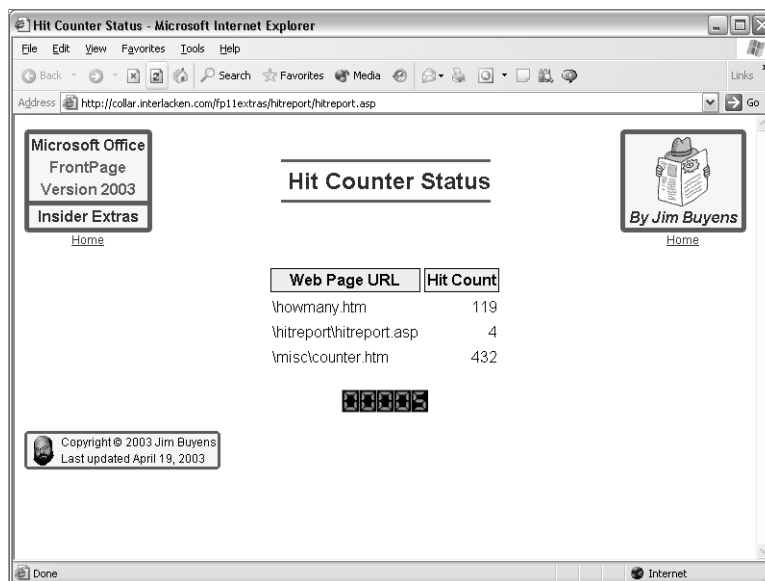


## Insider Extra 12

# Accessing Hit Counts with Scripts

Hit counters provide a good snapshot of usage for a single page, but the more hit counters your site contains, the more tedious checking them will be. The Web page shown in Figure IE12-1 provides a solution by displaying, on one page, all the current hit counts in a Web site.



**Figure IE12-1.** This Web page displays a list of all hit counts in a Web site.

Unfortunately, there's no web component or wizard that can produce a page like the one shown in Figure IE12-1. Producing such a page requires that you be at least slightly proficient in ASP or Microsoft ASP.NET programming. In addition, you must have a server-based Web site that resides on a computer running Microsoft Windows 2000, Windows XP Professional, or Windows Server 2003. Finally, the folder where the Web page resides must be marked as executable.

## Microsoft Office FrontPage 2003 Inside Out

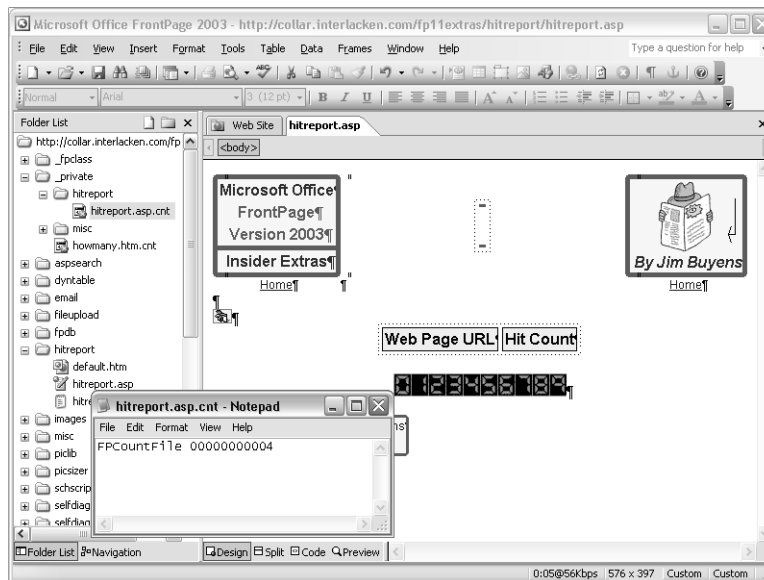
**Note** Did you notice the quirk in the hit counts shown in Figure IE12-1? The table in the center of the page shows a hit count of 4 for the hitreport.asp page, and the actual hit counter shows a count of 5. This is because the code that generates the report runs *before* the HTML for the Web page leaves the Web server. The code that increments the hit counter and displays the new value, on the other hand, runs *after* the browser receives the HTML and requests the hit counter picture.

