and `runat="server"` attributes to the tag, you convert the table to a server control. The following code then removes the table entirely for a logged in visitor:

```
If User.Identity.IsAuthenticated Then
    Me.tblLogin.Visible = False
End If
```

An authenticated visitor does not see the `tblLogin` element. ASP.NET generates no HTML for it in building the response.

It is just as easy to add information that appears only to logged-in visitors. Simply add an element in the proper position like so:

```
<span id=pLoggedIn runat="server" visible="false"></span>
```

If the visitor has not logged in, the element does not appear. For a logged-in visitor, though, you can display personalized information as follows:

```
If User.Identity.IsAuthenticated Then
    Me.pLoggedIn.Visible = True
    Me.pLoggedIn.InnerHtml = "<b>Some personalized information...<b>"
End If
```

The `visible` property gives you great flexibility in tailoring the appearance of a Web page for particular situations. Another way that Web page displays can vary is through the use of data-bound controls, as covered in Chapter 7.

Incorporating Bound Data into a Web Page

If you download the code for this chapter, you can check out all of the HTML originally delivered with the VB Snippets design. As you will see, the design versions of many of the Web pages, including index.htm, show tabular information in HTML tables. In VB Snippets, however, we plan to bind those displays to current database information. Therefore, the ASPX pages replace those tables in the design with data-bound controls.

VB Snippets uses the DataGrid control in several pages to display bound data. Templates customize the displays within the columns of the grid. Listing 15-1 defines the DataGrid used in the content area of the index.aspx page. The `columns` element defines the three column-display, with an `asp:templatecolumn` element enclosing the definition of each column. Because this is a display-only Web application, only `itemtemplate` elements are included within each column.

Listing 15-1. The `grdNewExamples` *DataGrid in index.aspx*

```
<asp:datagrid id="grdNewExamples" runat="server" cellpadding="5"
    cssclass="tborder" autogeneratecolumns="False">
    <headerstyle horizontalalign="Center" cssclass="thead"></headerStyle>
    <itemstyle cssclass="tmain"></itemstyle>
    <columns>
        <asp:templatecolumn itemstyle-borderwidth="2" headerstyle-borderwidth="2"
            itemstyle-horizontalalign="Center" itemstyle-wrap="False" >
            <itemtemplate>
                <a href='<%# Container.DataItem("Link") %>'>
                    <%=rm.GetString("ViewCode")%></a>
            </itemtemplate>
        </asp:templatecolumn>
        <asp:templatecolumn itemstyle-borderwidth="2" headerstyle-borderwidth="2"
            itemstyle-horizontalalign="Center" itemstyle-wrap="False" >
            <itemtemplate>
                <%# Container.DataItem("Date") %><br>
                <asp:label id="lblNew" runat="server" cssclass="new" >
                    <%# Container.DataItem("New") %></asp:label>
            </itemtemplate>
        </asp:templatecolumn>
        <asp:templatecolumn headertext="Description" itemstyle-borderwidth="2"
            headerstyle-borderwidth="2">
            <itemtemplate><%# Container.DataItem("Description") %>
                <p class="topsep" /><b><%=rm.GetString("keywords")%></b>
                <%# Container.DataItem("Keywords") %>
            </itemtemplate>
        </asp:templatecolumn>
    </columns>
</asp:datagrid>
```

As you can see in Listing 15-1, a data grid can display both localized strings and bound data. You can also add information to a data grid programmatically as we did here:

```
Me.grdNewExamples.Columns(0).HeaderText = rm.GetString("GrabIt")
Me.grdNewExamples.Columns(1).HeaderText = rm.GetString("Added")
Me.grdNewExamples.Columns(2).HeaderText = rm.GetString("Description")
```

This code adds the localized header strings that appear at the top of each column in the grid.

All data for the grid displays comes from methods of the `CVbCode` class, which provides access to the VbCode database. Later in this chapter, we take a look at the

inner workings of this class. When a visitor is logged in, the following code retrieves and binds data for the `grdNewExamples` display:

```
Me.grdNewExamples.DataSource = oCode.NewExamples( _
    NewExampleDate:=oVisitor.LastVisit, _
    StartDate:=CDate(Me.txtSearchDate.Text), _
    Culture:=_sCulture)
Me.grdNewExamples.DataBind()
```

Here, the `oCode` and `oVisitor` variables reference instances of the `CVbCode` and `CVisitor` classes, respectively. The private `_sCulture` variable specifies the language to display for the current visitor.

To complete the implementation of a Web design, you work through each page, converting its HTML content into the ASPX equivalent. In the natural order of things, some pages will be somewhat complex and others will be very simple. To verify for yourself the straightforward nature of this process, you can compare the Web pages in the VB Snippets case study with the HTML versions delivered by the Web designer.

From a Visual Basic .NET programmer's perspective, the implementation of controls used within the Web pages provides a more interesting challenge. Now let's take a look at the controls used in the VB Snippets case study.

Encapsulating Design Elements in Controls

As with Web pages themselves, the common elements in a Web design can range from very simple to quite complex. By writing controls for the common design elements, you ensure that the HTML and code for each element is reusable across all of the Web pages in the application. Should changes be required, you have only one place to make them. In Chapter 10, 11, and 12, we covered the creation of user controls and custom Web controls. In this chapter, we demonstrate how to create controls to implement the VB Snippets design. We start with a couple of simple controls and progress to the more substantial.

Using Graphics in a Header Control

The top section of each VB Snippets Web page contains a graphical logo. Figure 15-10 shows the appearance of the logo on a white background. As you can see, the image contains no pixels beyond those absolutely necessary to produce the logo. Reducing the size and number of the graphics in a Web site reduces the response time of a Web site, thereby improving the visitor's experience with it.

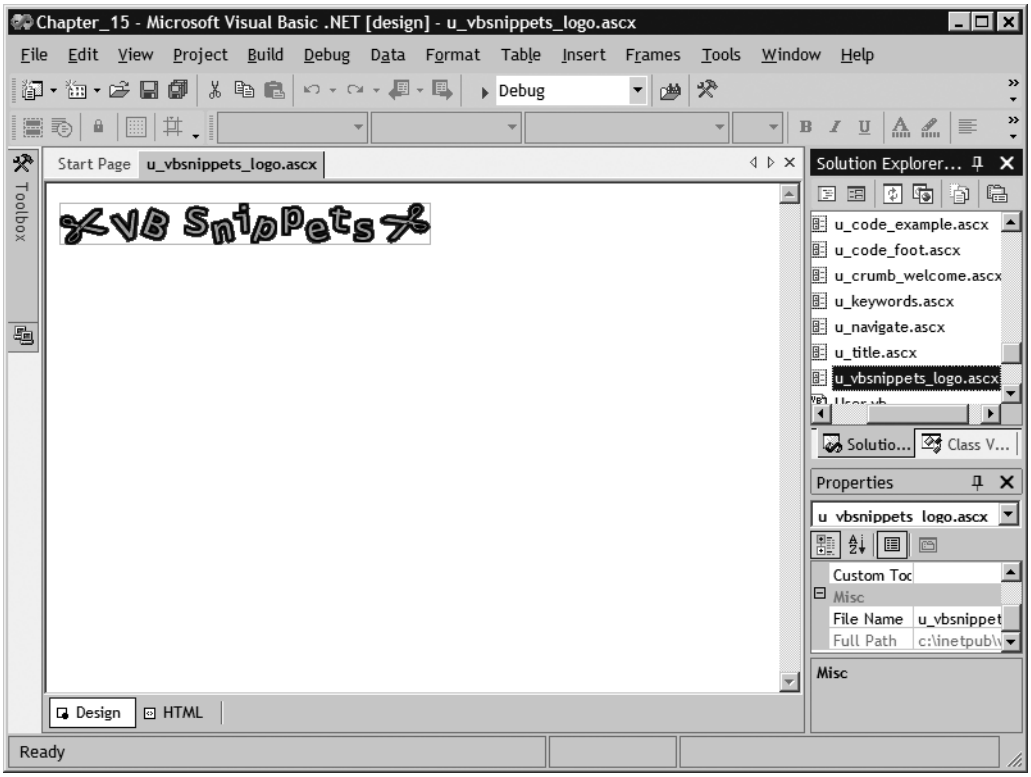

Figure 15-10. The Web designer purposely kept the VB Snippets logo.gif image as small as possible.

The VB Snippets logo is displayed by the simple user control u_vbsnippets_logo.ascx in Listing 15-2. Although encapsulating the logo in a user control doesn't buy much in this case, it will still be helpful if the logo should change. You would then have to edit in only one place to change the width and height of the new logo, rather than having to change each ASPX file in the Web application.

Listing 15-2. The u_vbsnippets_logo.ascx User Control

```
<%@ Control Language="vb" AutoEventWireup="false"
    Codebehind="u_vbsnippets_logo.ascx.vb"
    Inherits="Chapter_15.u_vbsnippets_logo"
    TargetSchema="http://schemas.microsoft.com/intellisense/ie5" %>
<img border="0" src="images/logo.gif" alt="VB Snippets Logo"
    width="290" height="33">
```

Once you add a `Register` directive to a Web page for this user control, you can add it to the proper table cell like this:

```
<td width="100%" class="tlogo" align="middle">
    <sma:logo id="smalogo" runat="server" />
</td>
```

In the implementation of VB Snippets, the basic structure page naturally contains the `Register` directives for those controls used throughout the design, making it a simple operation to add them to any new ASPX file.

The `tlogo` CSS class specifies the background color for the table cell that contains the logo. The definition of the `tlogo` class in Styles.css is as follows:

```
.tlogo{
background-color: #369;
border: 1px solid #000;
}
```

The Web designer created the logo to avoid dithering when displayed against this specific blue-green Web-safe color.

Developing a Localized Footer Control

The bottom section of the right-hand side of each VB Snippets page (look back at Figure 15-5) contains a footer that displays business information and links. This user control is localized using the same code that you use to localize a Web page. Listing 15-3 contains the localized ASCX code for this control.

Listing 15-3. The u_code_foot.ascx User Control

```
<%@ Control Language="vb" AutoEventWireup="false"
    Codebehind="u_code_foot.ascx.vb"
    Inherits="Chapter_15.u_code_foot"
    TargetSchema="http://schemas.microsoft.com/intellisense/ie5" %>
<p class="footertopic">© 2002
    <a class="footer"
        href="http://www.smartisans.com">SoftMedia Artisans, Inc.</a>
    <%=rm.GetString("CodePermission")%>
    <a class="footer" href="terms_of_use.htm">
        <%=rm.GetString("TermsOfUse")%></a>
    <%=rm.GetString("forDetails")%>
</p>
```

Styles provided by the Web designer control the font, size, and color of the text displayed in the `u_code_foot` user control, as well as the background and link colors. Listing 15-4 shows the complete code behind this user control. As you see, the only code requirement is to supply the `ResourceManager` instance required for localization.

Listing 15-4. The u_code_foot.ascx.vb User Control Code

```
Protected rm As ResourceManager

Private Sub Page_Load(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles MyBase.Load
    'Put user code to initialize the page here
    rm = New ResourceManager("Chapter_15.strings", GetType(index).Assembly)
End Sub
```

Although useful, the logo and footer user controls don't offer much programming challenge. The navigation menu control, although still fairly simple, varies its display from page to page. Let's look at it now.

Developing a Control for the Navigation Menu

The left side of each VB Snippets Web page displays a navigation menu with an item at the top for the homepage and with items for the second-level pages in the Web site beneath the homepage item. The design adds this wrinkle: the item for the *currently* displayed page appears highlighted instead of as a link. The navigation menu for lower-level search pages and browse pages highlight the search and browse items, respectively. As we shall see, breadcrumb links provide for navigation within the search and browse pages.

