

Bonus Chapter 1

VBA Programming in FrontPage

In This Chapter

- ▶ Understanding how VBA can make FrontPage better
 - ▶ Working with FrontPage-related objects
 - ▶ Creating FrontPage documents
 - ▶ Developing FrontPage templates with automation
 - ▶ Designing your own FrontPage template
-

FrontPage is the Microsoft Office Web page design tool. It provides some of the more interesting situations in which you can use VBA to add new features or provide increased functionality. Because Web pages are essentially pure text, you can do a lot more with them.

The VBA environment for FrontPage is unique in that the programs you create are part of the application and not the document. For example, when you create a VBA program for Word, you can assign that program to either the document or its associated template. Excel and Access both associate the programs that you write with the document (database). Consequently, every program that you write for FrontPage is accessible to every document. Think about FrontPage macros as being global, akin to writing macros for Word's Normal.dot or Normal.dotm template.

Using FrontPage with VBA

In this chapter, you discover the various FrontPage-related objects that are available in VBA. The FrontPage objects relate to FrontPage itself rather than to a particular document. However, that doesn't mean that you can't create some interesting Web pages by using VBA. The following list describes some of the ways in which you can use VBA to make FrontPage significantly better:

- ✓ Create new document types as well as add to existing documents.
- ✓ Enhance templates and provide specialized formatting in your FrontPage documents.

- ✔ Create an automated code designer for your Web page. The automation might add common elements that you can't easily add by using templates. For example, a header or footer might contain common elements, but these elements might be in a slightly different position based on the kind of page you create.
- ✔ Define code snippets for common tasks. For example, you might want to automate the creation of `<meta>` tags for your Web page.
- ✔ Track team projects with greater accuracy. You can use VBA to record changes to a Web page automatically and to add documentation entries that include the user's login name.
- ✔ Implement a check-in and check-out system. Depending on the size of your organization, using a good source code product might be necessary, but smaller organizations can often make do with something simpler.
- ✔ Automatically configure your environment based on the project. One of the issues of working with FrontPage is that it doesn't provide some of the developer automation provided with other products. VBA can make the automation gap smaller.

Understanding the FrontPage-Related Objects

FrontPage provides a number of objects that you can use to interact with the program and documents. You can perform any task, from creating new documents to listing the templates installed in the current machine, by using these objects. However, because FrontPage doesn't associate your program with any particular document, you must either provide additional code to check for specific documents or write the code to work with any document.

FrontPage works with several libraries. In addition to the standard Office, VBA, and StdOLE (Standard Object Linking and Embedding) libraries, a minimal FrontPage setup also includes the FrontPage, FrontPageEditor, and Microsoft FrontPage libraries. Each of these libraries works with major FrontPage object groups. Microsoft groups the FrontPage objects into two major categories: `Page` and `FrontPage`.

The `Page` objects affect individual documents directly. These objects appear in the FrontPageEditor library. The Microsoft documentation says that these objects work with Internet Explorer 4.0 or above, but you can make the objects work with other browsers by employing careful testing. You can find a detailed list of these objects at <http://msdn.microsoft.com/library/en-us/vbafpd10/html/fphowExplorePOM.asp>.

The `FrontPage` objects affect the application, the application environment, and the user. For example, this is where you find the `CommandBars` collection used to change the application toolbars. (See the “Manipulating Toolbars and Menus” section of Chapter 12 for details.) You can find a hierarchical chart of these objects at <http://msdn.microsoft.com/library/en-us/vbafpw10/html/fptocObjectModelApplication.asp>.



Normally, you work with `FrontPage` using the older toolbar-and-menu approach because Microsoft didn't upgrade this product for Office 2007. However, when interacting with an Office 2007 product, you may need to consider the Ribbon as part of your code. Make sure that you understand the comparison between the old and new user interfaces, as described in Chapter 12, before you begin writing an application that interacts with applications such as Word, Access, Excel, Outlook, or PowerPoint.

Using the Application object



You use the `Application` object to access most application features, such as the product name and version. This object also contains information about the user, such as the username and organization. Finally, you use this object to access information about the current document, including formatting and content. Listing BC1-1 shows some of the ways that you can use the `Application` object. (You can find the source code for this example on the [Dummies.com](http://www.dummies.com/go/vbafpd5e) site at <http://www.dummies.com/go/vbafpd5e>.)

Listing BC1-1 Using the Application Object

```
Public Sub GetAppStats()
    ' Contains the application information.
    Dim Output As String

    ' Get the application statistics.
    With Application
        Output = Output + .UserName + vbCrLf
        Output = Output + .OrganizationName + vbCrLf
        Output = Output + .Name + vbCrLf
        Output = Output + .Version + vbCrLf + vbCrLf

        ' Get some of the active document information.
        With ActiveDocument
            Output = Output + "Active Document" + vbCrLf
            Output = Output + vbCrLf + .nameProp + vbCrLf
            Output = Output + .DocumentHTML
        End With
    End With

    ' Display the output.
    MsgBox Output, vbInformation, "Application Statistics"
End Sub
```