The first challenge in reporting hit count data is finding out where the data resides. A bit of nosing about should reveal that the hit count for each file resides in a file named `_private/path/pagename.cnt`. The Folder List shown in Figure IE12-2, for example, shows the three hit count files listed in Table IE12-1.

**Tip** If the `_private` folder isn't visible in your Web site, choose Site Settings from the Tools menu, click the Advanced tab, select Show Hidden Files And Folders, and click OK.

**Table IE12-1. Location of Some Hit Count Files**

Web page URL (relative to root of current Web site)	Hit count file location
hitreport/hitreport.asp	_private/hitreport/hitreport.asp.cnt
misc/counter.htm	_private/misc/counter.htm.cnt
howmany.htm	_private/howmany.htm.cnt



## Accessing Hit Counts with Scripts

**Figure IE12-2.** Each hit counter in a Web site stores its count in a file that resides within the site's `_private` folder. The X icons indicate that publishing a Web site doesn't copy hit counts to the destination site.

Figure IE12-2 also shows the content of a typical hit count file: one record containing the identifier *FPCountFile*, followed by a space and the current count value.

Here are the requirements for displaying the data shown in Figure IE12-1:

- 1 Walk the folder tree that starts at the `_private` folder of the current Web site.
- 2 Within each folder, locate all files that have a file extension of `.cnt`.
- 3 Open each such file, read the first record, and select the string of characters that follows the first space. Report this as the count.
- 4 Report the Web page location as the location of the hit count file, minus the folder tree prefix `_private` and the file extension `.cnt`.

The next two sections describe two Web pages that implement these requirements. The first page uses ASP, and the second page uses ASP.NET.

## Reporting Hit Counts with ASP

This section explains how to write an ASP page that reports the current value of all Hit Counter components in a Web site. To view the complete solution as you read the explanation, use Microsoft Office FrontPage 2003 to open the file `hitreport/hitreport.asp` in the Insider Extra Web site you installed from the companion CD, and then switch to Code view.

The script begins by declaring the variables it's going to use and then initializing one object and one variable, as follows:

```
Set FSO = Server.CreateObject("Scripting.FileSystemObject")
begDir = Server.MapPath(GetAppPath() & "_private/")
```

The first of these statements creates a *Scripting.FileSystemObject* object that other parts of the script use for performing file operations. The second statement obtains and stores the physical folder path that's equivalent to the `_private` folder in the current application's root folder. This is the correct starting point to use when scanning for hit count files—assuming, of course, that the Web page doing the scanning resides in a first-level subdirectory of the Web site's root folder. (Remember that a double dot (`..`) means “Go back one folder.”) If your hit counter analysis page resides in a different location, you'll need to adjust this relative URL.

The following statement appears later in the Web page and kicks off the entire reporting process:

```
CheckDir (begdir)
```

## Microsoft Office FrontPage 2003 Inside Out

*CheckDir* is a subroutine that searches for folders and .cnt files within a directory. Here's how this subroutine works:

- For each file it finds, *CheckDir* checks for a file extension of .cnt. Then, for each file that satisfies this condition, it opens the file, gets the count, and writes one row of an HTML table.
- For each subfolder it finds within the given path, *CheckDir* calls the *CheckDir* subroutine again, passing it the current path (including the subfolder).

Did you notice that the *CheckDir* subroutine calls itself? This is perfectly acceptable in all forms of Microsoft Visual Basic. Each time you call a subroutine, Visual Basic loads a new copy into memory, and it has no problem keeping several copies of the same subroutine in memory at the same time.

**Note** Problems can result from having hundreds of copies of a Visual Basic subroutine in memory at one time, but this is seldom a problem when processing subfolders. There's only one copy of the subroutine in memory for each folder level, and file system restrictions limit the number of subfolder levels.