The VB Snippets navigation menu has been implemented as a custom control *within* a user control. The custom control uses property settings to render the correct HTML for each Web page, and the user control encloses the custom control in a table that positions the snip-snip-snip dotted line graphics in the upper left and lower right corners.

Rendering the Navigation HTML

The custom control in Listing 15-5 uses the value of its `CurrentIndex` property to control the rendering of its HTML. Because the control resides in a separate DLL, it also requires a `ResourceManager` instance from its caller to provide the desired localization.

Listing 15-5. The Navigation.vb Custom Control

```
Option Strict On
Imports System.Resources
Imports System.ComponentModel
Imports System.Web.UI

<DefaultProperty("CurrentIndex"), _
ToolboxData("<{0}:Navigation runat=server></{0}:Navigation>")> _
Public Class Navigation
    Inherits System.Web.UI.WebControls.WebControl

    Private _rm As ResourceManager

    Private _asLink() As String = {"index.aspx", _
        "code_search.aspx", "code_browse_all.aspx", _
        "my_profile.aspx", "terms_of_use.aspx", "privacy_policy.aspx"}

    Private _iCurrent As Integer = 0

    <Bindable(True), Category("Behavior"), DefaultValue(0)> _
    Property CurrentIndex() As Integer
    Get
        Return _iCurrent
    End Get
    Set(ByVal Value As Integer)
        If Value <= UBound(_asLink) Then
            _iCurrent = Value
        End If
    End Set
    End Property

    <Bindable(False)> _
    Property ResourceManager() As ResourceManager
    Get
        Return _rm
    End Get
    Set(ByVal Value As ResourceManager)
        _rm = Value
    End Set
    End Property
```

```
Protected Overrides Sub Render(ByVal output As System.Web.UI.HtmlTextWriter)
    If _rm Is Nothing Then
        Throw New Exception("C15_Ctl_Lib-Navigation: " _
            & "No resource manager provided.")
    End If

    Dim asLabel() As String = Split(_rm.GetString("NavStrings"), ",")
    Dim i As Integer, iMax As Integer = UBound(asLabel)

    If iMax <> UBound(_asLink) Then
        Throw New Exception("C15_Ctl_Lib-Navigation: " _
            & "Wrong number of localized navigation strings.")
    End If

    'Move to left margin
    output.WriteLine()

    For i = 0 To iMax
        If i = 4 Then
            'Insert separator line before next paragraph
            output.AddAttribute(HtmlTextWriterAttribute.Align, "center")
            output.RenderBeginTag(HtmlTextWriterTag.P)
            output.AddAttribute(HtmlTextWriterAttribute.Border, "0")
            output.AddAttribute(HtmlTextWriterAttribute.Width, "120")
            output.AddAttribute(HtmlTextWriterAttribute.Height, "8")
            output.AddAttribute(HtmlTextWriterAttribute.Alt, "")
            output.AddAttribute(HtmlTextWriterAttribute.Src, _
                "images/box_h8.gif")
            output.RenderBeginTag(HtmlTextWriterTag.Img)
            output.RenderEndTag()
            output.WriteLine()
        End If

        'Insert link text paragraph
        output.RenderBeginTag(HtmlTextWriterTag.P)
```

```
                If i = 0 Then
                    'Add shim for spacing
                    output.AddAttribute(HtmlTextWriterAttribute.Width, "8")
                    output.AddAttribute(HtmlTextWriterAttribute.Height, "8")
                    output.AddAttribute(HtmlTextWriterAttribute.Alt, "")
                    output.AddAttribute(HtmlTextWriterAttribute.Src, _
                        "images/shim.gif")
                    output.RenderBeginTag(HtmlTextWriterTag.Img)
                    output.Write("<br />")
                End If

                If i = _iCurrent Then
                    'Already linked to this page (or section)
                    output.AddAttribute(HtmlTextWriterAttribute.Class, "navtopic")
                    output.RenderBeginTag(HtmlTextWriterTag.Span)
                    output.Write(asLabel(i))
                    output.RenderEndTag()
                Else
                    'Output link
                    output.AddAttribute(HtmlTextWriterAttribute.Href, _asLink(i))
                    output.AddAttribute(HtmlTextWriterAttribute.Class, "nav1")
                    output.RenderBeginTag(HtmlTextWriterTag.A)
                    output.Write(asLabel(i))
                    output.RenderEndTag()
                End If

                If i = iMax Then
                    'Add spacing
                    output.Write("<br /> ")
                End If

                'Close paragraph
                output.RenderEndTag()
                output.WriteLine()
            Next
        End Sub
    End Class
```

The `CurrentIndex` property for the navigation custom control uses a value of 0 for the homepage item, and the second level items start with a `CurrentIndex` value of 1. The `aslink()` string array contains the list of links for all of the Web pages in the control, with the list ordered to correspond to the associated `CurrentIndex` values. The text for each item comes from the `NavStrings` data in the resource file that's determined by the `ResourceManager` instance. For English-speaking visitors, the `NavStrings` data appears in strings.resx like this:

```
<data name="NavStrings">
    <value>Home,Search for Code,Browse Code,My Profile,Terms of Use,Privacy
        Policy</value>
</data>
```

Each of the other supported languages has its own `NavStrings` data.

The `Render` method outputs links for all of the navigation menu items except for the item corresponding to the `CurrentIndex` property. The `Render` method outputs a `span` element for the `CurrentIndex` item. CSS classes supplied by the Web designer control the appearance of the elements within the control. The `Render` method also inserts a graphical separator after the first four items.

The actual HTML rendered by the navigation custom control matches that supplied by the Web designer for each page. By looking at the HTML delivered by the Web designer for each page, it was a simple task to see the pattern. Testing the control consisted of comparing the HTML rendered by the control with that in the original design for each Web page.

One of the great things about custom controls is that you can render *any* HTML specified by the Web designer. In tough situations, this can be a lifesaver.

Embedding a Custom Control in a User Control

In VB Snippets, the navigation custom control is enclosed within the u_navigate.ascx user control. You can register a custom control in a user control in the same way as you do in a Web page. Figure 15-11 shows the Design view of the navigation menu user control.

The code for the navigation menu user control (Listing 15-6) provides the property values required by the custom control it encloses. The code instantiates the `ResourceManager` class and sets the `smaccNav.ResourceManager` property to reference it before using its own `CurrentIndex` property to set the `smaccNav.CurrentIndex` property.

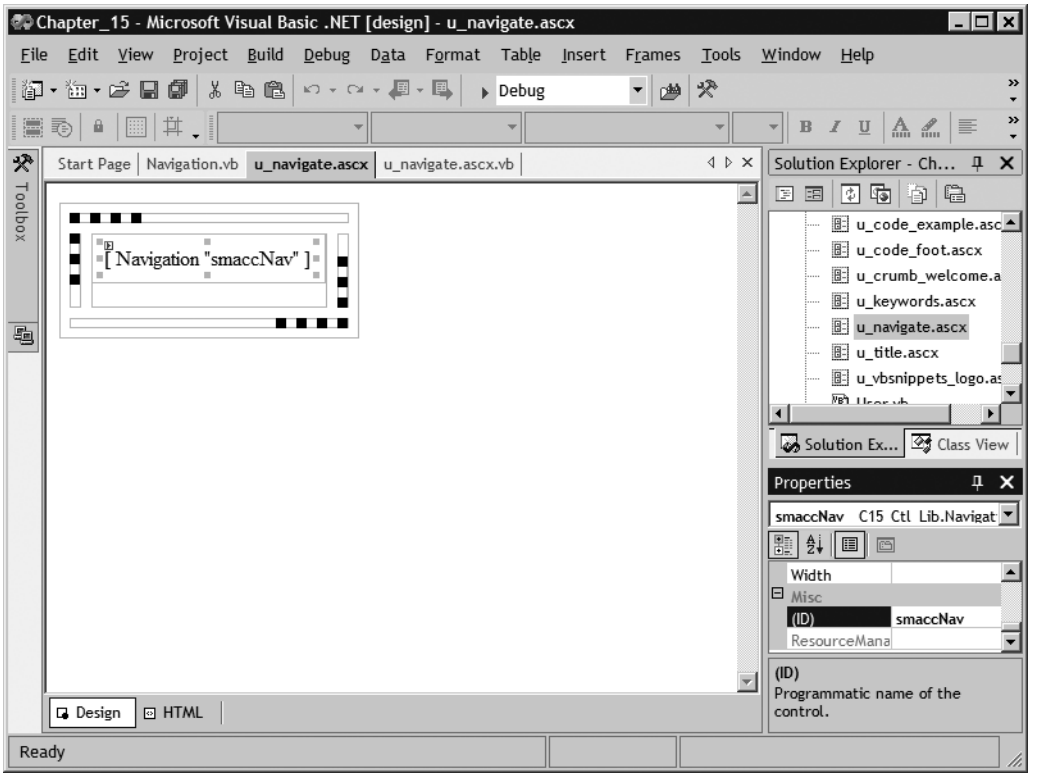

Figure 15-11. For design purposes, the navigation menu user control encloses a custom control within a table containing graphical elements.

Listing 15-6.The u_navigate.ascx.vb User Control Code

```
Option Strict On
Imports System.Resources

Public MustInherit Class u_navigate
    Inherits System.Web.UI.UserControl
    Protected WithEvents smaccNav As C15_Ctl_Lib.Navigation

    [Designer generated code omitted]

    Private Sub Page_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load
        'Put user code to initialize the page here
        Dim rm As ResourceManager
```

```
        rm = New ResourceManager("Chapter_15.strings", GetType(index).Assembly)
        smaccNav.ResourceManager = rm
        smaccNav.CurrentIndex = _CurrentIndex
    End Sub

    Private _CurrentIndex As Integer = 0

    Property CurrentIndex() As Integer
    Get
        Return _CurrentIndex
    End Get
    Set(ByVal Value As Integer)
        _CurrentIndex = Value
    End Set
    End Property
End Class
```

The navigation menu user control makes it a trivial task to output the HTML for the left-hand cell of each VB Snippets Web page. For example, the following lines form the complete specification for the left-hand cell of index.aspx:

```
<td vAlign="top">
    <sma:nav id="smaNav" runat="server" currentindex="0" />
</td>
```

The code for each page varies only according to the `CurrentIndex` value. In most cases, this value can be set once (as just shown) and forgotten. However, it is possible to navigate to a code_example page from the homepage, a search page, or a browse page. For the code_example page, therefore, the `CurrentIndex` property is set programmatically.

Developing a Breadcrumb Control

Hansel and Gretel used breadcrumbs to mark their path in the forest, but the birds had a competing agenda. Well-designed Web sites use the same idea to assist a visitor in navigation: breadcrumb links above the main content of a Web page help to orient a visitor within a Web site and provide a convenient way to retrace one's last steps. Fortunately, Web sites don't have to worry about birds.

If you surf the Web as often as most of us do, you will have noticed that breadcrumb links are not standardized across Web sites. Figure 15-12 shows an example of breadcrumb links as implemented in VB Snippets.

