FrontPage Web Object Model



WebWindows

WebWindow

PageWindows

PageWindow

PageWindow

WebFile

WebFile

Lists

NavigationNode

NavigationNodes

Properties

WebFiles

WebFiles

WebFiles

WebFiles

WebFiles

WebFiles

WebFiles

Legend

Object and collection Object only

Other notable members of the FrontPage object model.

asiclist	DocumentLibrary	
<u>ListFields</u>	ListFields	Pag
- <u>Web</u>	- <u>Web</u>	
Lists		
-NavigationNode	- <u>NavigationNode</u>	
<u>WebFile</u>	WebFile	
NavigationNodes	NavigationNodes	
Properties	Properties	
Themes	Themes	
<u>L</u> <u>Theme</u>	Theme	
- <u>WebFiles</u>	WebFiles	
WebFile	WebFile	
- <u>WebFolders</u>	WebFolders	
	WebWindows	
	WebWindow	
<u>WebFolder</u>	L <u>WebFolder</u>	
List	List	
Properties	Properties	
- <u>WebFiles</u>	WebFiles	
WebFile	<u>WebFile</u>	
L WebFolders	WebFolders	

New Web Object Model Objects

Visit the Office Developer Center on the Microsoft Developer Network Web site for the latest information about programming with Microsoft Office FrontPage 2003, including product news, technical articles, downloads, and samples.

The following table lists objects added to the Office FrontPage 2003 object model.

Object	Description
Discussion	Contains information about a Discussion list in a Microsoft FrontPage Web site.
<u>ListFieldAttachments</u>	Contains information about the attachments field when attachments are enabled for a list in a FrontPage Web site.
ListFieldRatingScale	Contains information about the rating scale list field for a survey list in a FrontPage Web site.
<u>WebPackage</u>	Represents a Web package that has been created in Microsoft Visual Basic for Applications.

New Web Object Model Properties (Alphabetical List)

Visit the Office Developer Center on the Microsoft Developer Network Web site for the latest information about programming with Microsoft Office FrontPage 2003, including product news, technical articles, downloads, and samples.

The following table lists properties added to the Office FrontPage 2003 object model (sorted alphabetically).

New Property	Object(s)
AllowAttachments	BasicList, DocumentLibrary, List, Survey, Discussion
AllowFillInChoices	ListFieldChoice
<u>AllowModerate</u>	BasicList, DocumentLibrary, List, Survey, Discussion
AllowRichHTML	ListFieldMultiLine
Author	WebPackage
<u>Company</u>	WebPackage
ComputationFormula	ListFieldComputed
DynamicTemplate	WebFile
<u>EndNumber</u>	ListFieldRatingScale
<u>IsHidden</u>	BasicList, DocumentLibrary, List, Survey, Discussion
<u>IsModified</u>	BasicList, DocumentLibrary, List, Survey, Discussion
OptimizeHTMLFlags	Application
OptimizeHTMLPublishFlags	Web, WebEx
<u>ReturnType</u>	ListFieldComputed
<u>StartNumber</u>	ListFieldRatingScale
<u>Subject</u>	WebPackage

New Web Object Model Properties (by Object)

Visit the Office Developer Center on the Microsoft Developer Network Web site for the latest information about programming with Microsoft Office FrontPage 2003, including product news, technical articles, downloads, and samples.

The following table lists properties added to the Office FrontPage 2003 object model (sorted by object name).

Object	New Properties
	OptimizeHTMLFlags
BasicList	<u>AllowAttachments, AllowModerate, IsHidden,</u> <u>IsModified</u>
Discussion	<u>AllowAttachments, AllowModerate, IsHidden,</u> <u>IsModified</u>
DocumentLibrary	<u>AllowAttachments, AllowModerate, IsHidden,</u> <u>IsModified</u>
List	AllowAttachments, AllowModerate
ListFieldChoice	<u>AllowFillInChoices</u>
ListFieldComputed	<u>ComputationFormula, ReturnType</u>
ListFieldMultiLine	AllowRichHTML
ListFieldRatingScale	EndNumber, StartNumber
List	<u>IsHidden, IsModified</u>
Survey	<u>AllowAttachments, AllowModerate, IsHidden,</u> <u>IsModified</u>
WebEx	OptimizeHTMLPublishFlags
WebFile	DynamicTemplate
Web	OptimizeHTMLPublishFlags
WebPackage	<u>Author, Company, Subject</u>

New Web Object Model Methods (Alphabetical List)

Visit the Office Developer Center on the Microsoft Developer Network Web site for the latest information about programming with Microsoft Office FrontPage 2003, including product news, technical articles, downloads, and samples.

The following table lists methods added to the Office FrontPage 2003 object model (sorted alphabetically).

New Method	Object
<u>ApplyDynamicTemplate</u>	WebFile
<u>ConvertToField</u>	ListField, ListFieldChoice, ListFieldComputed, ListFieldCounter, ListFieldCurrency, ListFieldDateTime, ListFieldFile, ListFieldInteger, ListFieldLookup, ListFieldMultiLine, ListFieldNumber, ListFieldSingleLine, ListFieldTrueFalse, ListFieldURL, ListFieldAttachments, ListFieldRatingScale
CreateDynamicTemplateState	Application
<u>CreatePackage</u>	Web, WebEx
CreateSearchInfo	Application
DecodeURL	Application
EncodeURL	Application
ImportWebPackage	Web, WebEx
Remove	WebPackage
ShowBordersShadingDialog	Application
<u>ShowFontDialog</u>	Application
ShowHTMLDialog	Application
<u>ShowHyperlinkParameters</u>	Application

ShowImportWebPackageDialog Application	
ShowPickURLDialog	Application
ShowPositionDialog	Application
<u>SplitArgs</u>	Application
<u>UpdateDynamicTemplate</u>	Web, WebEx, WebFile

New Web Object Model Methods (by Object)

Visit the Office Developer Center on the Microsoft Developer Network Web site for the latest information about programming with Microsoft Office FrontPage 2003, including product news, technical articles, downloads, and samples.

The following table lists methods added to the Office FrontPage 2003 object model (sorted by object name).

New Method	Object
	DecodeURL
	CreateDynamicTemplateState, CreateSearchInfo,
	EncodeURL, ShowBordersShadingDialog,
	ShowFontDialog, ShowHTMLDialog,
Application	<u>ShowHyperlinkParameters,</u>
	ShowImportWebPackageDialog,
	ShowPickURLDialog, ShowPositionDialog,
	<u>SplitArgs</u>
ListFieldAttachments	S <u>ConvertToField</u>
ListFieldChoice	ConvertToField
ListFieldComputed	ConvertToField
ListField	ConvertToField
ListFieldCounter	ConvertToField
ListFieldCurrency	ConvertToField
ListFieldDateTime	ConvertToField
ListFieldFile	ConvertToField
ListFieldInteger	ConvertToField
ListFieldLookup	ConvertToField
ListFieldMultiLine	ConvertToField
ListFieldNumber	ConvertToField
ListFieldRatingScale	ConvertToField

ListFieldSingleLine	ConvertToField
ListFieldTrueFalse	ConvertToField
ListFieldURL	ConvertToField
Web	<u>CreatePackage</u>
WebEx	<u>CreatePackage, ImportWebPackage, UpdateDynamicTemplate</u>
WebFile	<u>ApplyDynamicTemplate, UpdateDynamicTemplate</u>
Web	ImportWebPackage
WebPackage	Remove
Web	<u>UpdateDynamicTemplate</u>

Creating Web Sites

Some of the content in this topic may not be applicable to some languages.

Microsoft FrontPage makes FrontPage-based Web site creation as easy as creating a new folder on your hard drive. The key to successful Web site management in FrontPage is planning the structure and design of your Web sites. With most Web servers, you have one Web site, but with FrontPage, you can create as many Web sites as you want, including nested Web sites, called subsites. A FrontPage-based Web site comprises three layers— Web site structure, folder structure, and navigation structure. Click one of the links below to select a specific topic.