To code the *CheckDir* subroutine, proceed as follows:

- 1 Declare the subroutine, assigning the name *CheckDir* and expecting one argument, named *aDir*. (This argument will specify the folder to process). In short, code these statements:

```
Sub CheckDir(aDir)
End Sub
```

- 2 Between the two statements you just coded, declare the six variables shown in the following code. This guarantees that each copy of the subroutine has its own copy of these variables.

```
Dim curDir
Dim File
Dim relPath
Dim folder
Dim url
Dim cnt
```

- 3 Next create a *Folder* object that represents the folder named in the *aDir* argument:

```
Set curDir = FS0.GetFolder(aDir)
```

- 4 Set up a loop that sequentially selects each file in the *Files* collection of the *curDir* folder object. To do this, add the following code next in sequence:

```
For Each File In curDir.Files
Next
```

## Accessing Hit Counts with Scripts

- 5 Within this loop, determine whether the last four characters of the file name are `.cnt`:

```
If LCase(Right(File.Path, 4)) = ".cnt" Then
End if
```

- 6 Within this condition, add code to:

- 1 Discard the leading portion of the file's physical path name (that is, the part ending in `/_private/`).
- 2 Discard the last four characters of the file name ("`.cnt`").
- 3 Call a function named `GetCount` that extracts the hit counter value from a given file.
- 4 Write the results as one row of an HTML table, as shown here:

```
url = Mid(File.Path, Len(BegDir) + 1)
url = Left(relPath, Len(relPath) - 4)
cnt = GetCount(File.Path)
response.write "<tr>" & _
    "<td>" & url & "</td>" & _
    "<td align=right>" & cnt & "</td>" & _
"</tr>" & vbCrLf
```

The first statement in this code warrants some additional explanation:

- A typical value for `File.Path` is `C:\InetPub\wwwroot\fp1\extras\_private\hitreport\hitreport.asp.cnt`.
- A typical value for `BegDir` is `C:\InetPub\wwwroot\fp1\extras\_private`. The length of this path is 38 characters.
- The expression `Mid(File.Path, Len(BegDir) + 1)` selects a substring of the `File.Path` string starting at position 39 (`Len(BegDir) + 1`) and ending at the end of the string.
- The result is `hitreport\hitreport.asp.cnt`.

This step completes processing of each hit count file in the current folder.

- 7 To search for hit count files that reside in subfolders, set up a loop that sequentially selects each folder in the `Subfolders` collection of the `curDir` folder object. This code goes after the `Next` statement from step 4, and before the `End Sub` statement from step 1.

```
For Each folder In curDir.Subfolders
Next
```

- 8 Within this loop, discard the leading portion of the current subfolder's physical path name (the part ending in `/_private/`), and store the result in a variable named `relPath`:

```
relPath = LCase(Mid(folder.Path, Len(BegDir) + 1))
```

## Microsoft Office FrontPage 2003 Inside Out

- 9 Make sure that the subfolder name doesn't contain the string "`\vti`". If it does, the subfolder is a FrontPage system folder and doesn't actually contain hit count files. The following statement accomplishes this task; code it next in sequence:

```
If InStr(relPath, "\vti") < 1 Then
End If
```

- 10 If the subfolder isn't a FrontPage system folder, call the *CheckDir* subroutine with the subfolder as an argument:

```
CheckDir(folder.Path)
```

Notice that the loop from step 7 runs the *CheckDir* subroutine once for each nonsystem subfolder in each folder.

This completes the coding for the *CheckDir* subroutine. Here's a complete listing of the code:

```
Sub CheckDir(aDir)
    Dim curDir
    Dim File
    Dim relPath
    Dim folder
    Dim url
    Dim cnt

    Set curDir = FSO.GetFolder(aDir)

    For Each File In curDir.Files
        If LCase(Right(File.Path, 4)) = ".cnt" Then
            url = Mid(File.Path, Len(BegDir) + 1)
            url = Left(url, Len(url) - 4)
            cnt = GetCount(File.Path)
            response.write "<tr>" & _
                "<td>" & url & "</td>" & _
                "<td align=right>" & cnt & "</td>" & _
                "</tr>" & vbCrLf
        End If
    Next

    For Each folder In curDir.Subfolders
        relPath = LCase(Mid(folder.Path, Len(BegDir) + 1))
        If InStr(relPath, "\vti") < 1 Then
            CheckDir (folder.Path)
        End If
    Next

    Set curDir = Nothing
End Sub
```

## Accessing Hit Counts with Scripts

The only remaining piece of code is the *GetCount* function that extracts the hit counter value from a given hit count file name. Here's how to code this function:

- 1 Declare the function, assigning the name *GetCount* and expecting one argument named *aFile*. This argument will specify the file to process. Here's the code:

```
Function GetCount(aFile)
End Function
```