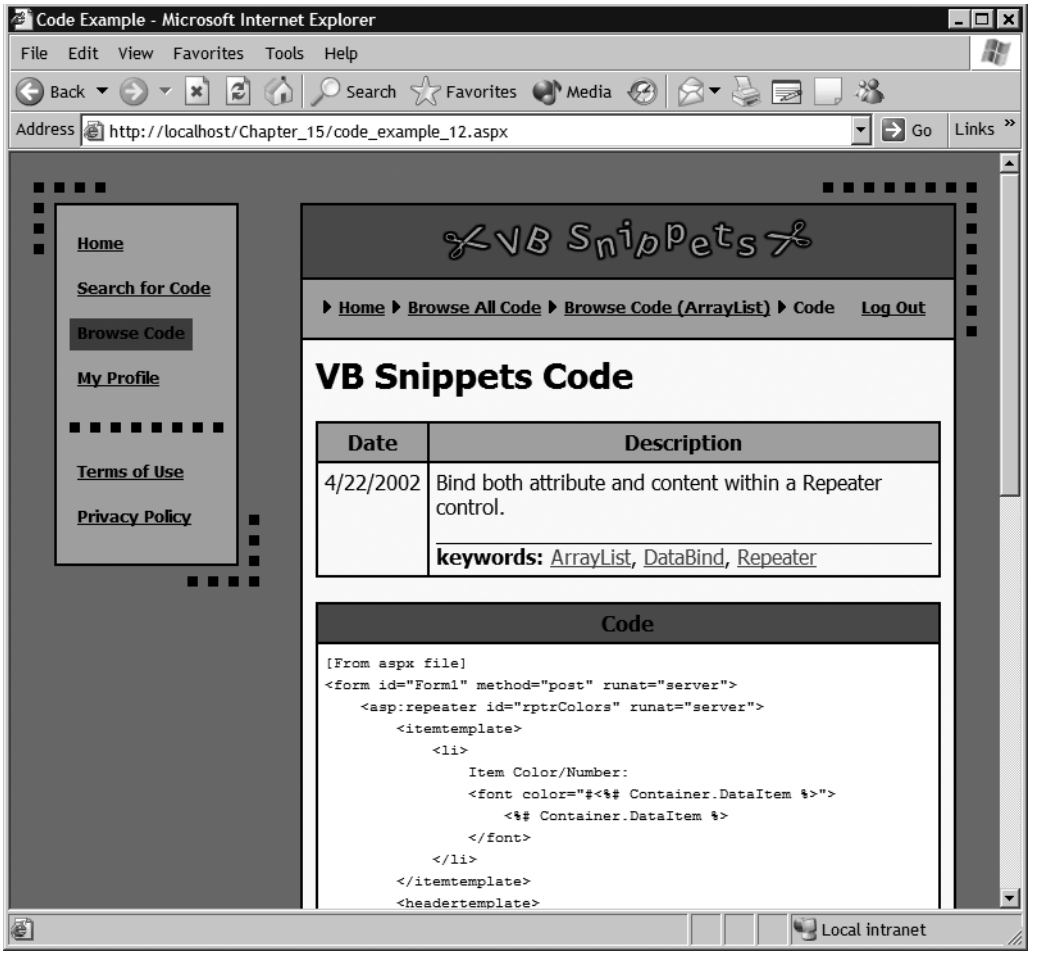

Figure 15-12. VB Snippets breadcrumbs indicate the path the visitor used to select a particular code snippet.

The VB Snippets Web site allows the visitor to browse through all of the code examples or to browse through just those snippets that are associated with a particular keyword. Whenever a visitor clicks a keyword link, the Web site displays the browse page for that keyword. The screenshot in Figure 15-12 displays a code snippet from the `ArrayList` browse page. The breadcrumb links provide an easy way for the visitor to return to that particular browse page, to the browse-all page, or to the homepage. And, of course, all but one of the links in the navigation menu are also available.

Deciding What Breadcrumb Links to Display

Breadcrumb links are not yet standardized for several reasons. When you begin to consider what breadcrumbs to display in each situation, you realize that providing the right links is not so simple as it originally appears. There are many convoluted paths through a Web site. If you adopt the approach of adding a breadcrumb link for each stop along the way, your page will display too many links to be useful. Such an approach makes it easy for the visitor to retrace his or her steps, but doing so obscures the overall organization of the Web site. To make this approach work, you need a well-conceived method of reducing the number of links displayed as the visitor navigates through the Web site.

Another possible approach to breadcrumb links is to show the position of the current page within the hierarchy regardless of the path the visitor used to arrive there. Such breadcrumb links keep a visitor oriented, but make it difficult to retrace one's steps. The example in Figure 15-12 points out another flaw in this hierarchical approach: some Web pages don't fit neatly into a hierarchy. You can display a code snippet from a browse page, a search page, or the homepage, so who's to say where it fits within the hierarchy?

As it happens, this is just the sort of issue that professional Web designers relish. You can expect a professional Web designer to specify breadcrumb links that are optimized for the particular requirements of your site. The VB Snippets Web designer carefully designed breadcrumb links to work effectively with the case study, and the BreadCrumbs.vb custom control implements that design.

Implementing a Hybrid Breadcrumb Link Design

Most of the VB Snippets Web pages display hierarchical breadcrumb links. When it comes to the display of code examples, though, the breadcrumb links make it easy for the visitor to return to the previous search page, browse page, or homepage.

Listing 15-7 contains the code for the breadcrumbs custom control. Like the navigation custom control, the breadcrumbs custom control exposes a `ResourceManager` property that's used for localization purposes. The `CurrentPage` property identifies the last breadcrumb in the chain—the breadcrumb that is not a link. The `BrowseItem` property identifies the specific keyword to display in a browse breadcrumb link. (Figure 15-12 illustrates how the `BrowseItem` property is used.) The `From` property identifies the route the visitor took to arrive at the current page. VB Snippets uses session-state variables to store this information. The read-only `CodePath` property returns the index used by the code_example page to control the navigation menu display. If the visitor arrived via a search page, for example, the `CodePath` property returns the value `1`.

Listing 15-7. The BreadCrumbs.vb Custom Control

```
Option Strict On
Imports System.Resources
Imports System.ComponentModel
Imports System.Web.UI
Imports System.Web

<DefaultProperty("CurrentPage"), _
ToolboxData("<{0}:BreadCrumbs runat=server></{0}:BreadCrumbs>")> _
Public Class BreadCrumbs
    Inherits System.Web.UI.WebControls.WebControl

    Private _rm As ResourceManager
    Private _sCrumbList As String = "Home,"
    Private _sLinkList As String = "index.aspx"
    Private _sCurrentPage As String
    Private _sFrom As String
    Private _iCodePath As Integer = 0
    Private _sBrowseItem As String = String.Empty

    <Bindable(True), Category("Behavior"), DefaultValue(""")> _
    Property CurrentPage() As String
    Get
        Return _sCurrentPage
    End Get
    Set(ByVal Value As String)
        'Values: Browse, Search, Code, Terms, Privacy
        _sCurrentPage = Value
        Select Case Value
        Case "Browse"
            'Browse all unless BrowseItem set
            If _sBrowseItem = String.Empty Then
                _sCrumbList &= "BrowseAll"
            End If
        Case "Search"
            'General search unless SearchQuery set
            _sCrumbList &= "SearchFor"
        Case "Results"
            _sCrumbList = "Home,SearchFor,Results"
            _sLinkList = "index.aspx,code_search.aspx"
```

```
    Case "Code"
        'Set default
        _sCrumbList &= "Code"
        'Code breadcrumbs depend upon visitor's navigation
        If Not _sFrom = String.Empty Then
            If _sFrom = "B" Then
                'Got here from Browse All
                _sCrumbList = "Home,BrowseAll,Code"
                _sLinkList = "index.aspx,code_browse_all.aspx"
                _iCodePath = 2
            ElseIf _sFrom.StartsWith("B:") Then
                'Got here from Browse Item
                _sCrumbList = "Home,BrowseAll,Browse,Code"
                _sBrowseItem = _sFrom.Substring(2)
                _sLinkList = "index.aspx,code_browse_all.aspx,code_browse_" _
                    & _sBrowseItem & ".aspx"
                _iCodePath = 2
           ElseIf _sFrom.StartsWith("S:") Then
                'Got here from Search Results
                _sCrumbList = "Home,SearchFor,Results,Code"
                _sLinkList _
                    = "index.aspx,code_search.aspx,code_search_results.aspx"
                _iCodePath = 1
            Else
                'Retain defaults
            End If
        End If
    Case Else


        'Homepage only link
        _sCrumbList &= Value
    End Select
End Set
End Property
```

```
<Bindable(True), Category("Behavior"), DefaultValue("")> _
Property BrowseItem() As String
Get
    Return _sBrowseItem
End Get
Set(ByVal Value As String)
    'Specific browse
    _sBrowseItem = Value
    _sCrumbList = "Home,BrowseAll,Browse"
    _sLinkList = "index.aspx,code_browse_all.aspx"
End Set
End Property

<Bindable(False)> _
Property From() As String
Get
    Return _sFrom
End Get
Set(ByVal Value As String)
    _sFrom = Value
End Set
End Property

<Bindable(False)> _
ReadOnly Property CodePath() As Integer
Get
    Return _iCodePath
End Get
End Property

<Bindable(False)> _
Property ResourceManager() As ResourceManager
Get
    Return _rm
End Get
Set(ByVal Value As ResourceManager)
    _rm = Value
End Set
End Property

Protected Overrides Sub Render(ByVal output As System.Web.UI.HtmlTextWriter)
    If _rm Is Nothing Then
        Throw New Exception("C15_Ctl_Lib-BreadCrumbs: " _
            & "No resource manager provided.")
    End If
```

```
Try
    Dim asCrumb() As String = Split(_sCrumbList, ",")
    Dim asLink() As String = Split(_sLinkList, ",")

    Dim i As Integer, iMax As Integer = UBound(asCrumb)

    'All except last crumb are links
    Debug.Assert(iMax = UBound(asLink) + 1, "C15_Ctl_Lib-BreadCrumbs: " _
        & "Mismatch in list counts.")

    'Crumbs: Home, BrowseAll, Browse, SearchFor, Results,
    '        Code, Terms, Privacy
    For i = 0 To iMax
        If asCrumb(i) = "Browse" Then
            asCrumb(i) = _rm.GetString("Browse") _
                & " (" & _sBrowseItem & ")"
        Else
            asCrumb(i) = _rm.GetString(asCrumb(i))
        End If
    Next

    'Move to left margin
    output.WriteLine()

    'Render breadcrumbs paragraph
    output.AddAttribute(HtmlTextWriterAttribute.Class, "lfloatpad")
    output.RenderBeginTag(HtmlTextWriterTag.P)

    For i = 0 To iMax
        'Render arrow image
        output.AddAttribute(HtmlTextWriterAttribute.Border, "0")
        output.AddAttribute(HtmlTextWriterAttribute.Width, "6")
        output.AddAttribute(HtmlTextWriterAttribute.Height, "10")
        output.AddAttribute(HtmlTextWriterAttribute.Alt, "")
        output.AddAttribute(HtmlTextWriterAttribute.Src, _
            "images/arrow.gif")
        output.RenderBeginTag(HtmlTextWriterTag.Img)

        If i < iMax Then
            'Create a link
            output.AddAttribute(HtmlTextWriterAttribute.Class, "crumb")
            output.AddAttribute(HtmlTextWriterAttribute.Href, asLink(i))
            output.RenderBeginTag(HtmlTextWriterTag.A)
            output.Write(asCrumb(i))
            output.RenderEndTag()
```

```
                    Else
                        'Display current page info
                        output.AddAttribute(HtmlTextWriterAttribute.Class, "crumb")
                        output.RenderBeginTag(HtmlTextWriterTag.Span)
                        'If Not _sBrowseItem = String.Empty Then
                        '    asCrumb(i) &= " (" & _sBrowseItem & ")"
                        'End If
                        output.Write(asCrumb(i))
                        output.RenderEndTag()
                    End If
                Next

                output.RenderEndTag()

                'Render log in/out link
                output.AddAttribute(HtmlTextWriterAttribute.Class, "rfloatpad")
                output.RenderBeginTag(HtmlTextWriterTag.P)
                output.AddAttribute(HtmlTextWriterAttribute.Class, "logout")

                If Page.User.Identity.IsAuthenticated Then
                    'Already logged in
                    output.AddAttribute(HtmlTextWriterAttribute.Href, _
                        "index.aspx?action=logout")
                    output.RenderBeginTag(HtmlTextWriterTag.A)
                    output.Write(_rm.GetString("LogOut"))
                    output.RenderEndTag()
                ElseIf _sCurrentPage <> "LogIn" Then
                    'Link to log in page
                    output.AddAttribute(HtmlTextWriterAttribute.Href, "log_in.aspx")
                    output.RenderBeginTag(HtmlTextWriterTag.A)
                    output.Write(_rm.GetString("LogIn"))
                    output.RenderEndTag()
                End If

                output.RenderEndTag()
                output.RenderEndTag()

            Catch ex As Exception
                Throw New Exception("C15_Ctl_Lib-BreadCrumbs: " & ex.Message)
            End Try
        End Sub
    End Class
```

The breadcrumbs custom control works by building two lists: one for the text to display for each breadcrumb and another list for the link associated with each breadcrumb except the last, which represents the current page. The control stores the lists in the string variables `_sCrumbList` and `_sLinkList` until the `Render` method needs them. The `Render` method splits the strings into string arrays named `asCrumb()` and `asLink()` for processing.