Understanding FrontPage Web structure

Understanding FrontPage folder structure

Understanding navigation structure

Creating Web sites programatically

Creating a Web site with the Add method

Creating a Web site with the MakeWeb method

Understanding FrontPage Web structure

Any folder on your Web server can be a Web site with its own folder hierarchy that can include subsites below the original Web site. When you install FrontPage, the program automatically provides a default name for your main Web site. On a disk-based system, the default name is C:\Documents and Settings*user name*\My Documents\My Web Sites for Microsoft Windows XP. You may want to name the individual subsites for the various company names themselves, such as Adventure Works, American Society of Science, Mightyflight Toys, or Coho Winery.

FrontPage provides a variety of Web site templates— corporate, discussion, customer support, and so on. These templates provide the foundation of the structure for each Web site. For example, Adventure Works may want you to establish a full-blown corporate presence for their Web site; and so on. The Web site hierarchy for a disk-based Web site is shown in the following diagram.



The following figure shows the Web site structure in Folders view. The subsites display a small globe within the folder icon.



Understanding FrontPage folder structure

The folder structure in FrontPage behaves in the same manner as the folder structure in Windows Explorer. However, to access these files from Windows Explorer, you have to export them to another location— either your hard drive or server. During the export process, the files are converted to HTML pages. In that sense, opening FrontPage is similar to opening a window to your Web sites. The folder hierarchy for a disk-based Web site is shown in the following diagram.



The following diagram shows the folder structure in Folders view.



Note This diagram displays the same information as the previous one because both folders and Web sites are displayed in the same view, but you can see from the Web site and folder diagrams that they each have their own structure. A Web site or subsite is a folder. However, a folder that is also a Web site contains meta data about that Web site. For example, if you apply a theme to one of your Web sites, all folders within that Web site will have the same theme. However, you can apply different themes to the Web sites on your Web server. When you change a Web site to a folder, you remove special settings that make that folder a Web site, and settings such as the theme change to match the "global" theme for the disk-based or server-based Web site that provides the container for your FrontPage-based subsites.

Understanding navigation structure

You can create files within your Web site, but the navigation structure that links these files to your Web site isn't automatically created when the files are created. However, each subsite can have its own home page. A home page is usually the starting page for any Web site in the navigation structure; but in FrontPage you can create alternate pages that exist at the same navigation level as the home page. You may want to add links to a home page that navigate to the home pages of other subsites that you're maintaining.

The navigation structure contains nodes that link each of the pages in your subsites and provide pointers to the locations of each page in the navigation structure. The navigation structure for a disk-based Web site is shown in the following diagram.



The following diagram shows the navigation structure in Navigation view.



Creating Web sites programmatically

Here's a very simple design for a Web site. The Coho Winery company wants to add a subsite called Wines Around the World that will start with pages for two regions, Spain and France. The folder structure will contain the Coho Winery Web site and the folder for the subsite, Wines Around the World, plus the hidden folder _private, and an Images folder. The navigation structure will contain the Wines Around the World home page (index.htm) and the two child pages (Spain.htm and France.htm— the left and right nodes in the navigation structure).

There are two ways to create FrontPage-based Web sites in Microsoft Visual Basic for Applications. You can use the <u>Add</u> method with the **Webs** collection, or you can use the <u>MakeWeb</u> method with a **WebFolder** object to change an existing folder into a Web site.

Creating a Web site with the Add method

Once you've designed how your Web site is going to look and function, you can use the **Set** statement as shown in the procedure below to create a new Web site.

Note To run the examples in this topic, you must have a Web site called "C:\My Documents\My Web Sites\Coho Winery", or you may substitute a Web site and files of your choice.

```
Private Sub Add()
   Dim myNewWeb As WebEx
   Set myNewWeb = ______
   Webs.Add("C:\My Web Sites\Coho Winery\Wines Around the World
End Sub
```

When you create a Web site with this method, you only create the Web site and its folder; you don't create a complete Web site with all of the folders, pages, and navigation in place. The next step is adding a home page. The following example adds a home page.

Because index.htm or default.htm are file names associated with names commonly used as home pages, FrontPage creates the appropriate navigation structure for a home page whenever you use one of these names. However, if you add further pages using the **Add** method with the **WebFile** object, you will add pages, but FrontPage will not automatically create the navigation structure for you— you will have to add the navigation structure manually as is illustrated in the following example.

Note The following example creates a new subsite in the Coho Winery Web site

and creates two pages in the new subsite: index.htm and Spain.htm.

```
Private Sub AddCompleteWeb()
    Dim myNewWeb As WebEx
    Dim myFiles As WebFiles
    Dim myUrl As String
    Dim myFileOne As String
    Set myNewWeb =
        Webs.Add("C:\My Web Sites\Coho Winery\Wines Around the World
    Set myFiles = myNewWeb.RootFolder.Files
    myFileUrl = _
        "C:\My Web Sites\Coho Winery\Wines Around the World\index.ht
   myFiles.Add(myFileUrl)
   myFileOne = "C:\My Web Sites\Coho Winery\Wines Around the World\
   myFileOne = myFileOne & "Spain.htm"
   myFiles.Add myFileOne
    Call myNewWeb.HomeNavigationNode.Children.Add(myFileOne, "Spain"
        fpStructLeftmostChild)
   myNewWeb.ApplyNavigationStructure
```

End Sub

Notice the last statement— the **ApplyNavigationStructure** method applies the changes that you've made to the navigation structure.

There are several constants you can use in the **Add** method for the **Children** property: **fpStructBaseOnSibling**, **fpStructLeftmostChild**, and **fpStructRightmostChild**. Very simply, these constants inform FrontPage which position you want to apply to the file in the navigation structure— left, right, or base the position on one of the siblings. Here, myFileOne becomes the leftmost child of the home page. The next step is to add the next page, so that you can view the navigation structure in Navigation view. The following adds another page and navigation node to the previous Web site.

```
myFileUrl = _______
"C:\My Web Sites\Coho Winery\Wines Around the World\index.ht
myFileOne = "C:\My Web Sites\Coho Winery\Wines Around the World\
myFileOne = myFileOne & "Spain.htm"
myFileTwo = "C:\My Web Sites\Coho Winery\Wines Around the World\
myFileTwo = myFileTwo & "France.htm"
myFiles.Add myFileOne
myFiles.Add myFileOne
myFiles.Add myFileTwo
Call myNewWeb.HomeNavigationNode.Children.Add(myFileOne, "Spain"
fpStructLeftmostChild)
Call myNewWeb.HomeNavigationNode.Children.Add(myFileOne, "Spain"
fpStructRightmostChild)
myNewWeb.ApplyNavigationStructure
```

End Sub

You can continue to add pages and navigation nodes to your Web site in this way until your Web site is complete. Or, you can create a **For** loop where you iterate through the Web site adding the number of pages and navigation nodes you need to complete the Web site. The following example adds five pages and navigation nodes to a new subsite in the Coho Winery Web site.