- 2 Between these statements, declare the five variables shown here:

```
Dim ctrFile
Dim ctrRcd
Dim ctrFld
Dim hitCount
Dim fld
```

- 3 Open the file that the *aFile* argument specifies, and create an object named *ctrFile* to manipulate the open file:

```
Set ctrFile = FS0.OpenTextFile(aFile)
```

- 4 Read one line from the open file, and store the content in a variable named *ctrRcd*:

```
ctrRcd = ctrFile.ReadLine
```

- 5 Split the *ctrRcd* value into an array named *ctrFld*, using a space character as a delimiter:

```
ctrFld = Split(ctrRcd, " ")
```

- 6 Set the hit counter value (in a variable named *hitCount*) to a default of zero:

```
hitCount = 0
```

- 7 Set up a loop that inspects each element in the *ctrFld* array. As soon as you find a numeric element, set the *hitCount* variable to that value, and exit the loop:

```
For Each fld In ctrFld
  If IsNumeric(fld) Then
    hitCount = CLng(fld)
    Exit For
  End If
Next
```

Note that if the file is corrupted and the first record doesn't contain any numeric fields, the default value of zero will remain after the loop ends.

- 8 Close the open file, destroy the open *File* object, and set the function's return value to the hit count stored in *hitCount*:

```
ctrFile.Close
Set ctrFile = Nothing
GetCount = hitCount
```

## Microsoft Office FrontPage 2003 Inside Out

This completes the coding for the *GetCount* function. A listing of the complete function appears here:

```
Function GetCount(aFile)
    Dim ctrFile
    Dim ctrRcd
    Dim ctrFld
    Dim hitCount
    Dim fld

    Set ctrFile = FS0.OpenTextFile(aFile)
    ctrRcd = ctrFile.readline
    ctrFld = Split(ctrRcd, " ")
    hitCount = 0

    For Each fld In ctrFld
        If IsNumeric(fld) Then
            hitCount = CLng(fld)
            Exit For
        End If
    Next

    ctrFile.Close
    Set ctrFile = Nothing
    GetCount = hitCount
End Function
```

Even if you don't understand everything about this page, you can add it to your own Web site and display your own counts rather easily. Just copy it into an executable subfolder in your Web site, delete the Include Page components at the top and bottom of the page, and browse it using an *http://* URL. After you get the page working, format it to suit.

As presented, this code doesn't account for certain error conditions, such as the *OpenTextFile* method failing because a hit count file is in use. A hit count file containing no records is another error not handled. However, these conditions are rare.

## Reporting Hit Counts with ASP.NET

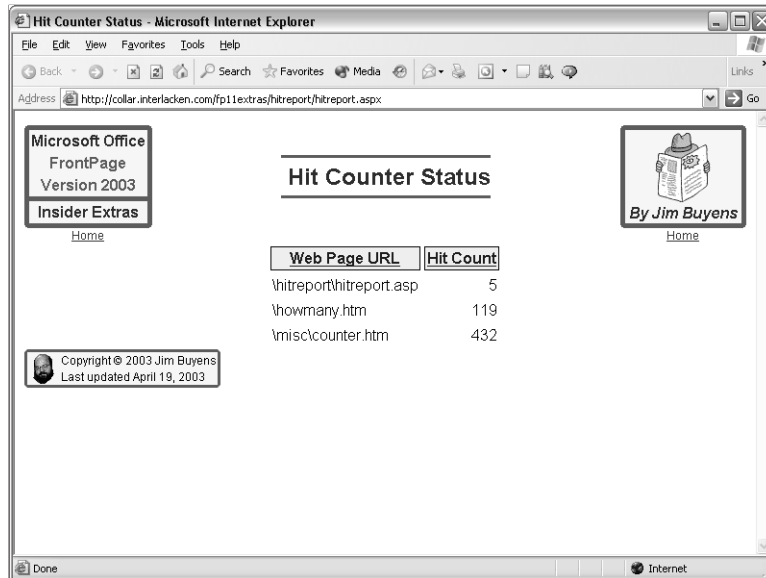
This section describes another Web page that reports the current value of all FrontPage Hit Counter components in a site. This page, however, uses up-to-date ASP.NET technology. Figure IE12-3 shows some typical results.