The values in `asCrumb()` serve as keys to resource files to obtain localized strings. The `Render` method creates the appropriate link for each breadcrumb except for the last one in the `asCrumb()` array.

Using Session State to Remember Previous Links

To enable a page to determine its predecessor, VB Snippets uses a session-state variable. When the `code_browse` page processes, for example, it executes the following instructions:

```
sKeyword = oCode.Keyword
Me.smaccCrumbs.BrowseItem = sKeyword
Session("From") = "B:" & sKeyword
```

After the page is disposed, the `From` session-state variable retains a value that identifies the last page active in the session and, in this case, the keyword used to build that particular browse page. Note that this is the same keyword used to set the `BrowseItem` property of the breadcrumbs control.

When the code_example page executes, it sets the `From` property equal to the `From` session-state variable, as in the following code from the `Page_Load` procedure:

```
smaccCrumbs.From = CStr(Session("From"))
smaccCrumbs.CurrentPage = "Code"
smaccCrumbs.ResourceManager = rm
```

The information supplied by this code enables the breadcrumbs control to construct the lists used by the `Render` method.

In return, the code_example page relies upon information supplied by the breadcrumbs control to set the `CurrentIndex` property of the navigation menu. The following code, also from the `Page_Load` procedure, transfers the appropriate index:

```
Dim ouNav As u_navigate
ouNav = CType(Me.FindControl("smaNav"), u_navigate)
ouNav.CurrentIndex = smaccCrumbs.CodePath
```

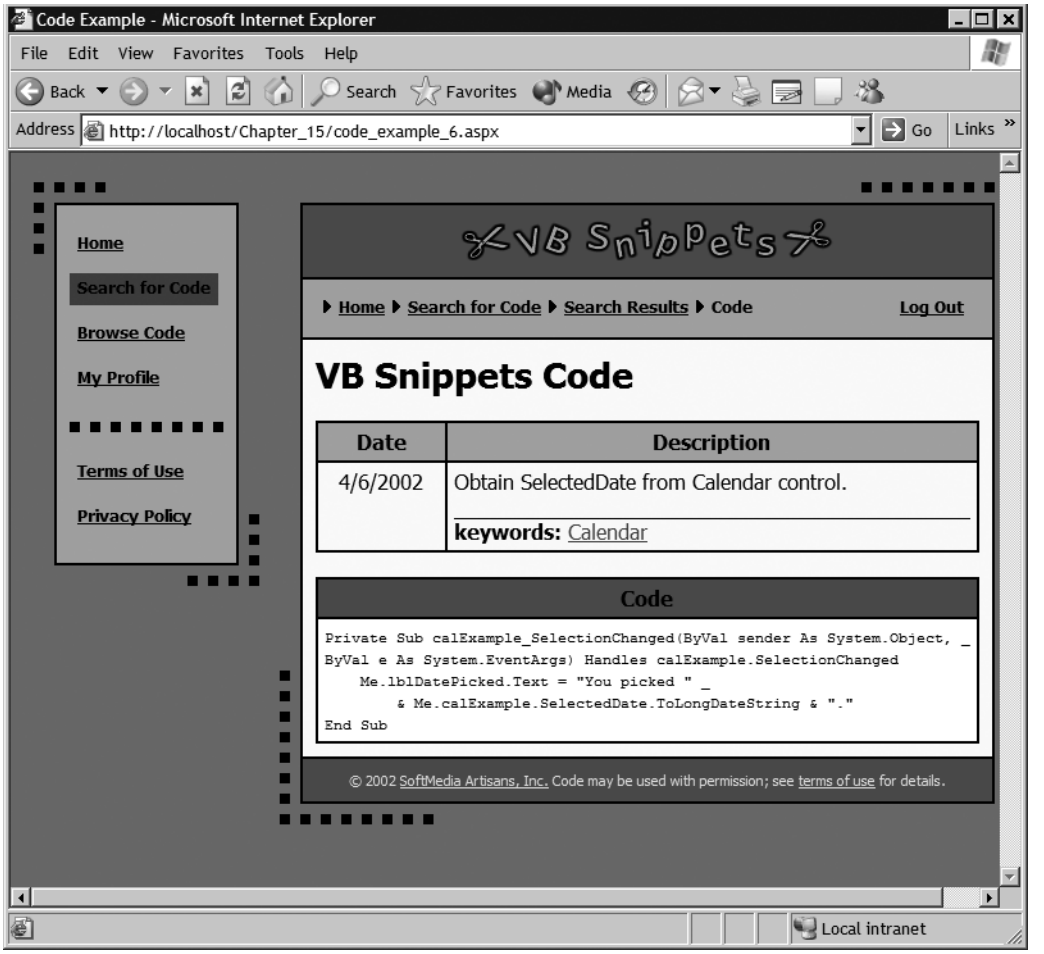

Figure 15-13. A visitor can display a code snippet by means of a search page.

Although we've focused on the code_browse page during this discussion, the code_example page also displays the correct breadcrumb links when accessed by other paths. Figure 15-13 shows a screenshot of a code snippet located through the code_search_results page.

The visitor in Figure 15-13 can return directly to the search results page that led to this snippet, can skip back to start another search, or can link to another page entirely from the navigation menu.

Using Authentication to Vary the Display

To conclude our presentation of the breadcrumb custom control, we need to mention one other feature. As Figure 15-13 shows, a logged-in visitor can also log out directly from the breadcrumbs control. If the visitor is not logged in, the logout link is replaced by a link to log_in.aspx.

To determine whether to display the logout link or the login link, the `Render` method checks the `Page.User.Identity.IsAuthenticated` property. An authenticated visitor gets the logout link, and an unauthenticated visitor gets the login link.

The controls presented to this point comprise all those that appear in every VB Snippets Web page. A control does not have to appear in every page to be useful, however. Let's look now at a more specialized control.

Localizing Graphic Effects

In several VB Snippets Web pages, the Web designer has created graphic links or buttons that change color when the cursor passes over them. When you use a graphic for a link or a button, changing the color in this way signals the visitor that the graphic is "live," which improves the usability of your Web site. Features like this help to distinguish good Web sites from the not so good.

The index.aspx page contains two such graphics, both of which serve as buttons, returning control to the index.aspx page. Figure 15-14 contains a screenshot of the index.aspx page scrolled down to make both buttons visible.

The MouseoverLink.vb custom control encapsulates the functionality that makes these graphic buttons and links operate. Listing 15-8 contains the code for this control, which exposes four properties. The `Culture` and `ResourceManager` properties are used for localization. The `Image` property identifies the name of the graphic to be displayed, for example, "go" or "log_in". The `Link` property identifies the link associated with the graphic, if any. If a link is specified, the custom control outputs HTML that links to that page when the visitor clicks the graphic. If no link is specified, the custom control generates a button that posts back to the page containing the graphic.

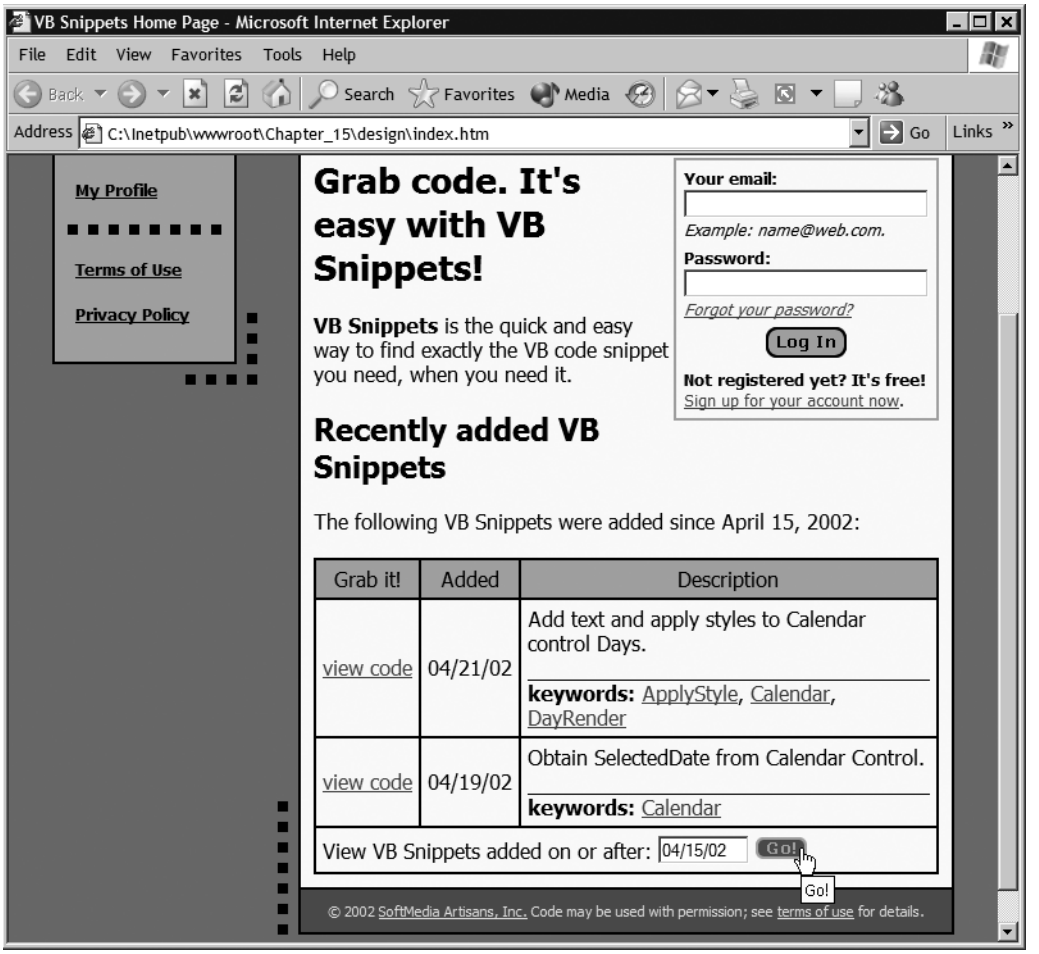

Figure 15-14. The Go button on the index.aspx page changes color to signal that it is a "live" graphic.