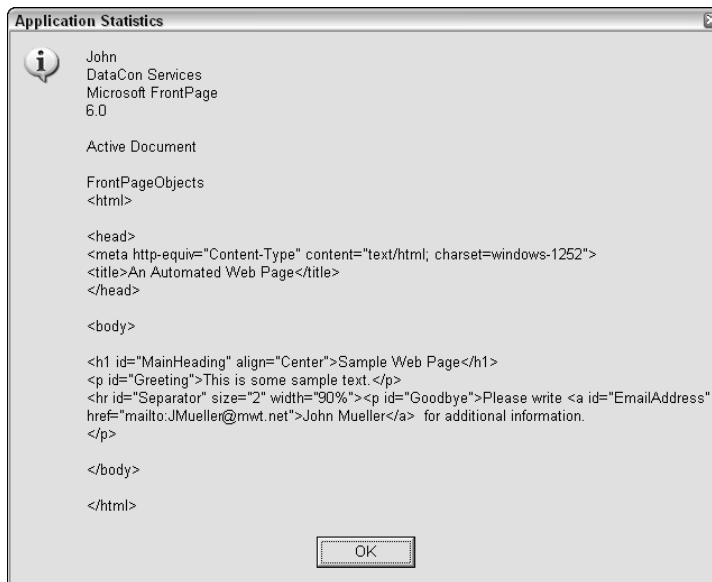
The code in this example begins by working with the `Application` object properties. You can get the user's name and organization to verify identity, or at least configuration. This information is suspect because it depends on the user entering the correct information during installation. In addition, someone else might actually use the software or log in under the registered user's name. However, it's one check that you can perform.

The `Name` and `Version` properties identify the product. This information is always correct because the product generates it for you. You can also get product-specific information, such as the product code.

Notice how the program uses nested `With` statements in this example. The `ActiveDocument` `With` statement is actually nested within the `Application` `With` statement, so you would read the internal statements as `Application.ActiveDocument` and not just `ActiveDocument`. Exercise caution when using nested `With` statements because you can confuse a property or method at one level with a property or method of the same name at another level, thus resulting in bugs that are extraordinarily difficult to find.

The `ActiveDocument` object contains a number of interesting properties and methods, many of which appear in the remaining examples in this chapter. The `nameProp` property indicates the active document name, and the `DocumentHTML` property contains the complete HyperText Markup Language (HTML) for the document. Figure BC1-1 shows the output from this program.

Figure BC1-1:
Listing application, user, and document information.



Using the FrontPageEditor (Page) objects

The `FrontPageEditor` and `Page` objects (Microsoft uses both terms to refer to the same object class) are the most useful FrontPage elements that you can discover. You use these objects to create Web pages. Any element that you can add to a Web page is also accessible as a `Page` object.



Unfortunately, the documentation for this set of objects is a little skimpy if you want to use the FrontPage-specific objects, even if you look online at <http://msdn.microsoft.com/library/en-us/vbafpd10/html/fphowFPSpecMethods.asp>. The secret is to look at the associated Internet Explorer interface elements at <http://msdn.microsoft.com/workshop/browser/mshtml/reference/ifaces/interface.asp>. For example, if you want information about the `FPHTMLHeaderElement`, look at the `IHTMLHeaderElement` documentation instead. You can also use the `IHTMLHeaderElement` object directly.



Ultimately, you can build any kind of Web page that you want. The Web page can use straight HTML tags or incorporate cascading style sheets (CSS). A CSS is a technique used for separating the format of information from the actual content on Web pages to make it easier to use and also make it more accessible to those users with special needs. Listing BC1-2 shows one way to use these objects. (You can find the source code for this example on the Dummies.com site at <http://www.dummies.com/go/vbaf5e>.)



Vista and help files

You might find it odd that you can't get help even though the application provides it, but that's the very problem you might encounter when working with FrontPage. I'm writing this book based on the Vista Release Candidate (which is still in beta). Vista doesn't currently support the older HLP (help) file format. It does support newer versions of HTML help, but not the older file format.

What this change in support means to you is that you might experience problems in getting access to help in some versions of FrontPage because this product uses HLP files. Vista displays what appears to be a non-helpful message that indicates some problem with help. What it's really telling you is that you can't find

help for FrontPage even though the files are installed on your system.

Fortunately, you don't really have to do without the help resources you require. You can find what you need on the Microsoft Web site. Look for general FrontPage help at <http://msdn.microsoft.com/office/program/frontpage/2003/getstarted/>. Office developer references appear at <http://msdn.microsoft.com/office/reference/default.aspx>, and you can find a FrontPage-specific page at <http://msdn.microsoft.com/office/program/frontpage/2003/>; these two Web pages include a link to the VBA references you need for FrontPage.

Listing BC1-2 Automating Web Page Creation

```
Public Sub ChangePage()  
  
    ' Create the Web page elements.  
    With Application.ActiveDocument  
  
        ' Create a heading.  
        Dim Heading As FPHTMLHeaderElement  
        Set Heading = .createElement("H1")  
        With Heading  
            .Id = "MainHeading"  
            .innerText = "Sample Web Page"  
            .Align = "Center"  
        End With  
  
        ' Create some text.  
        Dim Greeting As FPHTMLParaElement  
        Set Greeting = .createElement("P")  
        With Greeting  
            .Id = "Greeting"  
            .innerText = "This is some sample text."  
        End With  
  
        ' Create a horizontal line.  
        Dim Separator As FPHTMLHRElement  
        Set Separator = .createElement("HR")  
        With Separator  
            .Id = "Separator"  
            .Size = "2"  
            .Width = "90%"  
        End With  
  
        ' Create a combined element.  
        Dim Contact As FPHTMLParaElement  
        Dim EmailAddr As FPHTMLAnchorElement  
        Set Contact = .createElement("P")  
        Set EmailAddr = .createElement("a")  
        With EmailAddr  
            .Id = "EmailAddress"  
            .href = "mailto:JMuedler@mwt.net"  
            .innerText = "John Mueller"  
        End With  
        With Contact  
            .Id = "Goodbye"  
            .insertAdjacentHTML "afterBegin", _  
                "Please write " + EmailAddr.outerHTML + _  
                " for additional information."  
        End With  
  
        ' Change the Web page title.
```

```

.Title = "An Automated Web Page"

' Design the Web page content.
.body.insertAdjacentHTML "afterBegin", _
    Heading.outerHTML + Greeting.outerHTML + _
    Separator.outerHTML + Contact.outerHTML
End With
End Sub

```

All the objects in this section follow the same set of creation steps. The code begins by defining the object. Make sure that you use the correct type because your code will fail otherwise. Next, the code calls the `createElement` method. The string that you provide is critical. For example, make sure that you supply `H1` as input when you want to create a level 1 header. After the code creates the element, it defines the necessary properties. Make sure that you include an `Id` property value because this value makes it easier to work with the objects later by giving you a reference name.