### On the Web

If your Web server runs on Windows 2000 or Windows XP Professional, you'll need to install the Microsoft .NET Framework after you install the Web server, and before you run this page. To obtain the .NET Framework, follow the links at [www.microsoft.com/net/](http://www.microsoft.com/net/).

## Accessing Hit Counts with Scripts



**Figure IE12-3.** This Web page displays a list of all hit counts in a Web site.

In addition to using newer technology, this page can sort its results on either Web Page URL or Hit Count. To sort on either column, click the hyperlink in the column heading.

Here's how to code this Web page. To view the complete solution as you read the explanation, use FrontPage to open the `hitreport.aspx` file that the Sample Files setup program installs at `[My Documents]\Microsoft Press\FrontPage 2003 Inside Out\hitreport`, and then switch to Code view.

- 1 Open a new, blank Web page, switch to Code view, and add a form server control to the `<body>` section. Here's the required code:

```
<form id="Form1" method="post" runat="server">
</form>
```

- 2 Add an ASP.NET DataGrid control to the form you just created. Name the DataGrid control `grdCounts`, and configure it not to auto-generate columns. Do configure it to allow sorting, and specify `grdCounts_SortCommand` as the subroutine that ASP.NET should run if the Web visitor clicks a sortable column heading. Here's the required code, which includes some additional formatting instructions:

```
<asp:DataGrid id="grdCounts" runat="server"
    AutoGenerateColumns="False" AllowSorting="True"
    onSortCommand="grdCounts_SortCommand"
    HorizontalAlign="Center"
    BorderWidth="0" CellSpacing="4" CellPadding="2">
</asp:DataGrid>
```

## Microsoft Office FrontPage 2003 Inside Out

- 3 Within the `<asp:DataGrid>` tags, define two columns: one to report a data field named *Page* and another to report a data field name *Hits*. Specify these field names as sort expressions as well. Here's the required code, again with some formatting attributes:

```
<columns>
  <asp:BoundColumn DataField="Page" SortExpression="Page"
                    HeaderText="Web Page URL">
    <headerstyle CssClass="thd"></headerstyle>
    <itemstyle VerticalAlign="Top"></itemstyle>
  </asp:BoundColumn>

  <asp:BoundColumn DataField="Hits" SortExpression="Hits"
                    HeaderText="Hit Count">
    <headerstyle CssClass="thd"></headerstyle>
    <itemstyle HorizontalAlign="Right" VerticalAlign="Top"></itemstyle>
  </asp:BoundColumn>
</columns>
```

- 4 At the top of the page, add an `@Page` directive, `@Import` directives for the *System.IO* and *System.Data* namespaces, and a server-side code declaration block. These statements appear even before the `<html>` tag:

```
<%@Page Language="vb" AutoEventWireup="true"%>
<%@Import Namespace="System.IO"%>
<%@Import Namespace="System.Data"%>
<script runat="server">
</script>
```

The Web page will use objects methods from the *System.IO* namespace to read folders and files within your Web site. The page will use objects and methods from the *System.Data* namespace to store the data in a format that the *DataGrid* control can use.

- 5 Declare the following variables within the code declaration block you created in the previous step (that is, within the `<script runat="server">` and `</script>` tags):

```
Dim strBegDir As String
Dim dtbHitCounts As New DataTable()
```

The *strBegDir* variable will store the physical location of your Web site's `_private` folder. The *dtbHitCounts* variable points to a *DataTable* object that will store the hit counts for reporting.

## Accessing Hit Counts with Scripts

- 6** Next, code a *Page\_Load* subroutine that accepts the usual ASP.NET arguments. Within this subroutine, check to see if the page is running because of a postback. If not, call a subroutine named *ScanForHitCounts*, passing an argument of "Page". The subroutine will respond to this argument by producing results in Page sequence.

```
Private Sub Page_Load(ByVal sender As Object, _
                    ByVal e As System.EventArgs)
    If Not Page.IsPostBack Then
        ScanForHitCounts("Page")
    End If
End Sub
```

**Note** In ASP.NET, a *postback* is the condition whereby a Web form submits input to the same page that displayed it. An initial page display isn't a postback, but clicking a sortable column heading is a postback.