Note Creating, moving, or deleting files and folders while attempting to modify the navigation structure may cause some changes to be lost. First, make the changes to the folder structure of the Web site, then make the navigation structure changes, and then apply the navigation structure to the Web site.

```
Private Sub AddDesignerCrystalWeb()
Dim myWeb As WebEx
Dim myParentWeb As WebEx
Dim myFolders As WebFolders
Dim myFolder As WebFolder
Dim myFiles As WebFiles
Dim myNewFiles(4) As WebFiles
Dim myChildNode As NavigationNode
Dim myNewFilename As String
Dim myFileURL As String
Dim myCount As Integer
Dim myBaseURL As String
Dim myWebURL As String
Dim myWebURL As String
Dim myInputMsg As String
Dim myExist As Boolean
```

```
Set myParentWeb = _
        Webs.Open ("C:/My Documents/My Web Sites/Coho Winery/")
   myParentWeb.Activate
    myBaseURL = "C:/My Documents/My Web Sites/Coho Winery/"
   myWebURL = myBaseURL & "Coho Winery Designer Crystal"
    mvExist = False
   myInputMsq =
        "All files will have "".htm"" appended. Type a file name: "
    Set myFolders = Webs(0).RootFolder.Folders
    For Each myFolder In myFolders
        'Check to see if myWebURL already exists.
        If myFolder.IsWeb And myFolder.Url = myWebURL Then
            myExist = True
        End If
    Next
    'Create myWebURL if it doesn't exist.
    If myExist = False Then Webs.Add(myWebURL).Activate
    Set myWeb = ActiveWeb
    Set myFiles = myWeb.RootFolder.Files
    'Create files.
    For myCount = 0 To UBound(myNewFiles)
        myNewFilename = InputBox(myInputMsg)
        myFileURL = myWeb.Url & "/" & myNewFilename & ".htm"
        myFiles.Add myFileURL
        myFiles(myFileURL).Edit
    Next
    'Add to navigation structure.
    For myCount = 0 To UBound(myNewFiles)
        'Check if the current page is index.htm, if so, skip it.
        If myFiles(myCount).Title = "index.htm" Then
            myCount = myCount + 1
        End If
        Set mvChildNode =
          myWeb.RootNavigationNode.Children(0)
        'Add navigation node to the current page.
        myChildNode.Children.Add myFiles(myCount).Url,
            myFiles(myCount).Title, fpStructLeftmostChild
    Next
    myWeb.ApplyNavigationStructure
End Sub
```

Creating a Web site with the MakeWeb method

If you already have an existing folder that you'd like to convert to a Web site, you can use the **MakeWeb** method with a **WebFolder** object as shown in the following example.

Note The following example assumes that Webs(0) is the Coho Winery Web site and that it contains a folder called FolderOne.

```
Private Sub MakeAWeb()
   Dim myWeb As WebEx
   Dim myFolder As WebFolder
   Set myWeb = Webs(0)
   myWeb.Activate
   Set myFolder = ActiveWeb.RootFolder.Folders("FolderOne")
   myFolder.MakeWeb
End Sub
```

You will need to create a navigation structure once PageOne is a subsite of Coho Winery.

Exploring Procedures

This topic is designed to give users who may be familiar with Microsoft FrontPage, but unfamiliar with Microsoft Visual Basic for Applications (VBA), a background on some of the basic concepts in a FrontPage-based programming environment. Programming in FrontPage Visual Basic for Applications provides you with HTML tools in an Microsoft Office programming environment where you can create procedures that perform a task or a series of tasks. For example, you could:

- Create a procedure that retrieves data from a Microsoft Access database and displays the data on your Web page.
- Publish a Microsoft PowerPoint presentation for automatic updates over the Internet.
- Automatically update a Microsoft Excel spreadsheet with input from users responding to your Web site.

This topic provides information on the following VBA programming concepts.

Organize code for modular use

Types of procedures

Public and private procedures

Types of procedure calls

Event procedures and arguments

Create a table in FrontPage from an Access database

Organize code for modular use

Visual Basic procedures provide a way for developers to organize code for modular use. Instead of writing the same calculator function over and over for each program, you can take that segment of code (the calculator function) and compile it into a general program, that can then be accessed by many other programs. In Visual Basic, a block of code is enclosed between a procedure heading and a closure statement— the **Sub** and **End Sub** statements.

The basic syntax of a procedure within Visual Basic is shown in the following code sample.

[Private|Public|Static] Sub procedurename(arguments)

statements

End Sub

To run any of the complete code examples included in the FrontPage Visual Basic for Applications online help, follow these steps:

- 1. Open FrontPage, select **Macro** from the **Tools** menu, and then click **Visual Basic Editor**.
- 2. Double-click Microsoft_FrontPage (or the current project) in the Project window and expand the Modules folder.
- 3. Double-click Module 1 to open the Code window.
- 4. Copy the code block from the documentation, and then paste it into the Code window.
- 5. Click **Run Sub/UserForm** on the toolbar.

Your code will automatically be saved when you close the Visual Basic Editor.

Types of procedures

FrontPage VBA provides two types of procedures, **Sub** and **Function** procedures. **Sub** procedures perform tasks but do not return any values. They may be called from other subroutines or executed in response to an event, such as a mouse click or a keystroke.

Note A **Sub** procedure can be an event procedure, but it can also perform a task without necessarily responding to an event.

The following procedure retrieves the version number of FrontPage from the active Web site but doesn't return the version number to any other procedure.

```
Sub DisplayVersion()
   Dim myWeb As WebEx
   Dim myVersion As String
   myVersion = "FrontPage version number: " & ActiveWeb.Application
End Sub
```

A **Function** procedure also performs tasks, but it can in addition return one or more values as arguments. The following code sample returns the version number of FrontPage to a calling procedure.

```
Function ReturnVersion() As Variant
   Dim varAppVersion As Variant
   varAppVersion = Application.Version
   ReturnVersion = varAppVersion
End Function
```

The variable ReturnVersion now contains the version number of FrontPage. To access this value in the calling procedure, you could write code similar to the following sample.

```
Sub GetAppVersion()
   Dim myAppVersion As Variant
   MsgBox "This version of FrontPage is version " _
   & ReturnVersion
End Sub
```

Alternatively, you could assign the expression ReturnVersion to a variable and

append the variable to the message box statement instead of the function call.

Both **Sub** and **Function** procedures can be called to perform their tasks, depending on whether the procedures are declared **Public** or **Private**.

A *macro* is a third term used to describe code in VBA. As a public **Sub** procedure that doesn't take arguments, a macro may or may not call other **Sub** or **Function** procedures and can be assigned to command bars and shortcut keys or run from the **Macro** dialog box.

Public and private procedures

Visual Basic provides two ways to access a procedure. By default, procedures are public— they can be called from any other procedure in any module within your application. For example, if you've written a procedure that lists images by file name on a Web page, you would want to declare that procedure public so that you could use it across all of your Web sites. However, if you've written a procedure that edits a specific database, you would want that procedure to be available only to the module that handles editing the database— in that case, you can declare the procedure private. Procedures that have been declared private can only be referenced by other procedures within the same module. The function shown previously has been declared a public function in the following code sample and can be called across modules and projects.

```
Public Function ReturnVersion() As Variant
    statements
End Function
```

In contrast, a procedure that is used to edit a database should be declared private.

```
Private Function EditCustomerName(strFirstName As String)
    statements
End Function
```

Types of procedure calls

How do you programmatically run a procedure? You declare it the same way that you would use a keyword, such as **Open**. The following procedure calls the ReturnVersion function and assigns the returned value to a local variable, MyVersion, for the value that is passed to the procedure.

```
Sub TestCall()
Dim MyVersion As Variant
MyVersion = ReturnVersion
End Sub
```

If you didn't have any information to pass from one procedure to another, you would simply declare the procedure name, as shown in the following code sample.

```
Sub TestCall2()
DisplayCompanySplashScreen
End Sub
```

The TestCall2 procedure calls another procedure,

DisplayCompanySplashScreen, which doesn't take any arguments or return any values.