FrontPage and the Vista User Account Control (UAC)

FrontPage, more than most Office products, appears to run afoul of the Vista User Account Control (UAC), a security feature designed to make it harder for evil people to access your system. Unfortunately, this feature also makes it harder for you to access your system. You may find that you can't even save your edits, so check security before you use FrontPage under Vista the first time. Try to make a small, single character and then edit and save it. Run the Web site to ensure that it works. Test before you make any major edits.

If you have problems, make sure that you give yourself permission to access all required directories on your system. FrontPage users often need access to more of the hard drive than someone using another product, such as Word. When working with a network drive, make sure that you have the proper rights to that drive. In addition, set the zone for the network drive by using the Internet Properties

applet of the Control Panel. The default setting is for the Internet. Select Trusted Sites and click Sites. Type **file://** plus the name of your server in the Add This Website to the Zone field and click Add. You should have access at this point.

Vista also has a significantly enhanced firewall. Make sure that you set an exclusion for your FrontPage activities. This may mean using the settings in the Windows Firewall with Advanced Security applet of the Control Panel.

When all else fails, you may have to change the local security policy to make UAC less restrictive. You don't want to take this step lightly because UAC has a definite purpose in protecting your system. Use the Local Security Policy console, in the Administrative Tools folder of the Control Panel, to make the required changes. The UAC settings appear at the end of the list in the `Local Policies\Security Options` folder.

Notice that you can combine elements. The Contact object contains text and the EmailAddr object. You combine elements by using the `insertAdjacentHTML` method, which requires a location and the text value as input. The code uses the `EmailAddr.outerHTML` property because it contains the full HTML tag for the object.

As with all HTML documents, you can access the `<head>`, `<title>`, and `<body>` tags from VBA. These essential tags give the HTML document structure. The example shows how to modify the `<title>` and `<body>` tag content. You set the `Title` property directly. The `body` property requires use of the `insertAdjacentHTML` method. Only after the code sets the `body` property does the content that you've created appear in the FrontPage editor, as shown in Figure BC1-2.

Figure BC1-2 shows that the output of this program is well formed (complies with all the required Web standards) and complete (no missing tag elements). Using this technique can help you automate some of the page creation tasks. This technique is especially helpful when the Web page follows the same basic format each time but can contain variable elements. VBA is a lot more flexible than using templates to create variable content. Of course, you can get the best of both worlds by combining VBA and templates.

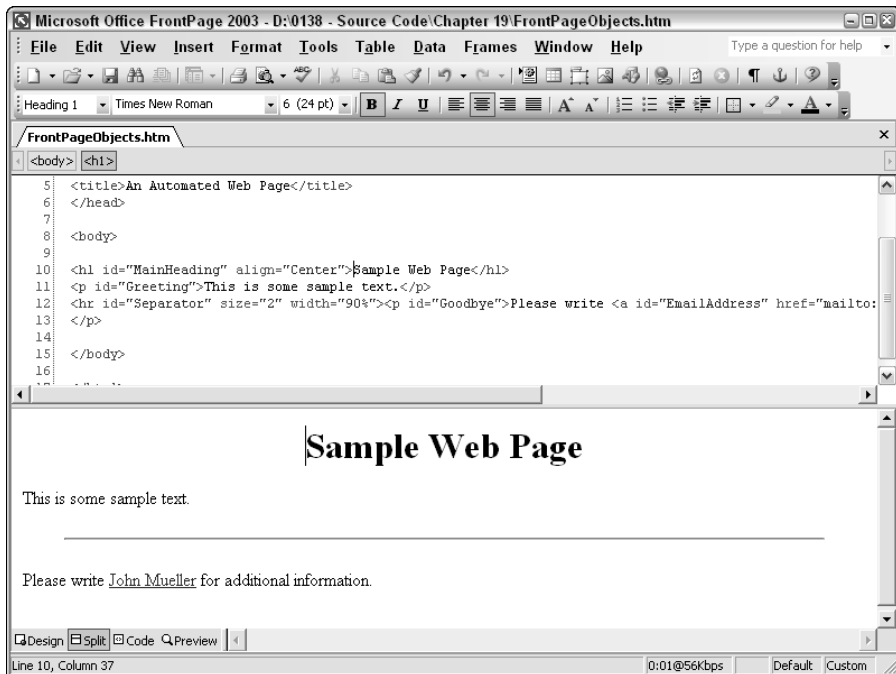


Figure BC1-2: Listing application, user, and document information.

Understanding the Themes collection



FrontPage comes with a wealth of *themes* (files with configuration information that helps you design a project in less time because some features, such as background and colors, are predefined for you). You can create your own themes or download themes created by other people. For example, you can find free themes at FrontPage 2002.com (<http://www.frontpage2002.com/>) and Theme Mart (<http://www.thememart.com/>). Because you could have so many themes to track, it's important to know how to list them. Listing BC1-3 shows one way to perform this task by using the Themes collection. (You can find the source code for this example on the Dummies.com site at <http://www.dummies.com/go/vbafd5e>.)