- 7** Code a subroutine named *grdCounts\_SortCommand* that ASP.NET will run whenever the visitor clicks a sortable column heading. (Notice that this is the subroutine name that the DataGrid control's *onSortCommand* attribute specified.) Arrange for this subroutine to call the *ScanForHitCounts* subroutine just as the *Page\_Load* subroutine did, but pass the value in *e.SortExpression* as an argument.

```
Private Sub grdCounts_SortCommand( _
    ByVal source As Object, _
    ByVal e As DataGridSortCommandEventArgs)
    ScanForHitCounts(e.SortExpression)
End Sub
```

Notice that the second argument to this subroutine is a *DataGridSortCommandEventArgs* object. ASP.NET requires that any subroutine handling *onSortCommand* events receive an argument of this type. The object's *SortExpression* property contains the value you specified in the *SortExpression* attribute of the column heading that the Web visitor clicked.

The next procedure explains how to code the *ScanForHitCounts* subroutine that both the *Page\_Load* and the *grdCounts\_SortCommand* subroutines invoke:

- 1** Declare a function named *ScanForHitCounts* that receives one *String* argument named *astrSortFld*. (This argument will specify the sort field.) Here's the code:

```
Sub ScanForHitCounts(ByVal astrSortFld As String)
End Sub
```

- 2** Within this subroutine, declare a variable named *dvwHitCounts* that will point to a *DataView* object. This object will present a sorted view of all the hit counts. Here's the code:

```
Dim dvwHitCounts As DataView
```

## Microsoft Office FrontPage 2003 Inside Out

- 3** Get the starting path to the current ASP.NET application, convert this to a physical folder location, append the folder name `_private`, and store the results in the `strBegDir` variable you declared in the previous procedure:

```
strBegDir = Path.Combine( _
    Server.MapPath(Request.ApplicationPath), _
    "_private")
```

- 4** Next add two columns to the `DataTable` object you created in the previous procedure; a string column named `Page` and an 32-bit integer column named `Hits`. Here's the code:

```
dtbHitCounts.Columns.Add( _
    New DataColumn("Page", Type.GetType("System.String")))
dtbHitCounts.Columns.Add( _
    New DataColumn("Hits", Type.GetType("System.Int32")))
```

- 5** To start filling the `DataTable` with data, call a subroutine named `CheckDir`, passing the `strBegDir` variable as an argument. This subroutine will locate all hit counter values in the folder tree, beginning in the location that the `strBegDir` variable specifies, and will store the results in the `DataTable` object `dtbHitCounts`.

```
CheckDir(strBegDir)
```

- 6** When the `CheckDir` subroutine ends, the `DataTable` will contain all the hit counts in your site. To start creating a sorted view of this data, create a new `DataView` object based on the `dtbHitCounts` table, and store its location in the `dvwHitCounts` variable. This requires the following code after the line from step 5:

```
dvwHitCounts = New DataView(dtbHitCounts)
```

- 7** If the `astrSortFld` argument (converted to lowercase) equals `hits`, tell the `DataView` object to sort the data it delivers in descending order by the `Hits` field, and then by the `Page` field. Otherwise, tell the `DataView` object to sort the data on the `Page` field. This requires the following statements after the code from step 7.

```
If LCase(astrSortFld) = "hits" Then
    dvwHitCounts.Sort = "Hits DESC, Page"
Else
    dvwHitCounts.Sort = "Page"
End If
```

- 8** To display the data, assign the `DataView` object `dvwHitCounts` as the `DataSource` of the `DataGrid` object `grdCounts`, and then load the data into the grid:

```
grdCounts.DataSource = dvwHitCounts
grdCounts.DataBind()
```

## Accessing Hit Counts with Scripts

This completes the coding for the *ScanForHitCounts* subroutine. A listing of the complete subroutine is shown here:

```
Sub ScanForHitCounts(ByVal astrSortFld As String)
    Dim dvwHitCounts As DataView
    strBegDir = Path.Combine( _
        Server.MapPath(Request.ApplicationPath), _
        "_private")
    dtbHitCounts.Columns.Add( _
        New DataColumn("Page", Type.GetType("System.String")))
    dtbHitCounts.Columns.Add( _
        New DataColumn("Hits", Type.GetType("System.Int32")))
    CheckDir(strBegDir)
    dvwHitCounts = New DataView(dtbHitCounts)
    If LCase(astrSortFld) = "hits" Then
        dvwHitCounts.Sort = "Hits DESC, Page"
    Else
        dvwHitCounts.Sort = "Page"
    End If
    grdCounts.DataSource = dvwHitCounts
    grdCounts.DataBind()
End Sub
```