Listing 15-8. The MouseoverLink.vb Custom Control

```
Option Strict On
Imports System.Resources
Imports System.ComponentModel
Imports System.Web.UI

<DefaultProperty("Text"), _
ToolboxData("<{0}:MouseoverLink runat=server></{0}:MouseoverLink>")> _
```

```
Public Class MouseoverLink
    Inherits System.Web.UI.WebControls.WebControl
    Implements IPostBackEventHandler

    Private _sImage As String = "go"
    Private _sLink As String = String.Empty
    Private _sCulture As String = String.Empty
    Private _WasClicked As Boolean = False
    Private _rm As ResourceManager

    <Bindable(True), Category("Appearance"), DefaultValue("")> _
    Property Image() As String
    Get
        Return _sImage
    End Get
    Set(ByVal Value As String)
        _sImage = Value
    End Set
    End Property

    Property Link() As String
    Get
        Return _sLink
    End Get
    Set(ByVal Value As String)
        _sLink = Value
    End Set
    End Property

    <Bindable(True), Category("Appearance"), DefaultValue("")> _
    Property Culture() As String
    Get
        Return _sCulture
    End Get
    Set(ByVal Value As String)
        _sCulture = Left(Value, 2)
        If _sCulture = "en" Then _sCulture = String.Empty
    End Set
    End Property
```

```
<Bindable(False)> _
Property ResourceManager() As ResourceManager
Get
    Return _rm
End Get
Set(ByVal Value As ResourceManager)
    _rm = Value
End Set
End Property

ReadOnly Property WasClicked() As Boolean
Get
    Return _WasClicked
End Get
End Property

Protected Overrides Sub Render(ByVal output As System.Web.UI.HtmlTextWriter)
    If _rm Is Nothing Then
        Throw New Exception("C15_Ctl_Lib-MouseoverButton: " _
            & "No resource manager provided.")
    End If

    'Develop localized image paths
    Dim sImgSrc As String = "images/" & _sImage & "_btn"
    If Not _sCulture = String.Empty Then
        sImgSrc &= "_" & _sCulture
    End If

    Dim sOver As String = "changeImage('" & _sImage & "','','" _
        & sImgSrc & "_over.gif',1)"

    sImgSrc &= ".gif"

    'Move to left margin
    output.WriteLine()

    'Add scripted mouseover events
    output.AddAttribute("onmouseover", sOver)
    output.AddAttribute("onmouseout", "restoreImage()")

    If _sLink = String.Empty Then
        'Link button: add postback event reference
        Dim sScript As String = "javascript:" _
            & Page.GetPostBackEventReference(Me, _sImage)
        output.AddAttribute(HtmlTextWriterAttribute.Href, sScript)
```

```
        Else
            'Link to page: add link reference
            output.AddAttribute(HtmlTextWriterAttribute.Href, _sLink)
        End If

        'Render beginning anchor tag
        output.RenderBeginTag(HtmlTextWriterTag.A)

        'Render image for button
        output.AddAttribute(HtmlTextWriterAttribute.Id, _sImage)
        output.AddAttribute(HtmlTextWriterAttribute.Name, _sImage)
        output.AddAttribute(HtmlTextWriterAttribute.Border, "0")
        output.AddAttribute(HtmlTextWriterAttribute.Src, sImgSrc)
        output.AddAttribute(HtmlTextWriterAttribute.Alt, _rm.GetString(_sImage))
        output.AddAttribute(HtmlTextWriterAttribute.Title,
            _rm.GetString(_sImage))

        'Provide width and height of image to improve browser performance
        Dim sGif As New System.Text.StringBuilder(sImgSrc)
        sImgSrc = sGif.Replace("/", "\").ToString
        Dim bmp As System.Drawing.Bitmap _
            = New System.Drawing.Bitmap(Page.Server.MapPath(sImgSrc))
        output.AddAttribute(HtmlTextWriterAttribute.Width, CStr(bmp.Width))
        output.AddAttribute(HtmlTextWriterAttribute.Height, CStr(bmp.Height))

        'Render img element within anchor element
        output.RenderBeginTag(HtmlTextWriterTag.Img)
        output.RenderEndTag()

        'Render end tag for anchor element
        output.RenderEndTag()
        output.WriteLine()
    End Sub

    Public Sub RaisePostBackEvent(ByVal eventArgument As String) _
    Implements System.Web.UI.IPostBackEventHandler.RaisePostBackEvent
        Debug.Assert(CType(eventArgument, String) = _sImage, _
            "MouseoverLink got incorrect postback")
        _WasClicked = True
    End Sub
End Class
```

This control makes it easy to add graphical buttons and links to a Web page but, as implemented, it does not stand alone. It requires the support of a JavaScript—

rollover.js—provided by the Web designer. The script is identified to the Web page in the HTML `head` section like this:

```
<script src="rollover.js" type=text/javascript></script>
```

It is also important to preload the graphics that will display when the visitor moves the cursor over the original graphic. If you don't do this, a delay occurs while the browser retrieves the missing graphic. In the index.aspx page, the code to preload the mouseover graphics looks like this:

```
<script language="javascript">
    <!--
    preloadSwapImages('images/log_in_btn_over.gif','images/go_btn_over.gif')
    //-->
</script>
```

Finally, the custom control depends upon the naming convention used by the designer for the graphics. For the originally displayed Go button, the image is named go_btn.gif. The mouseover version of the same graphic is named go_btn_over.gif. Thus, the control can construct the filenames for both graphics from the name supplied in the `Image` property.

The localized versions of the buttons extend the same convention. For example, the French version of the Go button is named go_btn_fr.gif, and its mouseover counterpart is named go_btn_fr_over.gif. The `Culture` property enables the custom control to construct the required filenames for its localized buttons. Figure 15-15 shows the Spanish versions of the graphical buttons on the index.aspx page. Here, the color of the log_in button shows that visitor has moved the cursor over it.

One additional point is worth mentioning about this custom control: it is important for performance at the client to specify the `height` and `width` attributes of each graphic in its `img` tag. Of course, the graphics for different languages vary in size, so the custom control determines the height and width of each graphic programmatically and includes those values in the HTML it renders.

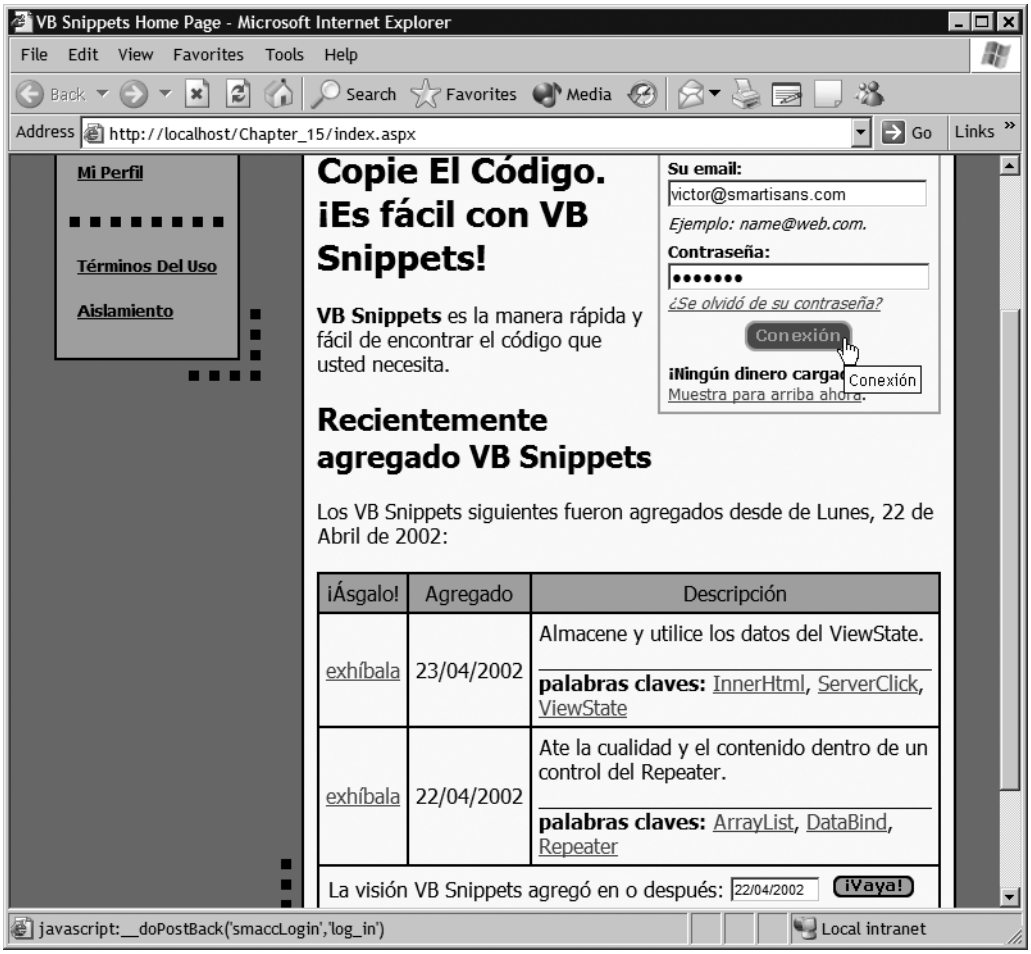

Figure 15-15. The MouseoverLink.vb custom control displays localized graphics.

Personalizing Web Pages

We mentioned at the beginning of this chapter that the VB Snippets case study uses registration for personalization rather than for security. Therefore, VB Snippets does not force the visitors to receive a login page before gaining access to the site. To support this approach, the Web.config file contains the following entries:

```
<authentication mode="Forms" />
<authorization>
    <allow users="*" /> <!-- Allow all users -->
</authorization>
```

This combination of entries provides for free access to the Web site, but it makes forms authentication available for registered visitors. The `CVisitor` class, which we present shortly, contains the code that authenticates the user. As we saw earlier, the breadcrumbs custom control tests the `Page.User.Identity.IsAuthenticated` property to determine whether to output a logout or login link. Let's look at an example that makes more use of the `Page.User.Identity` class.

Getting the User Name from an Identity Object

The index.aspx page occupies the top of the page hierarchy, so it does not use the breadcrumbs custom control. The u_crumb_welcome.ascx user control appears in place of the breadcrumbs control. If the visitor is logged in, the control greets the visitor by name and offers a logout link. If the visitor is not logged in, the control displays a generic greeting and no link. (The visitor can log in directly from the index.aspx page if he or she chooses.)

Listing 15-9 contains the ASCX code for this user control. As with all the VB Snippets controls, the text strings are localized.

Listing 15-9. The u_crumb_welcome.ascx User Control

```
<%@ Control Language="vb" AutoEventWireup="false"
    Codebehind="u_crumb_welcome.ascx.vb"
    Inherits="Chapter_15.u_crumb_welcome"
    TargetSchema="http://schemas.microsoft.com/intellisense/ie5" %>
<p class="lfloatpad">
    <span class="crumb"><img border="0" src="images/arrow.gif"
        width="6" height="10"> 
        <%=rm.GetString("Welcome")%>,
        <span id="txtName" runat="server"></span>,
        <%=rm.GetString("toVbSnippets")%>
    </span>
</p>
<p class="rfloatpad" id="pLogout" runat="server">
    <a href="index.aspx?action=logout"
        class="logout"><%=rm.GetString("LogOut")%></a>
</p>
```

Notice how the logout link works. If the visitor clicks it, the control links back to index.aspx with a query string of `action=logout`. In fact, the logout link in the breadcrumbs custom control works the same way. The index.aspx page includes the following code at the top of its `Page_Load` procedure:

```
Dim sAction As String = Request.Params("action")
If sAction = "logout" Then
    Dim oVisitor As CVisitor = New CVisitor(Me)
    oVisitor.LogOut()
    Response.Redirect("index.aspx")
End If
```

The `Logout` method of the `CVisitor` class, which we present shortly, updates the database with current information and relinquishes the visitor's authentication. The `Response.Redirect` method redisplays the index.aspx page in its generic form.

The code in Listing 15-10 shows how the user control in the index.aspx page personalizes its message. If the `Page.User.Identity.IsAuthenticated` property is `True`, the control displays the `Page.User.Identity.Name` property value. This property returns the `Username` value supplied when the visitor is authenticated.