Listing BC1-3 Accessing FrontPage Theme Information

```
Public Sub ThemeLister()
    ' Holds an individual theme.
    Dim ThisTheme As Theme

    ' Holds the theme list.
    Dim ThemeArray(2, 100)

    ' Keeps track of the current theme number.
    Dim Counter As Integer
    Counter = 0

    ' Clear any existing list items.
    MyThemeList.lbThemes.Clear

    ' Get the theme list.
    For Each ThisTheme In Application.Themes
        ThemeArray(0, Counter) = ThisTheme.Label
        ThemeArray(1, Counter) = ThisTheme.Name
        ThemeArray(2, Counter) = ThisTheme.Format
        Counter = Counter + 1
    Next

    ' Add the theme list to the list box.
    MyThemeList.lbThemes.Column() = ThemeArray

    ' Display the list.
    MyThemeList.Show
End Sub
```

This example begins by creating a single Theme object used to hold an individual theme, an array to hold the Theme values, and a counter to track which theme is in use. The MyThemeList object is a UserForm used to display the output from this example. You can also use a message box, but the number of values makes a UserForm easier to work with in this case.

After the code creates the essential objects, it builds a list of themes. You might wonder why I didn't use a `For . . . Next` loop rather than the `For Each . . . Next` loop shown. Attempting to use an index to access the individual `Theme` objects in the `Themes` collection causes FrontPage to crash in some cases. The method shown always works.

Notice that the code gets the `Label`, `Name`, and `Format` properties from the `ThisTheme` object. You might also want to view the `Version` or other properties, but these three properties are all that you need in most cases.



The `Format` property contains the version of the theme and not the version of FrontPage. Most recent themes use version 2.0. When you download a FrontPage 98 theme, the version number is usually 0.0 or 1.0. Older themes might not have the same functionality that newer themes provide. Make sure that you use version 2.0 or newer themes to ensure maximum browser and FrontPage compatibility.



The `Label` and `Name` properties provide essentially the same information. The `Label` property contains the friendly version of the name. Use the `Label` property when you want to create lists for users and the `Name` property when you want to create lists for selecting a specific theme for a Web page. Combine both fields (keeping the `Name` field hidden) when you need to ask the user which theme to select; then use that information to add the theme to the current Web page. You can keep the field hidden by setting its column width to 0. Here's an example of how you can hide the `Name` property for this example:

```
MyThemeList.lbThemes.ColumnWidths = "2in;0;"
```

Notice that you can include a measurement with the column width. FrontPage normally uses points for column measurements, but including a measurement overrides this feature. In this example, the first column is set to 2 inches, the second is hidden, and the third uses the remaining space.

After the code creates an array containing the values, it adds this list to the `MyThemeList` list box. Use the `Column` versus the `List` property for this kind of array. (For more about arrays, check out Chapter 9 of this book.) The `List` property transposes the elements and displays the information incorrectly. The `MyThemeList.Show` method displays the form. Figure BC1-3 shows how the form looks with all three columns in place.

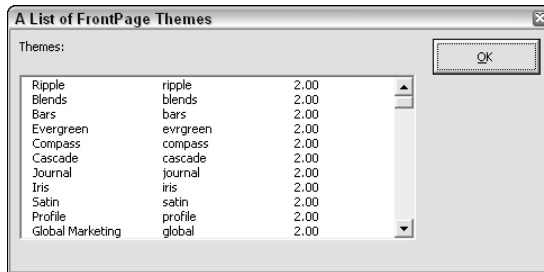
Understanding the `Webs` collection



The `Webs` collection provides you with access to your Web site. The Web site can be a remote location or a local hard drive. Any time that you open a site in FrontPage, you create a new `WebEx` object that FrontPage places in the `Webs` collection. However, as with any other collection that you might use,

you can also add new Web sites to your FrontPage setup programmatically by creating a new WebEx object by using the `Webs.Add` method. Listing BC1-4 shows how you can create a new WebEx object and then garner statistics and information from it. (You can find the source code for this example on the Dummies.com site at <http://www.dummies.com/go/vbafd5e>.)

Figure BC1-3:
Listing themes.



Listing BC1-4 Working with the Webs Collection

```
Public Sub DisplayWebs()
    ' Holds an individual Web object.
    Dim AWeb As WebEx

    ' Holds an individual folder.
    Dim AFolder As WebFolder

    ' Contains the output information.
    Dim Output As String

    ' Define a Web object.
    Application.Webs.Add "D:\My Web Site"
    Application.Webs.Open "D:\OnlineSite"

    ' Display some statistics.
    For Each AWeb In Application.Webs

        ' Get the site name.
        Output = AWeb.Title + vbCrLf + vbCrLf

        ' Parse the folder list.
        For Each AFolder In AWeb.AllFolders
            Output = Output + AFolder.Name + vbCrLf
        Next

        ' Display the results.
        MsgBox Output, vbInformation, "Web Statistics"
    Next
End Sub
```

This example begins with FrontPage as you initially open it. There are no Web sites open, so the code begins by opening two Web sites. Notice that one call uses the `Add` method, and the second uses the `Open` method. You can use either method when working with an existing Web site. However, if you're converting a folder to a Web site or establishing a new Web site, use the `Add` method. Note that you need to change the locations shown in the example code to match Web sites on your local system (unless you add these folders to your machine).

After the code has some Web sites to work with, it uses the `Webs` collection to access individual `WebEx` objects. Notice that this is one case where the individual object has a slightly different name from the collection that it supports. Each `WebEx` object contains information about the individual Web site, including a list of files and folders. You can also apply themes and templates, search for files or folders, and set viewing options, such as hiding hidden files and folders.

The code uses the `AllFolders` collection to get a list of all the folders that the Web site contains. Each `WebFolder` object contains information about a single folder and any files and folders that it contains. You can also use the `RootFolder` object or `Folders` collection to access folders level by level (rather than all at once). In this case, the code records just the folder name. Figure BC1-4 shows the output from this program. (Your display will differ from mine unless your Web site is precisely the same as mine.)