As you no doubt recall, the *ScanForHitCounts* subroutine calls a *CheckDir* subroutine that you haven't written yet. To rectify this omission, proceed as follows:

- 1 At the bottom of the code declaration block (that is, just before the `</script>` tag), declare a subroutine named *CheckDir* that receives one *String* argument, named *astrDir*. This argument will specify the name of the folder to check for hit counters. Here's the code:

```
Sub CheckDir(ByVal astrDir As String)
End Sub
```

- 2 Within this subroutine, declare the following variables:

```
Dim dinFolder As DirectoryInfo
Dim dinSubDir As DirectoryInfo
Dim finFile As FileInfo
Dim drwCounts As DataRow
Dim strRelPath As String
Dim strUrl As String
```

A *DirectoryInfo* object has properties and methods that describe and manipulate a specific folder. A *FileInfo* object provides similar capabilities for a specific file. A *DataRow* object contains one row of a *DataTable* object.

- 3 Create a new *DirectoryInfo* object that describes the folder that the *astrDir* argument specifies, and save its address in the *dinFolder* variable:

```
dinFolder = New DirectoryInfo(astrDir)
```

- 4 Write a loop that provides access to each file in the folder that the *dinFolder* object describes:

## Microsoft Office FrontPage 2003 Inside Out

```
For Each finFile In dinFolder.GetFiles
Next
```

Notice that the *FileInfo* object named *finFile* will describe a different file during each iteration of the loop.

- 5 Within the loop you coded in step 4, check each file extension to determine whether it equals `.cnt`:

```
If LCase(finFile.Extension) = ".cnt" Then
End If
```

- 6 If the file meets this condition, retrieve its fully qualified file name, remove the path to your Web site's `_private` folder, remove the `.cnt` file extension, and store the results in the *strUrl* variable:

```
strUrl = Mid(finFile.FullName, Len(strBegDir) + 1)
strUrl = Left(strUrl, Len(strUrl) - 4)
```

- 7 Still within the *If* statement from step 5 (and after the code from step 6) create a new *DataRow* object with the same columns as the *DataTable* object *dtbHitCounts*, and store its address in the *drwCounts* variable:

```
drwCounts = dtbHitCounts.NewRow
```

- 8 Continuing within the *If* from step 5, copy the *strUrl* value from step 6 into the *Page* column of the new *DataRow* object:

```
drwCounts("Page") = strUrl
```

- 9 Likewise, store the corresponding hit count value in the *Hits* column. To get the hit count, call a function named *GetCount*, passing the hit count file's fully qualified name as an argument:

```
drwCounts("Hits") = GetCount(finFile.FullName)
```

- 10 Next, add the new *DataRow* object to the *DataTable* object named *dtbHitCounts*:

```
dtbHitCounts.Rows.Add(drwCounts)
```

- 11 At the end of the subroutine (just before the *End Sub* statement), write a loop that inspects each subfolder that the *dinFolder* object contains:

```
For Each dinSubDir In dinFolder.GetDirectories
Next
```

## Accessing Hit Counts with Scripts

- 12** Within this loop (and therefore for each subfolder), remove the leading path to your Web site's `_private` folder. Then determine whether the remaining path contains the characters `\_vti` (indicating a FrontPage system folder). If not, call the *CheckDir* subroutine again, passing the full name of the subdirectory as an argument.

```
strRelPath = LCase(Mid(dinSubDir.FullName, Len(strBegDir) + 1))
If InStr(strRelPath, "\_vti") < 1 Then
    CheckDir(dinSubDir.FullName)
End If
```