Event procedures and arguments

If you want an event, such as clicking a command button, to trigger the execution of code in cases where you would usually pass a value into the calling procedure, you can execute the results from the function rather than return the results. In this case the ReturnVersion function becomes a subroutine and initiates the display of the version number for the application.

```
Sub ReturnVersion()
   Dim varAppVersion As Variant
   varAppVersion = Application.System.Version
   DisplayMsgBox varAppVersion
End Sub
```

The DisplayMsgBox subroutine shown in the following code sample displays the contents of the variable varGotAppVersion that was passed from the ReturnVersion subroutine.

```
Sub DisplayMsgBox(varGotAppVersion As Variant)
   Dim varDisplayAppVersion As Variant
   varDisplayAppVersion = varGotAppVersion
   MsgBox "This application is version " _
        & varDisplayAppVersion
End Sub
```

An event procedure can now initiate the display of the value that is passed from the ReturnVersion subroutine.

```
Private Sub CommandButton1_Click()
ReturnVersion
End Sub
```

Create a table in FrontPage from an Access database

The following procedure combines objects from the <u>Page object model</u> and the <u>Web object model</u> to retrieve data from an open Microsoft Access database and insert it into a table on a FrontPage-based Web page. The ParseDBTable procedure provides the parameters for the ParseAccessTable function which calls the following functions to create and populate the table:

- AddDBTableToPage— creates a new table
- AddDBRow— inserts a row onto the Web page
- AddMemo— retrieves the memos from the Access database, returns them as bookmarks at the bottom of the page below the new table, and returns the URL to the bookmark

Note The Access database, Northwind.mdb, was used for this example. To run the example, you must have references in the Visual Basic Editor to the Microsoft DAO 3.6 Object Library and the Microsoft Access Object Library. You must also open an Access database before running the example, and you must add a blank temporary file called tmp.htm in the active Web site. If you use a database other than Northwind.mdb, you must specify the database name and table in the ParseDBTable procedure.

```
Function AddDBTableToPage(myPage As PageWindowEx, _
   myTableName As String, myFields As Integer)
   Dim myTable As FPHTMLTable
   Dim myHTMLString As String
   Dim myCount As Integer
   myHTMLString = "
   myTableName & """>" & vbCrLf
   myHTMLString = myHTMLString & "" & vbCrLf
   For myCount = 1 To myFields
      myHTMLString = myHTMLString & "
          myCount & """> " & vbCrLf
   Next myCount
   myHTMLString = myHTMLString & "" & vbCrLf
   myHTMLString = myHTMLString & "" & vbCrLf
   Call myPage.Document.body.insertAdjacentHTML("BeforeEnd", _
      myHTMLString)
```

End Function Function AddDBRow(myDBTable As FPHTMLTable) Dim myHTMLString As String Dim myTableRow As FPHTMLTableRow Set myTableRow = myDBTable.rows(0) myHTMLString = myTableRow.outerHTML Call myDBTable.insertAdjacentHTML("BeforeEnd", myHTMLString) End Function Function AddMemo(myCurrentPage As PageWindowEx, myDBMemo As String, myBkMarkField As String, myIndex) As String Dim myHTMLString As String Dim myMemoBkMark As String Dim myBookMark As FPHTMLAnchorElement myMemoBkMark = myBkMarkField & "_" & myIndex myHTMLString = " Memo #" & _ myIndex & "" & vbCrLf 'Add the bookmark to the page. Call myCurrentPage.Document.body.insertAdjacentHTML("BeforeEnd", myHTMLString) Set myBookMark = myCurrentPage.Document.all(myMemoBkMark) 'Add the memo text to the page. Call myCurrentPage.Document.body.insertAdjacentHTML("BeforeEnd", myDBMemo) AddMemo = "" End Function Function ParseAccessTable(myDBName As String, myTableName As String) 'Access/DAO Declarations. Dim myDBApp As Access.Application Dim myRecordSet As DAO.recordset Dim myDBField As DAO.Field 'FrontPage Page object model declarations. Dim myPage As PageWindowEx Dim myTable As FPHTMLTable Dim myTableRow As FPHTMLTableRow Dim myTableCell As FPHTMLTableCell

```
'Function declarations.
Dim myCount As Integer
Dim myFieldValue As String
Dim myRecordCount As Integer
myRecordCount = 0
'Function constants.
Const myTempPage = "tmp.htm"
'Get the current Access database.
On Error GoTo AccessNotThereYet
    Set myDBApp = GetObject(, "Access.Application")
'Get the database table.
On Error Resume Next
Set myRecordSet = myDBApp.CurrentDb.OpenRecordset(myTableName)
'Add a new page to the current Web site.
Set myPage = ActiveWeb.LocatePage(myTempPage)
myPage.SaveAs myTableName & ".htm"
'Delete the temporary file from Web site.
ActiveWeb.LocatePage(myTempPage).File.Delete
'Add a database-ready table to the page with the proper number o
AddDBTableToPage myPage, myTableName, myRecordSet.Fields.Count
'Get a reference to the table.
Set myTable = myPage.Document.all.tags("table").Item(0)
'Populate the first row.
For myCount = 0 To myRecordSet.Fields.Count - 1
    myTable.rows(0).cells(myCount).innerHTML = "<b>" &
        Trim(myRecordSet.Fields(myCount).Name) & "</b>"
Next
'Populate the rest of the table.
While Not (myRecordSet.EOF)
    AddDBRow myTable
    Set myTableRow = myTable.rows(myTable.rows.Length - 1)
    For myCount = 0 To myRecordSet.Fields.Count - 1
        Set myTableCell = myTableRow.cells(myCount)
        If IsNull(myRecordSet.Fields(myCount)) Then
            myFieldValue = "None"
        Else
            myFieldValue = Trim(myRecordSet.Fields(myCount).Valu
```

```
End If
            If myRecordSet.Fields(myCount).Type = DAO.dbMemo Then
                myFieldValue = AddMemo(myPage,
                    myRecordSet.Fields(myCount).Value, __
                    myRecordSet.Fields(myCount).Name, myRecordCount)
            End If
            myTableCell.innerHTML = myFieldValue
        Next myCount
        myRecordSet.MoveNext
        myRecordCount = myRecordCount + 1
   Wend
   myPage.Save
   myDBApp.Quit
   Exit Function
   AccessNotThereYet:
        Debug.Print Err.Number & ":" & Err.Description
        Resume
End Function
Private Sub ParseDBTable()
   Call ParseAccessTable("Northwind.mdb", "Products")
End Sub
```

Sharing Programming Projects

Microsoft FrontPage doesn't provide the ability to create multiple projects, so sharing a project may be a little confusing at first glance. You can share your Microsoft FrontPage projects using one of the following two methods:

- As a ComAddIn.
- By exporting the .bas, .cls, and .frm modules to a directory on your hard drive (or a server location) so that the user can import the modules to their project in FrontPage.

Sharing as a COMAddIn

For a detailed description of how to develop a COM add-in for FrontPage, see the FrontPage Software Developer Kit, which you can view online or download from the Microsoft Developer Network Web site.

Sharing by exporting the project

You can export your modules from the Visual Basic Editor by selecting the module in the Project window and then pointing to **Export File** on the **File** menu. In the **Export File** dialog box, select the directory you want the file to reside in and FrontPage takes care of the rest.

Note When exporting UserForm files, you must also include the .frx file. This file is used during the import process and does not need to be imported, but does need to be available in the same directory as the other modules.
Understanding Absolute and Relative URL Addressing in Microsoft FrontPage

Microsoft FrontPage uses absolute URL addressing. However, FrontPage does provide a way to change the addressing of URLs between absolute and relative addressing through the <u>MakeRel</u> and <u>MakeAbs</u> methods.

What is an absolute URL?

A URL defines the location of an object. When a URL is absolute, it defines unambiguously where the object is located. For example, http://www.microsoft.com/FrontPage/default.htm is the exact location of the welcome page for Microsoft FrontPage. The object, default.htm, is stored in the FrontPage folder or Web subsite on the Web server, www.microsoft.com.

A disk-based Web site that contains your Web sites on your hard drive might have a base absolute URL such as file:///C:/My Documents/My Web Sites, file:///C:/WINNT/Profiles/your logon alias/Personal/My Web Sites/index.htm, or file:///C:/Documents and Settings/your logon alias/My Documents/My Web Sites.

As Web sites are updated and the structure of a Web site evolves, documents can often be moved from one location to another in a Web site. If you're using a relative address, you may break some links. For example, if you have a document that has a hyperlink to an object, you can use an absolute URL to ensure that the hyperlink always refers to that object. For example, a document such as file:///C:/My Documents/My Webs/mydocument.htm, may contain an absolute URL that refers to the welcome page,

http://www.microsoft.com/FrontPage/default.htm. If the document, default.htm, is moved to another location, the absolute URL for the document will always refer to the object on the Web server at www.microsoft.com in the subsite named FrontPage. If the URL for the hyperlink to default.htm is a relative address, the link could be broken if mydocument.htm is moved to a different subsite or main Web site.