Listing 15-10. The u_crumb_welcome.ascx.vb User Control Code

```
Option Strict On
Imports System.Resources

Public MustInherit Class u_crumb_welcome
    Inherits System.Web.UI.UserControl
    Protected WithEvents pLogout As System.Web.UI.HtmlControls.HtmlGenericControl
    Protected WithEvents txtName As System.Web.UI.HtmlControls.HtmlGenericControl

    [Designer generated code omitted]

    Protected rm As ResourceManager

    Private Sub Page_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load
        'Put user code to initialize the page here
        rm = New ResourceManager("Chapter_15.strings", GetType(index).Assembly)

        If Page.User.Identity.IsAuthenticated Then
            'Get name and display logout option
            txtName.InnerText = Page.User.Identity.Name
            Me.pLogout.Visible = True
```

```
        Else
            'Visitor is not logged in
            txtName.InnerText = rm.GetString("friend")
            Me.pLogout.Visible = False
        End If
    End Sub
End Class
```

The control uses the `Visible` property to control whether ASP.NET generates the HTML for the logout link. If the visitor is logged in, the link is displayed; if not, the link is suppressed.

The authentication used to control the personalization of VB Snippets occurs in the `CVisitor` class. We present that class now.

Storing Visitor Information in Persistent Cookies

The `CVisitor` class contains information about a visitor to VB Snippets. Whenever a page instantiates the class, the class constructor checks to see whether the visitor has already been authenticated. Because the `Login` method creates a persistent authentication cookie, a visitor may already be logged in whenever he or she enters the Web site.

If the class constructor determines that the visitor is already authenticated, the code looks for an additional persistent cookie, `VbCodeDb`, that contains more information about the visitor. The `VbCodeDb` cookie contains the email address of the visitor, which acts as a key to the visitor's information stored in the VbUser database. It also tracks the date of the visitor's last use of VB Snippets. The code updates this date information at most once per day. VB Snippets uses the `LastVisit` property to determine which code snippets have been added since the last visit (and thus should be marked with the word *new*). The constructor initializes the readonly `Email`, `Username`, and `LastVisit` properties with values from the persistent cookies.

The VbUser database contains information about the visitor's keyword preferences. The `CVisitor` class constructor accesses these preferences via the `CVbUser` class and initializes the `CVisitor.Keywords` property accordingly. Listing 15-11 contains the code for the `CVisitor` class.

Listing 15-11. The `CVisitor` *Class*

```
Option Strict On
Imports System.Web.Security

Public Class CVisitor
    Private _Page As Page
    Private _dtmLastVisit As Date = #12/31/9999#
    Private _sEmail As String = String.Empty
    Private _sUsername As String = String.Empty
    Private _sKeywords As String = String.Empty

    Sub New(ByVal Page As Page)
        _Page = Page
        If _Page.User.Identity.IsAuthenticated Then
            Dim oCookieIn As HttpCookie = _Page.Request.Cookies("VbCodeDb")
            If Not oCookieIn Is Nothing Then
                _sEmail = oCookieIn.Values("Email")
                _dtmLastVisit = CDate(oCookieIn.Values("LastVisit"))

                'If new visit date, update cookie
                If oCookieIn.Values("CurrentVisit") _
                    <> Today.ToShortDateString Then
                    Dim oCookieOut As HttpCookie _
                        = _Page.Response.Cookies("VbCodeDb")
                    If Not oCookieOut Is Nothing Then
                        oCookieOut.Expires = Today.AddYears(1)
                        oCookieOut.Values("Email") = oCookieIn.Values("Email")
                        oCookieOut.Values("CurrentVisit") _
                            = Today.ToShortDateString
                        oCookieOut.Values("LastVisit") _
                            = oCookieIn.Values("CurrentVisit")
                        _dtmLastVisit = CDate(oCookieOut.Values("LastVisit"))
                    End If
                End If

                'Get preference info from database
                Dim oVbUser As CVbUser = Me.VisitorRow
                _sKeywords = oVbUser.Keywords
                oVbUser.Dispose()
```

```
        Else
            Throw New Exception("Chapter_15-CVisitor: " _
                & "Missing VbCodeDb Cookie")
        End If
    End If
End Sub

ReadOnly Property Email() As String
Get
    Return _sEmail
End Get
End Property

ReadOnly Property Username() As String
Get
    Return _sUsername
End Get
End Property

ReadOnly Property LastVisit() As Date
Get
    Return _dtmLastVisit
End Get
End Property

Property Keywords() As String
Get
    Return _sKeywords
End Get
Set(ByVal Value As String)
    _sKeywords = Value
    Dim oVbUser As CVbUser = Me.VisitorRow
    oVbUser.Keywords = _sKeywords
    oVbUser.Update()
    oVbUser.Dispose()
End Set
End Property
```

```
Function LogIn(ByVal Email As String, _
ByVal Password As String) As Boolean
    Dim oVbUser As New CVbUser()
    Dim EmailFound As Boolean = oVbUser.Find(Email)
    If EmailFound AndAlso oVbUser.Password = Password Then
        'Visitor is registered, get info
        _dtmLastVisit = oVbUser.LastVisit
        _sUsername = oVbUser.Username
        _sKeywords = oVbUser.Keywords
        oVbUser.Dispose()

        'Prepare persistent cookie
        Dim oCookie As HttpCookie = New HttpCookie("VbCodeDb")
        oCookie.Expires = DateTime.MaxValue
        oCookie.Values("Email") = Email
        oCookie.Values("LastVisit") = _dtmLastVisit.ToShortDateString
        oCookie.Values("CurrentVisit") = Today.ToShortDateString

        'Add cookies to response
        _Page.Response.Cookies.Add(oCookie)
        FormsAuthentication.SetAuthCookie(_sUsername, True)
        Return True
    Else
        'Couldn't find email-password combination
        Return False
    End If
End Function

Sub LogOut()
    'Update database with last visit date
    Dim oVbUser As CVbUser = Me.VisitorRow
    oVbUser.LastVisit = Today
    oVbUser.Update()
    oVbUser.Dispose()

    'Remove persistent cookies
    _Page.Request.Cookies.Remove("VbCodeDb")
    FormsAuthentication.SignOut()
    _sUsername = String.Empty
    _dtmLastVisit = #12/31/9999#
End Sub
```

```
    Private Function VisitorRow() As CVbUser
        Dim oVbUser As New CVbUser()
        Dim EmailFound As Boolean = oVbUser.Find(_sEmail)
        If EmailFound Then
            Return oVbUser
        Else
            Throw New Exception("Chapter_15-CVisitor: " _
                & "Missing VbUserDb Row")
        End If
    End Function
End Class
```

The `CVisitor.Login` method validates the visitor's registration, prepares the `VbCodeDb` cookie, and adds both the `VbCodeDb` cookie and the `FormsAuthentication` cookie to the response.

The `CVisitor.Logout` method updates the VbUser database, removes the `VbCodeDb` cookie, and calls the `FormsAuthentication.Signout` method to rescind the visitor's authentication. We have already presented the code, which is triggered by clicking a logout link and which calls the `Logout` method. The index.aspx page calls the `Login` method from the `Page_PreRender` event handler as follows:

```
If Me.smaccLogin.WasClicked Then
    'Visitor wants to log in
    Dim IsLoggedIn As Boolean = oVisitor.LogIn( _
        Me.txtEmail.Value, Me.txtPassword.Value)
    If IsLoggedIn Then
        'Return with authenticated credentials
        Response.Redirect("index.aspx")
    Else
        'Provide error message
        Me.spanErrMsg.InnerHtml = "<br /><font color='red'>" _
            & "Email or password not found.</font>"
    End If
End If
```

This code executes when the `PreRender` event fires to make sure that the MouseoverLink.vb custom control has had the opportunity to set its `WasClicked` property. If the login succeeds, the index.aspx redisplays in its personalized format. If not, the code adds a red error message.

Reading Database Information Efficiently

Although they are very useful for updating databases from Web applications, the disconnected datasets of ADO aren't needed by applications like VB Snippets that only display database information. (The VB Snippets database is maintained in a separate application.) To make such an application scale as well as possible, use *stored procedures* to define the rows needed from the database and use *data readers* to retrieve those rows for display.

You don't have to be a database expert to create stored procedures: Visual Studio .NET can do the work for you. Figure 15-16 shows the creation of the `BrowseAll` stored procedure using Visual Studio .NET's Query Builder. If you create and configure a data adapter to use a newly created stored procedure, the Query Builder does all the work. The Advanced settings allow you to stop the creation of stored procedures other than the `Select` procedure that you really need. You can discard the data adapter, and the stored procedures remain in the database for your use.

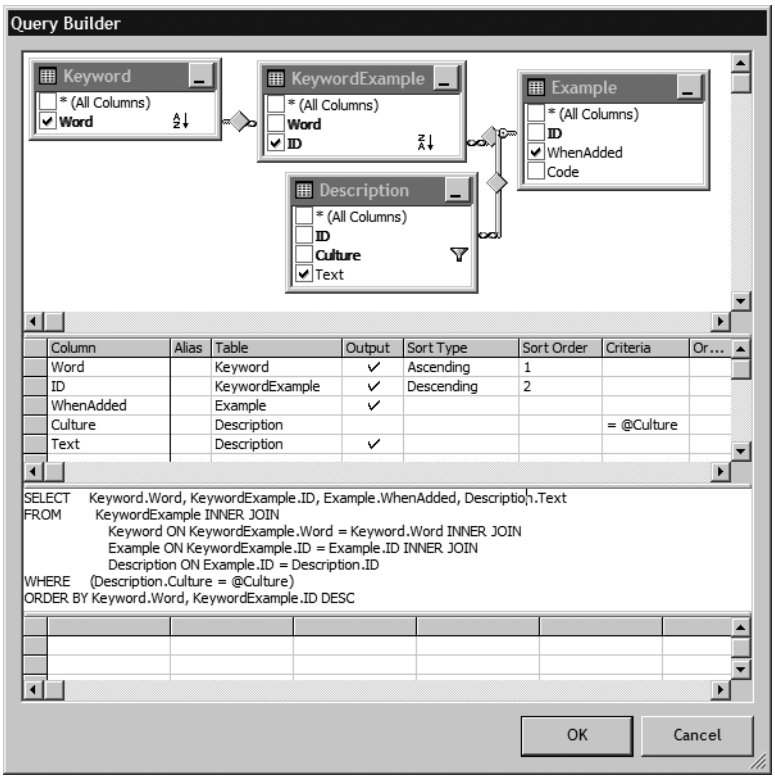

Figure 15-16. The Visual Studio .NET Query Builder creates a stored procedure with an inner join of four tables.

When you use a stored procedure, the database does not need to parse the SQL in a command to determine what rows to return. This naturally increases efficiency and improves the scalability of your Web application. When you execute a stored procedure, you need to know the parameters it requires, and your code must set the values correctly. The `BrowseAll` stored procedure created in Figure 15-16, for example, needs a parameter containing a valid culture.

To use a data reader, your code needs a data connection and a data command. The `ExecuteReader` method of the data command creates a data reader that retrieves database rows with great efficiency. One caveat: a data reader supports forward-only reading of the rows selected. Once you move past a row, you cannot get back to it without reading the database again. Make sure that your code takes this fact into account.

The `CVbCode` class uses stored procedures and data readers to access the information presented by VB Snippets. Let's take a look at that class.

Creating a Database Access Class

The `CVbCode` and `CVbUser` classes contain a lot of code, much of it repetitive. In this section, we show how to create a database access class for Web applications like VB Snippets, trusting that you can follow the same patterns in your own applications. You can, of course, download and examine both classes in detail.

The code in Listing 15-12 shows the `CVbCode` class in its very first version. This class returns the information needed to display a particular code snippet in the code_example page. This initial version of the class illustrates the common pattern used to access data using stored procedures and data readers.