Figure BC1-4:
Listing folders on a Web site.

Theoretically, you can also open a remote site by using the `Add` or `Open` method of the `Webs` collection by using a statement such as the following:

```
Application.Webs.Open "ftp://ftp.mysite.net/" , "myname", "mypassword"
```

You can certainly open a remote site by using manual methods from within FrontPage. However, in practice, FrontPage seems to ignore `Add` and `Open` method calls that contain remote locations. Interestingly enough, all the Microsoft examples show local drives as the location. When you open a remote site, the FrontPage display changes, as shown in Figure BC1-5.

FrontPage considers a remote location as a combination of a local and a remote Web site. Consequently, every time that you open a remote location, you add two `WebEx` objects to the `Webs` collection. The `DisplayWebsNoOpen` sub-procedure supplied with the code example lets you test a remote Web site by using display code similar to the code shown in Listing BC1-4.

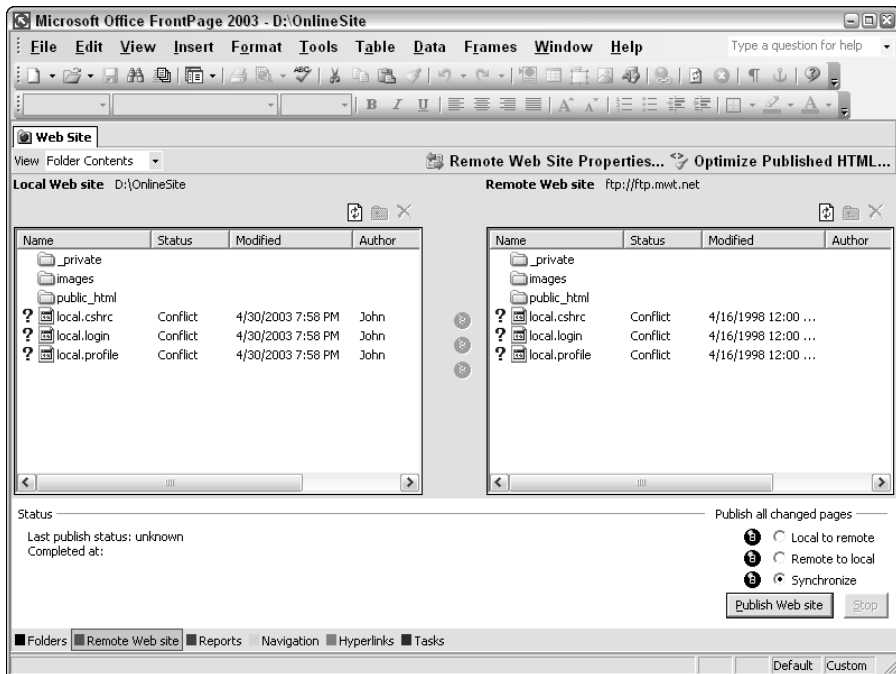


Figure BC1-5:
A view of a remote Web site.

Understanding the WebWindows collection



The `WebWindows` collection contains one `WebWindowEx` object entry for every FrontPage window that you have open. FrontPage associates every window with a single Web site. Look again at Listing BC1-4, and you'll notice that the code opens two Web sites, which means that FrontPage opened two `WebWindowEx` objects. Every time that you open a file located in the current Web site or create a new file, FrontPage also creates a `PageWindowEx` object that it places in the `PageWindows` collection. All these entries help you track the status of the open Web sites that FrontPage is managing.



Understanding the relationships between these various objects is important because you have to move from one to the next in a logical order. Listing BC1-5 demonstrates how the various collections and objects interact. Because you can use index values with the collections, you can access a particular object quickly. (You can find the source code for this example on the Dummies.com site at <http://www.dummies.com/go/vbafd5e>.)

Listing BC1-5 Working with the Webs Collection

```
Public Sub ListWebWindows()  
    ' Holds an individual Web window.  
    Dim AWindow As WebWindowEx  
  
    ' Holds an individual Page window.  
    Dim APage As PageWindowEx  
  
    ' Contains the output information.  
    Dim Output As String  
  
    ' View each of the windows in turn.  
    For Each AWindow In Application.WebWindows  
  
        ' Get the window information.  
        Output = Output + AWindow.Caption + vbCrLf  
  
        ' Display the Web site associated with this window.  
        Output = Output + vbTab + AWindow.Web.Title + vbCrLf  
  
        ' View each of the pages in turn.  
        For Each APage In AWindow.PageWindows  
            Output = Output + vbTab + APage.Caption + vbCrLf  
        Next  
    Next  
  
    ' Display the results.  
    MsgBox Output, vbInformation, "Web Window Statistics"  
End Sub
```

The code begins by creating the various objects needed for this example. Notice that the names don't precisely correspond to the usual naming conventions for collections and associated objects.

The first level of access is the `WebWindows` collection. The code uses this collection to get a single `WebWindowEx` object. A `WebWindowEx` object contains all the content for a single window, such as the one shown in Figure BC1-5.

After the code gets the `WebWindowEx` object, it uses the `Web` object to determine the name of the Web site associated with the window. You have full access to all Web site information through the `Web` object and can follow this object down to locate both folders and files.

The `WebWindowEx` object also contains the `PageWindows` collection. Every open file tab in a window appears within this collection as a `PageWindowsEx` object. The code uses the `Caption` property to get the name of the file. The final step is to output this information. Figure BC1-6 shows the results.

Figure BC1-6:
The relationship between Web windows, pages, and sites.