What is a relative URL?

Instead of specifying every piece of a URL that fully defines how to find an object, you can abbreviate a URL to make it "relative" to a current location. Relative URLs are typically used when creating a Web site in which the pages in it refer to other pages on the site. A page such as file:///C:/My Documents/My Web Sites/mydocument.htm can have a relative URL to "newdocument.htm" or to "../My Web Sites2/default.htm". In these two cases, the referred addresses for these files are file:///C:/My Documents/My Web Sites2/default.htm. Notice that the full Web

server or explicit directory isn't specified; the location is based on where the document is located.

Relative URLs are very useful, particularly when constructing a site in one place and then publishing it to another place. For example, you might not want to specify a server name while authoring a Web page, because the server name will change when the Web site is published.

How can an absolute URL be made relative?

URLs can be made relative either to a server or to a page. In order to make a URL relative to a folder or to a Web site, you need to make the URL relative either to a server or to a page, because FrontPage will treat the URL as if it were being made relative to a page in that folder or Web site. Each relative URL, combined with its base URL fully specifies where the object is located.

When you have an absolute address such as

http://www.microsoft.com/mywebsite with an image file called MyPicture.gif residing in the images folder on the Web site, you can create a relative URL for that address by deciding whether the relative address will refer to a server or a page. For a server-relative URL, you'd use "mywebsite/images/MyPicture.gif" with the server name as the base for the relative URL, which fully defines the location of the object. For a page-relative URL, you'd use "images/MyPicture,gif" with the server name and the Web site name as the base for the relative URL, which fully defines the location of the object. For a folderrelative URL, you'd use "MyPicture.gif" with the server name, Web site name, and folder name as the base for the relative URL; this relative address fully defines the location of the object.

A folder-relative URL can also be specified from a different folder, for example, "../images/MyPicture.gif" could be used if the referring document were in another top-level folder on the same Web site and server. The ".." is the notation used to indicate up one folder level from the current URL address.

Programmatic Access to Relative URLs

Once you've decided whether to use server, Web site, or folder-relative URLs, you can determine which objects to use. For server or Web site-relative URLs, you would be working with methods related to the **WebEx** object. For folder-relative URLs, you would be working with methods related to the **WebFolder**

object.

Converting relative and absolute URLs

Recommended use for FrontPage is absolute addressing. However, once you have specified a relative URL within FrontPage, you can use the **MakeAbs** method to convert it into an absolute URL. Similarly, if you have an absolute URL and you want to insert a reference into a document as a relative URL, you can use the **MakeRel** method to convert the address to a relative address. You can also use the **MakeAbs** and **MakeRel** methods to convert a hyperlink for a **WebEx**, **WebFolder**, **WebFileEx**, **NavigationNode**, or **IHTMLDocument2** object.

Exploring the Object Model in FrontPage

The Microsoft FrontPage object model consists of three separate object models — the Application object model, the Web object model, and the Page object model that is based on the Microsoft Internet Explorer 4.0 Document object model.

These object models are designed to provide Web designers with a rich set of programming interfaces to manage individual HTML documents and FrontPagebased Web sites. The object models parallel other Microsoft Office application object models as closely as possible in order to leverage the experience and knowledge users currently have with other programmable Office applications. In addition, the Page object model is compatible with the Document object model of Microsoft Internet Explorer 4.0 and later. The Page object model also leverages the experience and knowledge of Web developers who program dynamic HTML (DHTML) using script.

Differences in the programming interfaces

FrontPage provides a design-time user experience versus a run-time user experience, because HTML pages are usually viewed in a browser rather than in FrontPage. Other Office applications provide a run-time user experience that contains an environment where the developer can create solutions for the user that run within the application. With FrontPage, you can create utilities for use during the design phase of the HTML document or Web site and, once you have designed and built your Web site, you can use Microsoft Visual Basic Scripting Edition (VBScript), Microsoft JScript, and DHTML to create interactive effects for use during run-time when the page is viewed in a browser.

Visit the Office Developer Center on the Microsoft Developer Network (MSDN) Web site for the latest Microsoft FrontPage development information, including new technical articles, downloads, samples, product news, and more.

Application object model functionality

The Application object is the top-level object in the Web object model hierarchy and provides access to all of the objects in FrontPage, such as the active Web site, active page, add-ins, command bars, system, and a collection of Web sites.

FrontPage Page object model functionality and restrictions

The Page object model provides access to the HTML in a page and has most of the functionality of the Internet Explorer Document object model, with some exceptions. Some objects and members of the Internet Explorer Document object model provide functionality that is only useful in an Internet Explorer environment. Those objects and members are not implemented for use in the Page object model.

Web object model functionality

The Web object model provides programmatic management of multiple FrontPage-based Web sites and their contents. This object model is based on conventions used by the applications in the Microsoft Office System and Microsoft Visual Basic for Applications (VBA). The Web object model also provides support for Microsoft Visual SourceSafe when accessed through FrontPage.

Object model interaction

Most of the programming for FrontPage-based Web sites will contain a combination of Web and Page object model programming elements. For additional information and examples of this, see one of the following topics.

Accessing Framesets with Microsoft Visual Basic

Modifying Pages Programmatically

Programming Dynamic Elements

Accessing Framesets with Microsoft Visual Basic

Frames are an important part of the design of a Web site. Microsoft FrontPage provides support for programming the content of frames. Click one of the following links for more information on a particular topic.

Exploring frames

Role of a frameset

Accessing HTML tags

Dynamic frame sources

Iterating all frames in a page window

Changing Meta tag content to another character set

Exploring frames

Accessing frames within a frameset in FrontPage is relatively straightforward as long as you keep in mind that each frame accesses its own page and that you access the contents of each page through the Page object model. The two windows involved in displaying the frames are the **PageWindowEx** object and **FPHTMLWindow2** objects. The equivalent expression for accessing the contents in the window associated with the active frame is Set myActiveFrameInFrameset = ActivePageWindow.Document. The **ActivePageWindow** object accesses the page window for the frame, and the **Document** object accesses the **FPHTMLWindow2** object.

Note You may get permission denied errors if you try to access objects while in HTML view. When you want to add code or text to a document object, you must set the **FpPageViewMode** constant for the <u>ViewMode</u> property to **fpPageViewNormal**. The value for the view mode cannot be set to **fpPageViewHtml** or **fpPageViewPreview**. Alternatively, in the FrontPage user interface (UI), you cannot have the **HTML** tab or the **Preview** tab open in Page view.

Role of a frameset

A frameset is the container for all of the frames in a Web window. Each frame is contained in an individual page window and has an individual page associated with it. By accessing the **FrameWindow** object from the Web object model, you access an **FPHTMLWindow2** object that contains the frame page document. From these objects you can access the windows, documents, and frames of the frameset. In the following statement, myFrameset is an **FPHTMLWindow2** object that returns the frames page document. From this object, you can access the <FRAME> and <FRAMESET> tags, or the window or document objects.

Set myFrameset = ActivePageWindow.FrameWindow

This statement returns an **FPHTMLWindow2** object through the Page object model. The **Document** property of myFrameset accesses the Page object model for the page that is equivalent to accessing the frames page **HTML** tab in Page view in FrontPage.

Note The Frames Page **HTML** and **No Frames** tabs are only available when frames exist on the current page.

Accessing HTML tags

You can access the same information that a frames tag accesses by declaring an object as an FPHTMLFrameElement. Some of the properties and methods available for this object include **border**, **borderColor**, **click**, **frameBorder**, **frameSpacing**, **innerHTML**, **innerText**, **insertAdjacentHTML**, and **insertAdjacentText**.

Dim myFramesElements() As FPHTMLFrameElement

You can access the information for a frameset tag by declaring an FPHTMLFramesetSite object.