This completes the coding for the *CheckDir* subroutine. A complete listing is shown here. Notice that as in the ASP version of this page, the *CheckDir* subroutine calls itself repeatedly, once for each subfolder it detects. This is how the subroutine “walks” the folder tree.

```
Sub CheckDir(ByVal astrDir As String)
    Dim dinFolder As DirectoryInfo
    Dim dinSubDir As DirectoryInfo
    Dim finFile As FileInfo
    Dim drwCounts As DataRow
    Dim strRelPath As String
    Dim strUrl As String

    dinFolder = New DirectoryInfo(astrDir)

    For Each finFile In dinFolder.GetFiles
        If LCase(finFile.Extension) = ".cnt" Then
            strUrl = Mid(finFile.FullName, Len(strBegDir) + 1)
            strUrl = Left(strUrl, Len(strUrl) - 4)
            drwCounts = dtbHitCounts.NewRow
            drwCounts("Page") = strUrl
            drwCounts("Hits") = GetCount(finFile.FullName)
            dtbHitCounts.Rows.Add(drwCounts)
        End If
    Next

    For Each dinSubDir In dinFolder.GetDirectories
        strRelPath = LCase(Mid(dinSubDir.FullName, Len(strBegDir) + 1))
        If InStr(strRelPath, "\_vti") < 1 Then
            CheckDir(dinSubDir.FullName)
        End If
    Next
End Sub
```

All that remains now is the task of coding the *GetCount* function. To address this pesky detail, proceed as follows:

- 1** At the bottom of the code declaration block (that is, just before the `</script>` tag), declare a function named *GetCount* that receives one *String* argument named *aFile*. This argument will specify the name of the hit count file. Here's the code:

```
Function GetCount(ByVal aFile As String) As Integer
End Function
```

## Microsoft Office FrontPage 2003 Inside Out

- 2 Within this function, declare the following variables:

```
Dim srdCount As StreamReader
Dim strCntRec As String
Dim strCntFlds() As String
Dim intCount As Integer
Dim strCount As String
```

A *StreamReader* is an object that can read data from an arbitrary sequence of bytes (such as, in this case, a file).

- 3 Open the file that the *astrFile* argument specifies, saving the resulting *StreamReader* address in the *srdCount* variable:

```
srdCount = File.OpenText(astrFile)
```

- 4 Initialize the *intCount* variable to zero. Later code will probably replace this value with an actual hit count.

```
intCount = 0
```

- 5 Determine whether the *StreamReader* object *srdCount* has a record available for reading. If so, read one line, and save the results in the *strCntRec* variable:

```
If srdCount.Peek <> -1 Then
    strCntRec = srdCount.ReadLine()
End If
```

- 6 Continuing within the *If* statement from step 5, split the *strCntRec* value into an array named *strCntFlds*, using a space character as a delimiter:

```
strCntFlds = Split(strCntRec, " ")
```

- 7 To finish the code within the *If*, set up a loop that inspects each element in the *strCntFlds* array. As soon as you find a numeric element, set the *intCount* variable to that value, and exit the loop:

```
For Each strCount In strCntFlds
    If IsNumeric(strCount) Then
        intCount = CLng(strCount)
        Exit For
    End If
Next
```

- 8 Add this statement immediately following the entire *If* structure. It closes the *StreamReader* object *srdCount*:

```
srdCount.Close()
```

- 9 Return the value in the *intCount* variable as the value of the function. This will be the last statement in the function, just before the *End Function* statement.

```
Return intCount
```

## Accessing Hit Counts with Scripts

This completes the coding for the *GetCount* function and, except for cosmetic formatting, the entire Web page. Here's a listing of the complete *GetCount* function:

```
Function GetCount(ByVal astrFile As String) As Integer
    Dim srdCntRec As StreamReader
    Dim strCntRec As String
    Dim strCntFlds() As String
    Dim intCount As Integer
    Dim strCount As String
    srdCntRec = File.OpenText(astrFile)
    intCount = 0
    If srdCntRec.Peek <> -1 Then
        strCntRec = srdCntRec.ReadLine()
        strCntFlds = Split(strCntRec, " ")
        For Each strCount In strCntFlds
            If IsNumeric(strCount) Then
                intCount = CLng(strCount)
                Exit For
            End If
        Next
    End If
    srdCntRec.Close()
    Return intCount
End Function
```

As you might have noticed, the ASP.NET version of this Insider Extra is both longer and more complex than the ASP version. This reflects the fact that ASP.NET is a much larger, more capable, and more rigorous programming environment than legacy ASP.