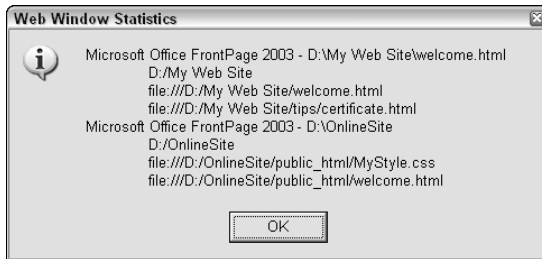


Figure BC1-6 shows that I have two windows open, both of which have an open Web site. Each of the windows also has two files open. The filenames include the `file:///` protocol indicator to show that they're a file rather than some other object.



It's also important to know that the `WebWindows` collection only deals with the local copy of a file when you work with remote sites. Unlike the `Webs` collection, which includes entries for both the local and remote `WebEx` objects, the `WebWindows` collection contains only the local `Web` object. Consequently, when you list the Web site shown in Figure BC1-5 (including two open files), you see the output shown in Figure BC1-7.

Figure BC1-7: Remote site connections include only the local file store.



Working with FrontPage Documents

Documents are at the center of the FrontPage operation, just as they are for any other application that you use. The purpose of using an application is to manipulate data in some way. Even utilities, such as a server monitor or a network security application, work with data. Consequently, getting the data manipulation capability in the form of document control from your applications is important.

FrontPage documents are easier to work with than many documents because they're pure text. You can see any changes quickly, and you don't have to worry about hidden elements. On the other hand, the FrontPage document environment is more complicated than many other environments because you have the concept of a Web site to consider. The Web site contains multiple folders, files, and resources, such as graphics and templates.



Although FrontPage provides a wealth of templates and art that you can use for your documents, you might want additional resources. You can find great art examples at Design Gallery Live (<http://dgl.microsoft.com>). The Template Gallery at <http://officeupdate.microsoft.com/templategallery/> provides a wealth of templates that you can use with FrontPage as well. Examining these templates can help you create custom templates that better suit your needs. In addition, many of these templates provide coding tricks that you can use.

Automating Web site creation



Preparing a new Web site for use is a time-consuming undertaking if you do it regularly. Fortunately, FrontPage makes it easy to automate this process so that you can perform the task literally in seconds without missing a single setting and with extreme consistency. Listing BC1-6 shows one technique for creating a Web site automatically and some of the settings that you might want to change. (You can find the source code for this example on the Dummies.com site at <http://www.dummies.com/go/vbafad5e>.)

Listing BC1-6 Creating a Web Site

```

Public Sub CreateWebSite()
    ' Contains the new Web site.
    Dim NewSite As WebEx

    ' Contains the default Web page.
    Dim WelcomePage As WebFile

    ' Create the new site.
    Set NewSite = Application.Webs.Add("C:\MyTempSite")

    ' Configure the new site.
    With NewSite

        ' Define the navigation key values.
        .Properties("vti_navbuttonprevlabel") = "Previous"
        .Properties("vti_navbuttonhomelabel") = "Go Home"
        .Properties("vti_navbuttonnextlabel") = "Next"
        .Properties("vti_navbuttonuplabel") = "Up a Level"

        ' Set the language and character set.
        .Properties("vti_defaultlanguage") = "en-us"
        .Properties("vti_defaultcharset") = "windows-1252"
        .Properties("vti_encoding") = "utf8-nl"

        ' Apply the changes.
        .Properties.ApplyChanges

        ' Refresh the site to match the new properties.
        .Refresh

        ' Define a theme for the site.
        .ApplyTheme ("Spring")

        ' Add basic folders to the new site.
        .RootFolder.Folders.Add "Graphics"
        .RootFolder.Folders.Add "Products"
        .RootFolder.Folders.Add "Contact Us"

        ' Add an initial Web page.
        Set WelcomePage = .RootFolder.Files.Add("Index.HTM")
        WelcomePage.Open
    End With
End Sub

```

The code begins by creating several objects. It uses the `Application.Webs.Add` method to create the `NewSite` object. This is a local object that you can publish later by using the `NewSite.Publish` method. For now, it's easier to work with the Web site locally so that data transfer times don't become a problem and so that you can maintain some level of security over the new site.

Setting up a new Web site means modifying properties. You can see these settings by right-clicking anywhere on the FrontPage Web Site tab and choosing Site Properties from the context menu. Modifying the properties in code is a little more difficult because you have to discover the names that Microsoft uses for the standard property entries. The example shows some standard properties that you can modify.



Microsoft doesn't do a particularly good job of telling you about the 32 properties that a Web site supports. The help file for the Properties property tells you about one property, and it isn't even a default property. You can use the Debugger to learn the names of any default property supported by any FrontPage object. (See the "Using the Locals Window" section of Chapter 6 for details on using the Debugger to view properties.) In most cases, the vti properties are FrontPage Server Extension meta keys. You can find a complete list of these meta keys at the SharePoint Products and Technologies site: <http://msdn.microsoft.com/office/server/moss/community/>. Look in the SharePoint Team Services SDK\RPC Protocol\FrontPage Server Extensions RPC Methods\Meta Keys folder. The vti values appear in alphabetical order, as shown in Figure BC1-8.