Dim myFramesetSite As FPHTMLFramesetSite

Dynamic frame sources

You can dynamically change the frame source in the HTML code by using the following statements. This code sets the frame source to a new URL, Inventory_1stQuarter.htm.

```
Dim myDoc As Object
Set myDoc = ActivePageWindow.FrameWindow.Document
myDoc.all.tags("frame").Item(0).src = _
    "Inventory_1stQuarter.htm"
```

Iterating all frames in a page window

To access the properties of the frameset elements that reside in a particular frames page, you must access the **FPHTMLDocument** object through the **Document** property. The following example iterates through the frameset and frame elements for the active frameset in Microsoft FrontPage. The frameset array (myFSElements) comprises each <FRAMESET> tag on the frames page. The frame windows array (myFramesElements) comprises each <FRAME> tag on the frames page. The frame windows array (myFramesWindows) comprises each **FPHTMLWindow2** object that points to each frame. You populate each of the arrays by iterating through their respective tags or objects. Once the arrays are populated, you change the **frameSpacing** property in the frameset element to "10", the **borderColor** property to "red", and change various other properties in the document.

```
Private Sub AccessFramesPage()
Dim myFPWindow As FPHTMLWindow2
Dim myFSElements() As IHTMLFrameSetElement
Dim myFramesWindows() As FPHTMLWindow2
Dim myFramesElements() As FPHTMLFrameElement
Dim myStyle As FPHTMLStyle
Dim i As Integer
Set myFPWindow = ActivePageWindow.FrameWindow
ReDim myFSElements(myFPWindow.Document.all.tags("FRAMESET").length)
ReDim myFramesElements(myFPWindow.Document.all.tags("FRAME").length)
ReDim myFramesWindows(myFPWindow.frames.length)
For i = 0 To UBound(myFSElements)
    Set myFSElements(i) =
        myFPWindow.Document.all.tags("FRAMESET").Item(i)
Next i
i = 0
For i = 0 To UBound(myFramesWindows)
    Set myFramesWindows(i) = myFPWindow.frames(i)
Next i
i = 0
For i = 0 To UBound(myFramesElements)
    Set myFramesElements(i) =
        myFPWindow.Document.all.tags("FRAME").Item(i)
Next i
```

```
myFSElements(0).frameSpacing = "10"
myFramesElements(0).borderColor = "red"
With myFramesWindows(2).Document
.bgColor = "green"
.body.innerHTML = " Added by FP Programmability"
Set myStyle = .all.cool.style
myStyle.backgroundColor = "white"
myStyle.display = False
myStyle.display = False
myStyle.textDecorationUnderline = True
myStyle.Font = "Tahoma, 24"
myStyle.fontStyle = "italic"
End With
End Sub
```

Changing Meta tag content to another character set

You can change all of the content-type META tags to a different character set (Central European) as shown in the following code sample. The current character set is shown in the **Language Settings** dialog box (available for page properties).

Note The entire content-type META tag contains a string similar to the following:

```
content = "text/html; charset = windows-1252"
```

The character set is "windows-1252" and is the default character set for U.S. English.

Each time the program iterates through the loop, you access the next frame in the **frameWindow** object, which is the same as accessing each HTML frames tag in succession. However, the **Frames** collection does not support the **For...each** construct. You cannot access HTTP-EQUIV type META tags via their name; you must instead use an index as shown in the following example. The expression beginning with myContentType.Content sets the character set to Central European.

Note FrontPage places the content type in <META> tag zero(0).

```
Private Sub ChangeCharSet()
   Dim myFrames As IHTMLFramesCollection2
   Dim myFrame As FPHTMLWindow2
   Dim myHTTPEquiv As String
   Dim myContentType As Object
   Dim myCount As Integer
   Set myFrames = ActivePageWindow.FrameWindow.frames
   Set myFrame = ActivePageWindow.FrameWindow.frames(0)
   myHTTPEquiv = 0
   For myCount = 0 To myFrames.Length - 1
      Set myFrame = myFrames(myCount)
      Set myFrame = myFrames(myCount)
      Set myContentType = _______myFrame.Document.all.tags("meta").Item(myHTTPEquiv)
      myContentType.content = _______"text/html; charset=iso-8859-2"
```

Next myCount End Sub

Accessing the Object Model in FrontPage from Design-Time Controls

Some of the content in this topic may not be applicable to some languages.

A design-time control (DTC) is one of a class of Microsoft ActiveX controls that is exclusively used at design-time as an aid to authoring Web site content. DTCs function just like embedded wizards— they can be edited to modify the output that the DTC generates on the Web page. Once the file is closed, the DTC becomes inactive— the text that the DTC generated has been embedded into the file.

How are design-time controls different from ActiveX controls?

Design-time controls do not contain a binary run-time component, but do have a special interface that provides the ability for the design-time control to persist and generate text. Design-time controls and ActiveX controls can co-exist on the same web page. For more information on DTCs, see the documentation on the Microsoft workshop Web site for design-time controls.

How do design-time controls differ from Visual InterDev?

FrontPage and Microsoft Visual InterDev are compatible in most areas. For design-time controls, FrontPage provides full access to the Document object model that's available in Microsoft Internet Explorer. The Web object model is also available for design-time controls and provides a rich array of objects, events, methods, and properties. FrontPage also provides support for removing tags (or text) that is inserted into a document by design-time controls.

Properties supported by FrontPage that are not supported by Microsoft Visual InterDev:

- Preview— returns a string containing the controls preview text.
- Tag— returns the tag context for the preview string.

Built-in options in Visual InterDev that FrontPage doesn't support:

- QueryBuilder
- Data connections
- Visual InterDev object model for .asp pages

How to access the FrontPage object model from a design-time control

A design-time control is a type of ActiveX control, and thus accesses its host's object model similarly to other controls (see FPHTMLObjectElement object). The design-time control has a **UserControl** property that returns the **UserControl** object. This object in turn has an **Extender** property that returns the environment where the ActiveX control resides. The expression, UserControl.extender returns the environment for the specified DTC.

The **Extender** property returns **FPHTMLObjectElement** object and provides access to all of the properties and methods in the **FPHTMLObjectElement** object. One of these properties is the <u>Document</u> property that accesses the FrontPage Page object model for the document where the design-time control resides. The following statements illustrate how to access the **Document** property from a DTC.

```
Dim myDTC As FPHTMLObjectElement
Dim myDoc As FPHTMLDocument
Dim myPageWindow As PageWindowEx
Set myDTC = UserControl.Extender
Set myDocument = myDTC.document
Set myPageWindow = myDoc.parentWindow.external
```

The last Set statement illustrates how to access the Web object model from the Page object model.

Coding in a Windowless Environment

In Microsoft FrontPage, it is possible to create a windowless environment to cut down on User Interface (UI) overhead, but there are some drawbacks to working in such an environment. For example, opening a page with the **ViewMode** property set to **fpPageViewNoWindow**, creates a windowless environment where window elements that require the UI won't work correctly. If you open a page without a window and try to access it using code such as PageWindow.Document.ParentWindow, the resulting code returns nothing instead of returning an **FPHTMLWindow2** object. The following statements provide access to windowless pages.

```
Application.LocatePage(DocumentUrl As String, ______
ViewMode As FpPageViewMode) As PageWindowEx
Web.LocatePage(DocumentUrl As String, ViewMode As FPPageViewMode) ______
As PageWindowEx
File.Edit(ViewMode As FpPageViewMode) As PageWindowEx
```

The following table describes the **PageWindowEx** methods and properties and the objects or error messages they return. Some methods and properties of the **PageWindowEx** object are disabled in a windowless environment and will return the results shown in the following table.

Disabled Method or Property of the PageWindow Object	Returned Object or Error Message
ActiveDocument	Returns "object or with variable not set" error message.
ActiveFrameWindow	Returns "object or with variable not set" error message.
Caption	Returns the URL for the PageWindow object.
Document	Returns a Document object.
File	Returns a File object.
ViewMode	This property can not be set. It returns the constant fpPageViewNoWindow .
Visible	Returns False .

Web	Returns the Web object if the object was located using the following statements.
	Web.LocatePage
	File.Edit
ApplyTheme	Returns a run-time error.
Close(ForceSave)	Closes the page window.
Refresh(ForceSave)	Refreshes the page window.
Save(ForceOverwrite)	Saves the page window.
SaveAs(ForceOverwrite)	Saves the page window to a new URL.