Listing 15-12. The First Incarnation of the `CVbCode` *Class*

```
Option Strict On
Imports System.Data
Imports System.Data.Common
Imports System.Data.SqlClient
Imports System.Text

Public Class CVbCode
    Protected WithEvents SqlConnCode As SqlConnection
    Protected WithEvents SqlReadKeywords As SqlCommand
    Protected WithEvents SqlReadCode As SqlCommand
```

```
Public Structure Example
    Public Description As String
    Public WhenAdded As String
    Public Code As String
    Public KeywordLinks As String
End Structure

Public Function CodeExample(ByVal ID As Integer, _
Optional ByVal Culture As String = "en") As Example
    SqlReadCode.Parameters("@ID").Value = ID
    SqlReadCode.Parameters("@Culture").Value = Culture

    Try
        'Get requested code example from the database
        Dim RequestedExample As Example
        SqlConnCode.Open()
        Dim rdrEx As SqlDataReader = SqlReadCode.ExecuteReader()
        rdrEx.Read()
        RequestedExample.Description = rdrEx.GetString(0)
        RequestedExample.WhenAdded = rdrEx.GetDateTime(1).ToShortDateString

        'Format code example for Web display
        Dim sExample As New StringBuilder(rdrEx.GetString(2))
        sExample.Replace("  ", "  ")
        sExample.Replace("<", "&lt;")
        sExample.Replace(">", "&gt;")
        sExample.Replace(ControlChars.CrLf, "<br />")
        RequestedExample.Code = sExample.ToString
        rdrEx.Close()

        RequestedExample.KeywordLinks = KeywordLinks(ID)
        SqlConnCode.Close()
        Return RequestedExample
    Catch ex As Exception
        Throw New Exception("Chapter_15-CVbCode: " & ex.Message)
    End Try
End Function
```

```
Private Function KeywordLinks(ByVal ID As Integer) As String
    SqlReadKeywords.Parameters("@ID").Value = ID
    Dim rdrKey As SqlDataReader = SqlReadKeywords.ExecuteReader
    Dim sKeyword As String
    Do While rdrKey.Read
        If Not sKeyword = String.Empty Then sKeyword &= ", "
        sKeyword &= "<a href='code_browse_" & rdrKey.GetString(0).ToLower _
            & ".aspx'>" & rdrKey.GetString(0) & "</a>"
    Loop

    rdrKey.Close()
    Return sKeyword
End Function

Sub New()
    InitializeConnection()
    InitializeKeywordsCommand()
    InitializeCodeCommand()
End Sub

Private Sub InitializeCodeCommand()
    Me.SqlReadCode = New SqlCommand()
    With Me.SqlReadCode
        .CommandText = "[SelectExampleAndDescription]"
        .CommandType = System.Data.CommandType.StoredProcedure
        .Connection = Me.SqlConnCode
        .Parameters.Add(New System.Data.SqlClient.SqlParameter( _
            "@RETURN_VALUE", System.Data.SqlDbType.Int, 4, _
            System.Data.ParameterDirection.ReturnValue, False, _
            CType(0, Byte), CType(0, Byte), "", _
            System.Data.DataRowVersion.Current, Nothing))
        .Parameters.Add(New System.Data.SqlClient.SqlParameter( _
            "@ID", System.Data.SqlDbType.Int, 4, "ID"))
        .Parameters.Add(New System.Data.SqlClient.SqlParameter( _
            "@Culture", System.Data.SqlDbType.VarChar, 5, "Culture"))
    End With
End Sub

Private Sub InitializeKeywordsCommand()
    Me.SqlReadKeywords = New SqlCommand()
    With Me.SqlReadKeywords
        .CommandText = "[SelectKeywordsForExample]"
        .CommandType = CommandType.StoredProcedure
        .Connection = Me.SqlConnCode
```

```
        .Parameters.Add(New System.Data.SqlClient.SqlParameter( _
            "@RETURN_VALUE", System.Data.SqlDbType.Int, 4, _
            System.Data.ParameterDirection.ReturnValue, False, _
            CType(0, Byte), CType(0, Byte), "", _
            System.Data.DataRowVersion.Current, Nothing))
        .Parameters.Add(New System.Data.SqlClient.SqlParameter( _
            "@ID", System.Data.SqlDbType.Int, 4, "ID"))
    End With
End Sub

Private Sub InitializeConnection()
    Me.SqlConnCode = New SqlConnection()
    Dim sConnUser As String _
        = ConfigurationSettings.AppSettings("VbUserDbConn")
    If Not sConnUser = String.Empty Then
        'Get connection string from Web.config
        Me.SqlConnCode.ConnectionString = sConnUser
    Else
        'Use development connection
        Me.SqlConnCode.ConnectionString _
            = "data source=THOR;" _
            & "initial catalog=VbCode;" _
            & "persist security info=False;" _
            & "user id=sa;" _
            & "workstation id=THOR;" _
            & "packet size=4096"
        End If
End Sub

Sub Dispose()
    Me.SqlConnCode.Dispose()
    Me.SqlConnCode = Nothing

    Me.SqlReadCode.Dispose()
    Me.SqlReadCode = Nothing
    Me.SqlReadKeywords.Dispose()
    Me.SqlReadKeywords = Nothing

    Me.dt = Nothing
End Sub
```

```
    Protected Overrides Sub Finalize()
        If Not Me.SqlConnCode Is Nothing Then Dispose()
        MyBase.Finalize()
    End Sub
End Class
```

The `CVbCode` class constructor initializes the database connection and two `SqlCommand` objects. The `SqlReadCode` command uses the `SelectExampleAndDescription` stored procedure to retrieve rows from the VbCode database. Your code must supply values for the `@ID` and `@Culture` parameters, and the stored procedure returns the localized description of the code, the date the code was added to the database, and the code itself.

The `SqlReadKeywords` command uses the `SelectKeywordsForExample` stored procedure. Your code must supply a value for the `@ID` parameter, and the stored procedure returns all the keywords for that `ID`.

The `CodeExample` method returns a structure containing the snippet requested by `ID` and `Culture`. The structure includes the localized `Description`, the `WhenAdded` date (in string format), the actual `Code`, and a `KeywordLinks` string that contains the HTML for a list of links to the browse pages of every keyword associated with the snippet. Look back at Figure 15-13 to see how the code_example page displays the information received. The parameters received by the `CodeExample` method determine the values set for the parameters passed to the stored procedure by the command, as follows:

```
SqlReadCode.Parameters("@ID").Value = ID
SqlReadCode.Parameters("@Culture").Value = Culture
```

Note that the `CodeExample` method edits the code snippet to allow the browser to display the code correctly. Because the browser condenses whitespace into a single space, the method substitutes nonbreaking spaces. To cause the browser to display HTML in the code examples as text, the method substitutes entity references for the angle brackets used to form HTML tags. Finally, the `CodeExample` method substitutes `br` tags for the carriage return and linefeed combination (`CrLf`) that marks the end of each code line. Unlike strings, which are immutable in .NET, the `StringBuilder` class provides for the direct modification of string information. Therefore, the `CodeExample` method uses `StringBuilder` class methods to prepare the code for browser display.

NOTE *Despite all this editing, you can copy these snippets directly from the browser and paste them into Visual Studio .NET or into Notepad. The clipboard holds the copied data in both HTML and text formats, and the paste operation supplies plain text when you paste into Code view or into Notepad. The code will be correct, with the edits having been removed. If you paste into HTML view, on the other hand, you get the HTML version with the edits included.*

Creating the BrowseAll *Function*

To conclude our presentation of database access using stored procedures and data readers, we present a method that returns data that may be bound directly to a data grid, data list, or repeater. The `BrowseAll` function, as incorporated into the `CVbCode` class, uses the `BrowseAll` stored procedure (see Figure 15-16) to retrieve a large number of rows from the VbCode database. To support the `BrowseAll` function, we modified the class constructor to execute the `InitializeBrowseAllCommand` and `InitializeDataTable` procedures. Listing 15-13 contains additions to the `CVbCode` class to support the `BrowseAll` function.

Listing 15-13. The `BrowseAll` *Function and Supporting Code Added to the* `CVbCode` *Class*

```
Protected WithEvents SqlBrowseAll As SqlCommand
Private rm As ResourceManager
Private dt As DataTable

Public Function BrowseAll(Optional ByVal NewExampleDate As Date = #12/31/9999#, _
Optional ByVal Culture As String = "en") As ICollection
    dt.Clear()
    Dim dr As DataRow
    Dim dtmNewExample As DateTime = NewExampleDate
    Dim dtmCurrExample As DateTime

    'Set parameter values
    SqlBrowseAll.Parameters("@Culture").Value = Left(Culture, 2)

    'Get examples
    SqlConnCode.Open()
    Dim rdrAll As SqlDataReader = SqlBrowseAll.ExecuteReader()
```

```
        Do While rdrAll.Read
            dr = dt.NewRow
            dr("Keyword") = rdrAll.GetString(0)
            dr("ID") = rdrAll.GetInt32(1)
            dr("Link") = "code_example_" & rdrAll.GetInt32(1).ToString & ".aspx"
            dtmCurrExample = rdrAll.GetDateTime(2)
            dr("Date") = dtmCurrExample.ToShortDateString
            If dtmCurrExample > dtmNewExample Then
                dr("New") = rm.GetString("new")
            End If
            dr("Description") = rdrAll.GetString(3)
            dt.Rows.Add(dr)
        Loop

        rdrAll.Close()

        'Get keywords for each example
        Dim i As Integer
        For i = 0 To dt.Rows.Count - 1
            dt.Rows(i)("Keywords") = KeywordLinks(CInt(dt.Rows(i)("ID")))
        Next
        SqlConnCode.Close()

        Dim dv As New DataView(dt)
        Return dv
    End Function

    Private Sub InitializeBrowseAllCommand()
        Me.SqlBrowseAll = New SqlCommand()
        With Me.SqlBrowseAll
            .CommandText = "[BrowseAll]"
            .CommandType = System.Data.CommandType.StoredProcedure
            .Connection = Me.SqlConnCode
            .Parameters.Add(New System.Data.SqlClient.SqlParameter( _
                "@RETURN_VALUE", System.Data.SqlDbType.Int, 4, _
                System.Data.ParameterDirection.ReturnValue, False, _
                CType(0, Byte), CType(0, Byte), "", _
                System.Data.DataRowVersion.Current, Nothing))
            .Parameters.Add(New System.Data.SqlClient.SqlParameter( _
                "@Culture", System.Data.SqlDbType.VarChar, 5, "Culture"))
        End With
    End Sub
```

```
Private Sub InitializeDataTable()
    dt = New DataTable()
    dt.Columns.Add("ID", GetType(Integer))
    dt.Columns.Add("Link", GetType(String))
    dt.Columns.Add("Date", GetType(String))
    dt.Columns.Add("New", GetType(String))
    dt.Columns.Add("Description", GetType(String))
    dt.Columns.Add("Keyword", GetType(String))
    dt.Columns.Add("Keywords", GetType(String))
End Sub
```

The `BrowseAll` stored procedure takes an `@Culture` parameter and returns localized descriptions for all the code snippets in the database. In fact, this stored procedure returns *more* rows than the database has snippets because a separate row is generated for each keyword-example combination. A code snippet that's associated with three keywords, for example, appears in three separate rows returned by the stored procedure. Figure 15-17 shows the first page of the `BrowseAll` display. The code_browse_all Web page uses the paging features of the DataGrid control to reduce each page to a manageable size.

You may have noticed that the links for the code_example page actually look like this: code_example_13.aspx. Similarly, the links for the code_browse page look like this: code_browse_arraylist.aspx. The explanation for this fact brings us to the last topic of this chapter, and of this book.