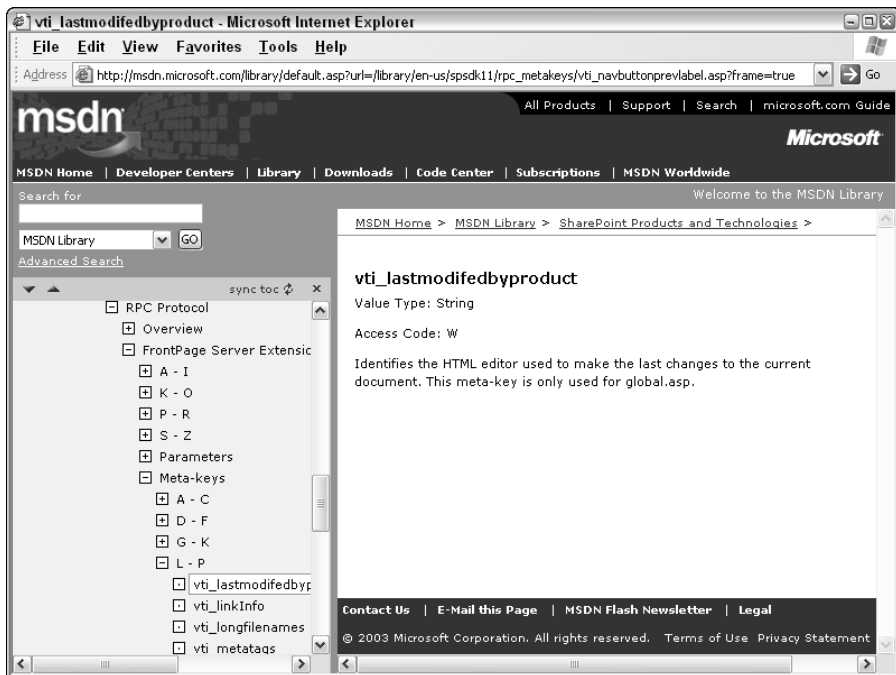


Figure BC1-8:
The hidden location of the vti values.

After the code changes the properties, it uses the `Properties.ApplyChanges` method to make the changes permanent. It then calls the `Refresh` method to synchronize the information between the program and the FrontPage application. You must perform both steps to ensure that you can see the changes that your program makes later.

Generally, you want the same theme used for an entire Web site so that all the pages look consistent. The `ApplyTheme` method lets you apply a theme to a Web site before you create any pages for it. This step ensures that your Web site has a consistent appearance from the very start.

When you know that you need to create certain folders for every Web site, it pays to make them part of the setup routine. The code uses the `RootFolder.Folders.Add` method to add the three standard folders to this Web site: `Graphics`, `Products`, and `Contact Us`.

Every Web site also needs an index or default page. The code adds this page by using the `RootFolder.Files.Add` method. Because you normally start working on this page immediately, it pays to have the code open it for you by using the `Open` method. As a further refinement, you can automate the process of creating this initial page by adding code to write some of the page automatically. Listing BC1-2, earlier in this chapter, shows some techniques you can use to perform this task. Figure BC1-9 shows the state of the Web site at this point.

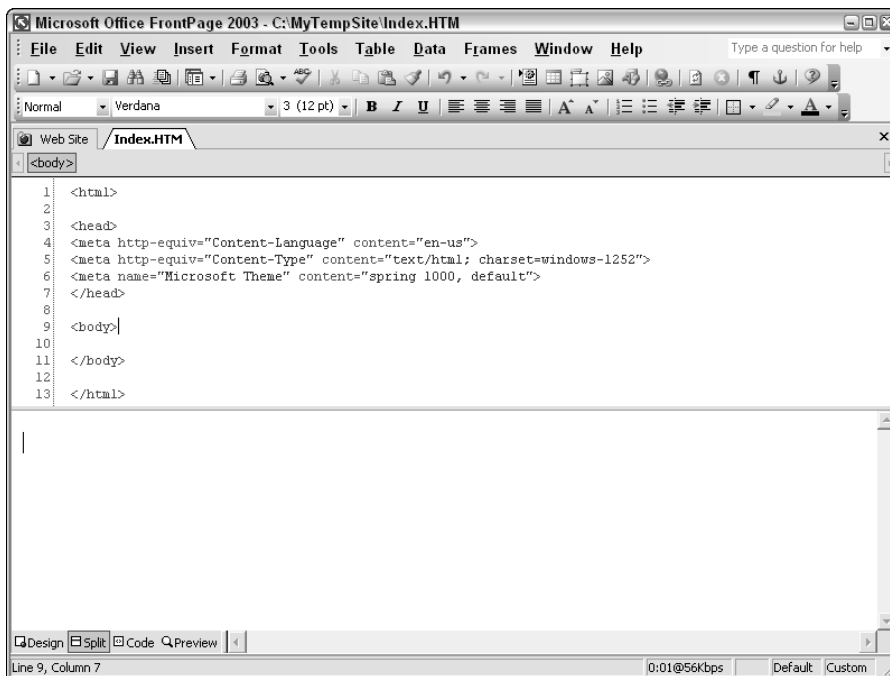


Figure BC1-9:
The results of automatic site creation.

Notice that the Web page includes the character set and the theme requested in Listing BC1-6. These two additions will appear on every Web page, along with any other defaults that you include. A seemingly small change when you create a Web site can result in a large time savings as you develop the Web site content.

Designing a basic template application

Templates can greatly reduce the work required to generate new Web pages because they take care of the common coding for you. All you need to worry about is the unique content of the page. Using templates also makes it easier to maintain a consistent page appearance because every page starts with the same content, layout, and functionality. A template consists of three essential elements:

- ✓ **HTML file:** This contains the template code.
- ✓ **Device Independent Bitmap (DIB) file:** This contains a picture of the template.
- ✓ **Information (INF) file:** This contains a description of the template.

The HTML file contains basic tags, any meta tags, and content that you want every Web page to use. You can create this file within FrontPage by using all the same FrontPage features that you've used in the past. Figure BC1-10 shows the HTML file used for this example.

Creating the DIB file comes next. Display the template file in a browser or by using the FrontPage Preview mode. Use a good screen-capture and graphics-manipulation program to get a copy of the screen. No, a copy of Microsoft Paint won't do the trick. I use Paint Shop Pro (<http://www.jasc.com/>) because you can download a copy free, and the price for the shareware is quite reasonable. You must resize the image to 110 x 124 pixels high. It's fine if you can't quite read the image because you only want to provide the user with an idea of what the page looks like. Save the captured and resized image as a DIB file.