Modifying Pages Programmatically

This topic illustrates how to work with the **FPHTMLDocument** object elements within the Page object model either to change the contents of a page or simply to access the contents of a page for verification.

Microsoft Visual Basic is a powerful tool that you can use to modify content in your Web pages. As you browse through the Object Browser, you'll see many of the same types of components that you're familiar with in Microsoft Internet Explorer.

Tip

To only view the programming elements that are compatible with the FrontPage <u>Page object model</u>, select "FrontPageEditor" as the object library in the Object Browser.

You can access the HTML elements of a page using the <u>ActiveDocument</u> or <u>Document</u> properties that return the document object via the <u>PageWindowEx</u> object. For example, the following statement changes the background color for the page in the active page window.

```
ActivePageWindow.Document.bgColor = "DarkBlue"
```

The following example checks for a specific hyperlink (index.htm) within the active document. If the hyperlink is found, the procedure exits, but if the hyperlink isn't found, the procedure first checks if the active document is index.htm and, if not, the hyperlink is added at the end of the document.

```
Private Sub VerifyIndexLink()
   Dim myDoc As FPHTMLDocument
   Dim myLinks As Variant
   Dim myLink As Variant
   Dim myNumberOfLinks As Integer
   Dim myAddLink As Boolean
   Dim myLinkName As String
   Dim myLinkName2 As String
   Set myDoc = ActivePageWindow.Document
```

```
Set myLinks = myDoc.Links
myNumberOfLinks = myLinks.length
myLinkName = "index.htm"
myLinkName2 = """" & myLinkName & """"
For Each myLink In myLinks
    If myLink = myLinkName Then
        myAddLink = True
        Exit For
    End If
Next
If myAddLink = False And myDoc.nameProp <> "index" Then
        Call myDoc.body.insertAdjacentHTML("BeforeEnd", "<a href=" _
            & myLinkName2 & ">" & myLinkName & "</a>")
    ActivePageWindow.Save
End If
End Sub
```

Note Notice that in the last **If** statement the active page window is saved before exiting the procedure. This would be a good statement to add to the **OnPageClose** event.

Understanding WebWindows and PageWindows

In Microsoft Internet Explorer, the window object is the root of the object hierarchy. In Microsoft FrontPage, the **Application** object is the root of the object hierarchy and the **WebWindowEx** object is an object in the third tier of the object hierarchy belonging to the collection of **WebWindowEx** object (second tier). The **Document** property is accessed via the **PageWindowEx** object (belonging to the collection of **PageWindowEx** object. The object hierarchy for the **WebWindows** collection is shown in the following diagram.

WebWindows WebWindowEx
PageWindows
PageWindowEx

What is a WebWindowEx object?

The **WebWindowEx** object is the window container for the Web site. When you open a Web site in FrontPage, you see the Web site in its "web" window. FrontPage opens a new window for each Web site you open.

What is a PageWindowEx object?

The **PageWindowEx** object is the container for the web page. When you open a page in a web window, that page is contained in a page window (visible in the right pane). FrontPage opens a new page window for each page you open; however each **PageWindowEx** object is contained within the **WebWindowEx** object for the individual Web site. Each page that you've opened is visible on the Windows menu in FrontPage.

Where does the DispFPHTMLDocument object fit in?

The **<u>FPHTMLDocument</u>** object contains the HTML elements for a page. The elements for the document are available using an expression such as

ActivePageWindow.Document.element where element is one of the HTML elements available in FrontPage. Note that not all HTML elements have corresponding properties that you can use to access the element. For example, there is no head property to access the HEAD element. In these cases, use activepagewindow.Document.all.tags.item("element") to return the object that corresponds to the specified element.

Returning an Object from a Collection

The **Item** property returns a single object from a collection. The following statements set the fileOne variable to a **WebFile** object that represents the first file in the **Files** collection.

Note All collections in the Microsoft FrontPage Web and Page object models are zero-based, so you access the first item in the collection by using a zero.

```
Dim fileOne As WebFile
```

Set fileOne = ActiveWeb.RootFolder.Files.Item(0)

The **Item** property is the default property for most collections, so you can omit the **Item** keyword as shown in the following statement.

Set fileOne = ActiveWeb.RootFolder.Files(0)

Named objects

Although you can usually specify an integer value with the **Item** property, it may be more convenient to return an object by name. The following example edits a file named Web Sales.htm in the active Web page.

ActiveWeb.RootFolder.Files("Web Sales.htm").Edit

Using Events to Control Actions

There are two types of events in Microsoft FrontPage— events that are raise from the Application and Web object models and events that are raised from the Page object model.

Application events

The events for the **Application** object model can be used to control under what conditions a Web site is published, whether you want to save a page whenever the **OnPageClose** event is fired, or whether you want to set styles, fonts, or backgrounds whenever a new page is created.

Page events

In the Microsoft Internet Explorer DHTML object model, event handlers are created using scripts for use at run-time.

However, in the FrontPage Page object model, you're programming with events and objects that are compatible with Internet Explorer, but for use at design time. To program events for runtime, you can use the standard Microsoft Visual Basic 5.0 (or higher) keywords to access the Page object model events just as you would to access the Web object model events. This method combines the two techniques described previously. The following example catches the <u>onclick</u> event for a hyperlink in FrontPage.

In the Visual Basic Editor, insert a class module and name it CatchOnClick. Add the following code to the class module.

```
Dim WithEvents eAnchor As FPHTMLAnchorElement
Dim WithEvents eDoc As FPHTMLDocument
Dim e As IHTMLEventObj
Private Sub Class_Initialize()
Set eDoc = ActiveDocument
Set eAnchor = eDoc.links(0)
End Sub
Private Function eAnchor_onclick() As Boolean
Set e = eAnchor.Document.parentWindow.event
If (MsgBox("OnClick Event for " & e.srcElement.tagName & _
    " would you like to cancel the event bubbling?", _
    vbYesNo) = vbYes) Then
    e.cancelBubble = True
    e.returnValue = False
Else
    e.cancelBubble = False
    e.returnValue = True
End If
End Function
Private Function eDoc_onclick() As Boolean
MsgBox "OnClick event for the Document object"
End Function
```

Next add a standard module and add the following code.

```
Public e As CatchOnClick
Sub GetClick()
Set e = New CatchOnClick
End Sub
```

Note To run the example, perform the following steps:

- Add a hyperlink to a page in FrontPage.
- Run the GetClick procedure to create a global instance of the CatchOnClick event handler class.
- Click the hyperlink.

A prompt is displayed stating that the **onclick** event fired. The prompt also queries the user to find out whether the program should pass the event on up the event chain. If Yes is chosen, the **onclick** event is passed up to the document object to be handled.

To control which document object the event is passed to, you must set both the **cancelBubble** and **returnValue** properties. The **cancelBubble** event works to cancel the event from going any farther up the event chain. Set the **cancelBubble** property of the **IHTMLEventObject** to **True** when you don't want the **onclick** event to be passed up to the next level of **onclick** events, otherwise, set the **cancelBubble** property to **False**. For example, if you have an image that has an **onclick** event placed on a document, which also has an **onclick** event, you would set the **cancelBubble** property for the **IHTMLEventObj** object to **True** for the image, if you don't want the **onclick** event to be passed on up to the document **onclick** event.