Inviting Spiders, Crawlers, and Surfers

When you develop a Web site for public use, your objective is to attract visitors to your site. In many cases, the profitability of a Web site depends upon the number of visits it gets. Many surfers use search engines to locate Web pages of interest, and these folks are likely to miss your Web site entirely if the search engines don't list it prominently. Here we show a couple of techniques that you can use to encourage search engines to list your site prominently.

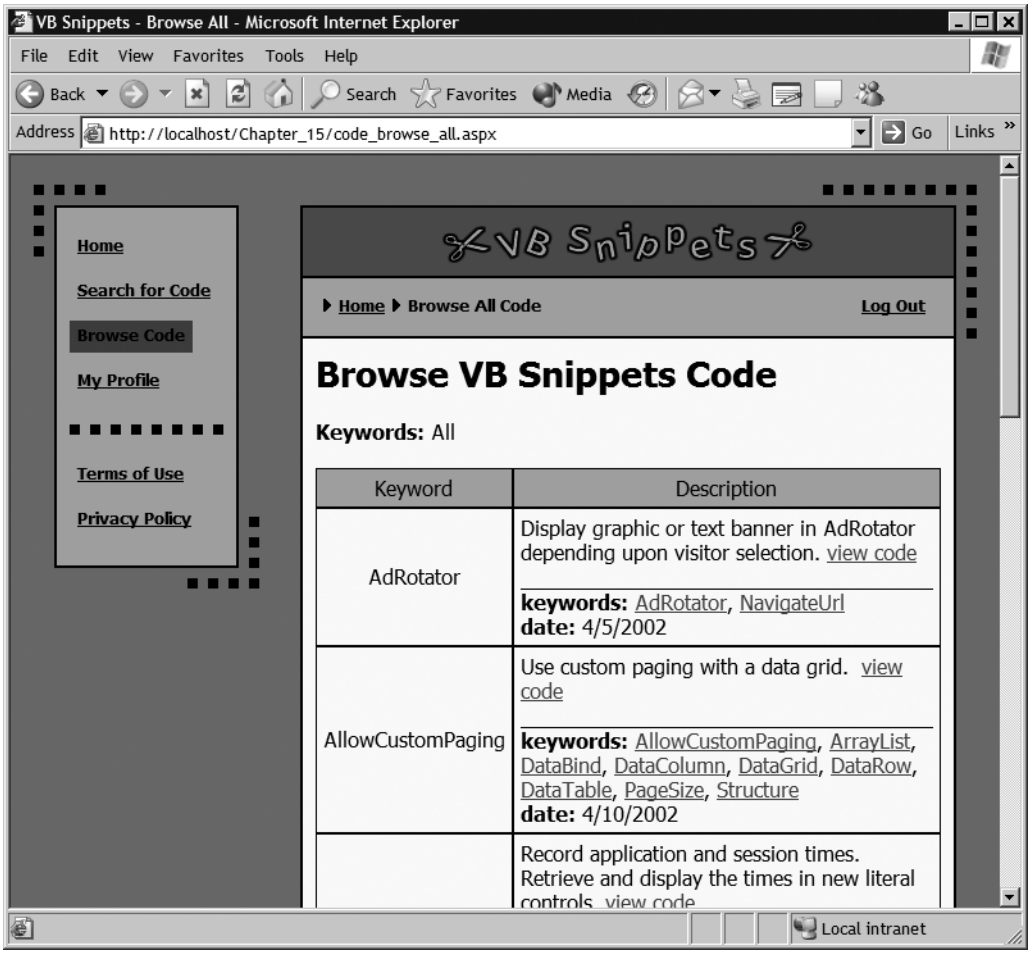

Figure 15-17. The code_browse_all page provides a means for a visitor to browse through all of the code snippets in a structured way.

Creating Web Pages on the Fly

What does this have to do with the names of your site's Web pages? Consider how the spiders and Web crawlers used by many search engines work: they happily follow link after link after link, recording information about the Web site's pages as they go. However, they most definitely *do not* make up values to enter into text-boxes to determine how a Web site responds, and they don't register themselves to view data that is unavailable to the general public. In fact, some search engines immediately stop crawling a site when they encounter a "?" in a URL's query string

to avoid the possibility of getting caught in a "spider trap"—a bit of dynamic code that requests information that the spider can't supply.

To help the spiders along, you want to make sure that every page in your Web site has its own URL, *even if the page does not exist until you create it*. Furthermore, you want to make sure that your Web site has several ways to link to every accessible page. By creating multiple links to the content-rich pages, you increase the odds that search engines will follow the links and find the most relevant pages of your Web site. In the VB Snippets case study, for example, many links exist to each code snippet.

So that explains why every code example and every browse page in VB Snippets has a different URL. The code in Listing 15-14 shows how VB Snippets analyzes the URLs and routes incoming requests to the code that creates those pages on the fly. The code that accomplishes this resides in the Global.asax file and executes before an ASPX file has been designated to respond to an incoming request. The `Application_BeginRequest` event handler contains the required code.

Listing 15-14. The `Application_BeginRequest` *Event Handler Routes Incoming Requests for Code Examples and Browse Pages.*

```
Sub Application_BeginRequest(ByVal sender As Object, ByVal e As EventArgs)
    ' Fires at the beginning of each request
    Const sCodeReq As String = "code_example_"
    Const sBrowse As String = "code_browse_"
    Dim sPath As String = Request.Path.ToLower
    Dim iLength As Integer = 0

    Dim iFirstDigit As Integer = sPath.IndexOf(sCodeReq)
    If iFirstDigit > -1 Then
        'Code example page name detected
        iFirstDigit += sCodeReq.Length
        iLength = sPath.IndexOf(".") - iFirstDigit
        If iLength > 0 Then
            'Code example request: convert request to parameters and create page
            Try
                'Find language directory, if any
                Dim asLang() As String = {"/es/", "/fr/", "/pt/", "/de/"}
                Try
                    asLang _
                      = ConfigurationSettings.AppSettings("UrlLangs").Split(",c)
                Catch
                    'Retain default
                End Try
```

```
                    Dim sLang As String = String.Empty
                    Dim i, iSlash As Integer
                    For i = 0 To UBound(asLang)
                        iSlash = sPath.IndexOf(asLang(i))
                        If iSlash > -1 Then
                            sLang = asLang(i).Substring(1, asLang(i).Length - 2)
                            Exit For
                        End If
                    Next

                    Dim iExampleID As Integer _
                        = CType(sPath.Substring(iFirstDigit, iLength), Integer)
                    Dim strNewPath As String = Request.ApplicationPath _
                        & "/code_example.aspx?exampleid=" & iExampleID.ToString
                    If sLang <> String.Empty Then
                        strNewPath &= "&language=" & sLang
                    End If

                    Me.Context.RewritePath(strNewPath)
                Catch
                    'On conversion error, do not rewrite path
                End Try
                Exit Sub
            End If
        End If

        Dim iFirstChar As Integer = sPath.IndexOf(sBrowse)
        If iFirstChar > -1 Then
            'Browse page detected
            iFirstChar += sBrowse.Length
            iLength = sPath.IndexOf(".") - iFirstChar
            Dim sKeyword As String = sPath.Substring(iFirstChar, iLength)
            If sKeyword <> "all" Then
                'Specific keyword browse, so create page
                Me.Context.RewritePath(Request.ApplicationPath _
                    & "/code_browse.aspx?keyword=" & sKeyword)
            End If
        End If
    End Sub
```

In a nutshell, the code in Listing 15-14 scans the URLs of incoming requests to determine if the URL includes “code_example_” or “code_browse_”. If not, the code does nothing and the requested page—index.aspx, for example—responds as usual. If a match is found, however, the event handler goes to work.

When the URL is for a particular code example, the event handler finds the example number, which corresponds to its `ID` in the VbCode database, and inserts into a query string like this:

```
?exampleid=13
```

The event handler uses the `Context.RewritePath` method to route the incoming request to code_example.aspx, with the query string appended. The code_example page uses the query string to determine what example has been requested and builds the response on the fly. When ASP.NET returns the response to the visitor, however, the URL in the browser's address line reflects the original request, not the page that actually created the response. To all appearances, the visitor requested a Web page with an independent existence, and the Web site returned that page.

If the request is for a code example with a non-English description, the URL includes the culture code, like this:

```
.../es/code_example_13.aspx
```

When the URL contains a recognized culture code, the event handler extracts it and sends it to code_example.aspx in a separate query string.

The code_browse processing works similarly. In this case, however, the event handler extracts the keyword from the URL and passes it to code_browse via a query string.

The `Request` object that's available to your `Application_BeginRender` code provides access to specific information about the visitor's browser. If your Web site provides different versions of Web pages for different browsers, your code can use the `Context.RewritePath` method to route an incoming request to the page specifically designed to handle it. From the visitor's viewpoint, the Web application simply returns the page requested, regardless of the ASPX file that actually responded to the request.

Adding Meta-Keywords Dynamically

To improve your Web site's ranking by search engines, you should carefully choose the keywords you provide in the head section of the HTML for each Web page. Your goal should be to include every word that might be entered by a potential visitor who is searching for sites like yours. When you build Web pages on the fly, you should consider adding pertinent meta-keywords on the fly. The code_browse page uses the u_keywords.ascx user control for precisely this purpose. The u_keywords.ascx file contain a single literal control, like this:

```
<asp:literal id="litKeywords" runat="server"></asp:literal>
```

The code behind the user control contains a static list of keywords that are applicable to every page in the site, and it exposes an `Append` property used to specify the word or words to add to the list. When the control is rendered, the static list and the appended word or words are included in a `meta` tag in the `head` section of the HTML. Listing 15-15 contains the code for this user control

Listing 15-15. The u_keywords.ascx.vb User Control Code

```
Public MustInherit Class u_keywords
    Inherits System.Web.UI.UserControl
    Protected WithEvents litKeywords As System.Web.UI.WebControls.Literal

    [Designer generated code omitted]

    Private Sub Page_Load(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles MyBase.Load
        'Put user code to initialize the page here
        Me.litKeywords.Text = sMeta & _sAppend & sQuote & ">"
    End Sub

    Const sQuote As String = """
    Const sMeta As String = "<meta name=" & sQuote & "keywords" _
        & sQuote & " content=" & sQuote _
        & "code, sample code, example code, Web code, Web database, " _
        & "Web database code, Visual Basic, Visual Basic .NET, " _
        & "Visual Studio, Visual Studio .NET"

    Private _sAppend As String

    Property Append() As String
    Get
        Return _sAppend
    End Get
    Set(ByVal Value As String)
        _sAppend = Value
    End Set
    End Property
End Class
```

To support the dynamic addition of keywords to the `meta` tag, the code_browse.aspx file contains this statement in the `head` section of its HTML:

```
<sma:keywords id="smaKey" runat="server"></sma:keywords>
```

The following code in the `Page_Load` procedure appends the current browse keyword to the static keywords rendered in the `meta` tag:

```
Dim ouKey As u_keywords
ouKey = CType(Me.FindControl("smaKey"), u_keywords)
ouKey.Append = ", " & sKeyword
```

This technique can also be used to modify other HTML elements that are not displayed in the browser window. As a matter of fact, the code_browse page modifies the title element on the fly, by incorporating the browse keyword into the title that's displayed at the top of the browser.

Recap

This chapter covered the development of a high-quality Web site based on a professionally created Web design. We discussed the benefits of factoring a Web application into reusable controls and classes and provided examples of each. The code examples in this chapter showed how to output localized HTML for navigation menus, breadcrumb links, and graphical buttons.

This chapter also demonstrated how to use persistent cookies to personalize a Web site. It concluded by showing how to create Web pages on the fly that look and behave like static pages.

This chapter concludes our book about programming the Web using Visual Basic .NET. We hope, someday soon, to visit *your* applications on the Web. Happy programming!