You can use any existing FrontPage template INF file as the basis for your INF file. All the files have the same format. Listing BC1-7 shows the code that you need for a template INF file. (You can find the source code for this example on the Dummies.com site at <http://www.dummies.com/go/vbafd5e>.)

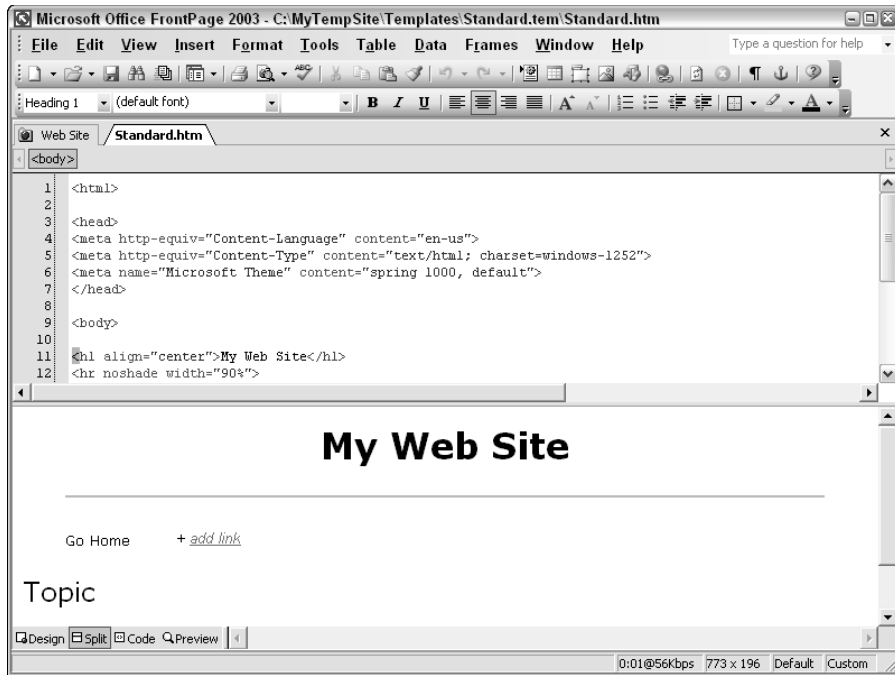


Figure BC1-10:
A template file.

Listing BC1-7 Creating a Template INF

```
[info]
_LCID=1033
_version=1.0.0.0
title=Standard Page
description=Create a Standard web page.
```

The entries define the locale identifier (`_LCID`; essentially, your location and language), version, title, and description of the template. The `_LCID` is always 1033 for United States English. Review the chart at http://krafft.com/scripts/deluxe-calendar/lcid_chart.htm to find the `_LCID` value for your location.



You can permanently add new templates to your FrontPage environment by placing the template in the `\Program Files\Microsoft Office\Templates\1033\PAGES` folder. Make sure that the template includes all three files, or else it won't be displayed properly in the Web Site Templates dialog box.

After you have a template to use, it's time to make it work for you. Listing BC1-8 shows how to automate template usage in FrontPage by using VBA.

Listing BC1-8 Adding Template Functionality

```
Public Sub MakeStandardPage()  
    ' Contains the target Web site.  
    Dim TheSite As WebEx  
  
    ' Get the site.  
    If (Application.Webs.Count > 0) Then  
        If (Application.Webs(0).Title = "C:/MyTempSite")  
            Then  
                Set TheSite = Application.Webs(0)  
            Else  
                Set TheSite =  
                    Application.Webs.Add("C:\MyTempSite")  
            End If  
        Else  
            Set TheSite = Application.Webs.Open("C:\MyTempSite")  
        End If  
  
        ' Set the Web site to use a template.  
        TheSite.ApplyTemplate TheSite.Title + _  
            "\Templates\Standard.tem", True  
  
        ' Contains the target folder.  
        Dim Target As WebFolder  
  
        ' Get the target folder.  
        Set Target = TheSite.RootFolder.Folders("Products")  
  
        ' Contains the Web page template.  
        Dim StdPage As WebFile  
  
        ' Create the new page.  
        Set StdPage = TheSite.RootFolder.Files("Standard.HTM")  
        StdPage.Copy Target.Url + "\NewFile.HTM", True, True  
  
        ' Contains the new Web page.  
        Dim NewPage As WebFile  
  
        ' Open the page for editing.  
        Set NewPage = Target.Files("NewFile.HTM")  
        NewPage.Open  
    End Sub
```

The code begins by determining the status of the Web site. If the Web site isn't open, the code opens it. Otherwise, the code uses the existing copy. This code is a little crude, but it gets the job done. Your program will require similar code to ensure that it detects the FrontPage status and reacts accordingly.

Next, the code applies the new template to the Web site by using the `ApplyTemplate` method. This method enables you to replace templates as needed to ensure that you can switch between document types. For example, you might use one template for product information and another for contacts.

You normally need to provide a location for the new file. The code assumes a standard location, but selecting a location is something that you could do in code by using a form. The idea is to make the program flexible so that you can use it for more than just one file.

At this point, the code has to make a copy of the template file and place it in the target location. The `TheSite.RootFolder.Files("Standard.HTM")` property contains the location of the template. The code uses the `StdPage.Copy` method to create a copy in the target location. Notice that the code renames the file to `NewFile.HTM` as part of the copy process.

Finally, the code assigns the new file to a `WebFile` object and uses the `Open` method to open it for editing. You can add code here to perform automated customization. For example, you can ask the user a series of questions that helps to format the new file and add some content to it.