The **returnValue** property is used to control the default action taken by FrontPage when an event fires. Using the previous example of an image placed on a document, if the **returnValue** property for the **IHTMLEventObject** for the image is set to **False** in the **onclick** event, then the shortcut menu would be disabled (because the right-click context menu is the default action for the **onclick** event).
Managing Publishing with MetaTags

Automate publishing your Web site by using a combination of events such as **OnBeforeWebPublish** and meta data that can be obtained by accessing the data in the **Properties** collection. For example, before publishing a Web site, you might want to check the meta data for all of the pages in the Web site. To check if Microsoft FrontPage generated all of the pages in your Web site, use the following code in the **OnBeforeWebPublishing** event, along with an event handler.

```
Private Sub CheckIfFP()
    Dim myFiles As WebFiles
    Dim myFile As WebFile
    Dim myMetaTags As MetaTags
    Dim myMetaTag As Variant
    Set myFiles = ActiveWeb.RootFolder.Files
    For Each myFile In myFiles
        Set myMetaTags = myFile.MetaTags
        'Check for any text files.
        If myMetaTags.Count = 0 And _
            myFile.Extension <> ".asa" Then
            MsgBox myFile.Name & " was not generated by FrontPage."
        End If
        'Check all web pages.
        For Each myMetaTag In myMetaTags
            If myMetaTag = "generator" Then
              If myFile.Properties("vti_generator") = _
                "Microsoft FrontPage 4.0" Then
                Exit For
              Else
                MsgBox myFile.Name & " was not generated by FrontPag
              End If
            End If
        Next
    Next
End Sub
```

You can also check the value of the **vti_donotpublish** property key before publishing. If the document is a draft or a document that isn't to be published, the **vti_donotpublish** property key will be set to **True**. The following example checks the value of the **vti_donotpublish** property key.

```
Dim myFiles As WebFiles
Dim myFile As WebFile
For Each myFile In myFiles
    If myFile.Properties("vti_donotpublish") = True Then
        MsgBox "Do not publish " & myFile.Name
    End If
Next
```

The **vti_donotpublish** property key can be used to disable publishing. When publishing is complete, the file will not be published to the server. The following example disables publishing for the first file in the **Files** collection.

```
Sub PublishThisFile(myFileName As String, myStatus As Boolean)
Dim myFile As WebFile
Set myFile = ActiveWeb.LocateFile(myFileName)
Call myFile.Properties.Add("vti_donotpublish", Not (myStatus))
myFile.Properties.ApplyChanges
End Sub
Private Sub PublishFile()
PublishThisFile Activeweb.RootFolder.Files(0), False
End Sub
```

Programming Dynamic Elements in Microsoft FrontPage

The objects, methods, and properties in Microsoft Internet Explorer are usually designed for run-time use. In Microsoft FrontPage, most of the elements are designed for design-time only. You can programmatically add content to an existing document, access selections on a page, create a scripting element, or modify an existing table.

Adding content to a document using a TextRange

Accessing user selections using a TextRange

Adding script to a page

Accessing tables

Adding content to a document using a TextRange

You can programmatically add HTML content to a Web page by creating a text range and adding the new content to the page as shown in the following example. The text range is created from the BODY element of the myDocument variable. If the value in the myClearPage parameter is **True**, then the entire content between the opening and closing BODY element is replaced with the new HTML content, otherwise the new content is appended to the original content.

```
Public Function AddHTMLToPage(myDocument As Object,
    myHTMLText As String, myClearPage As Boolean) As Boolean
    Dim myRange As IHTMLTxtRange
    Dim myBodyText As FPHTMLBody
    On Error GoTo CannotAddHTML
    'Create a TextRange object
    If myClearPage Then
        Set myRange = \_
            myDocument.all.tags("BODY").Item(0).createTextRange
        'Clear the current document
        Call myRange.pasteHTML("")
        myRange.collapse False
        Set myRange = Nothing
    End If
    Set myBodyText = myDocument.all.tags("BODY").Item(0)
    myBodyText.innerHTML = myBodyText.innerHTML & myHTMLText & vbCrL
   AddHTMLToPage = True
    Exit Function
CannotAddHTML:
    AddHTMLToPage = False
End Function
Sub AddNewHTML()
    Dim myHTMLString As String
    Dim myBodyElement As FPHTMLBody
   myHTMLString = "<B> <I> New Sale on Vintage Wines! </I> </B>" &
    If AddHTMLToPage(ActivePageWindow.Document, myHTMLString, True)
        Set myBodyElement = _
```

ActivePageWindow.Document.all.tags("BODY").Item(0) End If End Sub

Accessing user selections using a TextRange

You can use the **IHTMLTxtRange** object to select HTML objects or manipulate a user selection on a specified document. The following example applies a background color to the current selection.

```
Private Sub ApplyStyleToSelection()
   Dim myRange As IHTMLTxtRange
   Set myRange = ActiveDocument.selection.createRange
   myRange.parentElement.style.backgroundColor = "SkyBlue"
End Sub
```

Adding script to a page

Scripting in Microsoft FrontPage Visual Basic for Applications is easy. Just load the script into a **String** variable and insert the **String** to the HEAD element for the page. (The HEAD element is accessed using an **IHTMLElement** object.) Once the **String** has been added to the page, it is a valid scripting element and can be accessed through the **FPHTMLScriptElement** object and modified. The following code adds a script element to the current page, verifies that the script was added, adds a query to the user with OK and Cancel buttons, and then prints some of the script element properties in the Immediate window of the Visual Basic Editor.

```
Private Sub CreateAScript()
    Dim myScriptElement As FPHTMLScriptElement
    Dim myHTag As IHTMLElement
    Dim myBodyTag As IHTMLElement
    Dim myBodyString As String
    Dim myHTMLString As String
    Dim myText As String
    'Build a script tag construct.
   myHTMLString = myHTMLString & "<script language=""VBScript"">" _
     & vbCrLf
   myHTMLString = myHTMLString & "Function doOK" & vbCrLf
   myHTMLString = myHTMLString & _
      "msgbox ""Please wait, an order form is being generated..."""
      vbCrLf
    myHTMLString = myHTMLString & "End Function" & vbCrLf & vbCrLf
   myHTMLString = myHTMLString & "Function doCancel" & vbCrLf
   myHTMLString = myHTMLString &
      "msgbox ""Exiting ordering process.""" & vbCrLf
   myHTMLString = myHTMLString & "End Function" & vbCrLf
   myHTMLString = myHTMLString & "</script>" & vbCrLf
    'Build a call tag construct.
    myBodyString = "<CENTER>" & vbCrLf
    myBodyString = myBodyString & _
      "<BUTTON onclick=""doOK()"">OK</BUTTON>" & vbTab
    myBodyString = myBodyString & _
      "<BUTTON onclick=""doCancel()"">Cancel</BUTTON>" & vbCrLf
   myBodyString = myBodyString & "</CENTER>"
    'Add text to the document
    myText = "I'd like to order some vintage wines."
```

```
'Access the HEAD element.
     Set myHTag = ActivePageWindow.Document.all.tags("HEAD").Item(0)
    'Append the script element to the HEAD element (myHTag).
    myHTag.innerHTML = myHTag.innerHTML & myHTMLString
    'Verify that the script element was added.
    If ActivePageWindow.Document.scripts.length = 1 Then
        'Access the script element just added.
        Set myScriptElement = ActivePageWindow.Document.scripts(0)
        'Print script element properties to the Immediate window.
        'JScript only: the next statement gets the FOR= attribute fr
        'the JScript, otherwise an empty string prints in the Immedi
        'window.
        Debug.Print myScriptElement.htmlFor
        'Retrieve the content of the script.
        Debug.Print myScriptElement.outerHTML
        'Check scripting language.
        Debug.Print myScriptElement.language
    End If
    'Add a query to the user and call the script element.
   ActiveDocument.body.insertAdjacentHTML "BeforeEnd", _
      "<B><I>" & myText & "</B></I><P>" & myBodyString
End Sub
```

Accessing tables

Anyone who has created tables and worked with their contents in HTML will find it easy to use Microsoft Visual Basic to access tables. The following program accesses a table on the current page and inserts a cell.

```
Sub AccessTables()
   Dim myTable As FPHTMLTable
   Dim myRow As FPHTMLTableRow
   Dim myCell As FPHTMLTableCell
   'Get the table.
   Set myTable = ActiveDocument.all.tags("TABLE").Item(0)
   'Get the first row.
   Set myRow = myTable.rows(0)
   MsgBox myRow.cells.Length
```

```
'Get the first cell.
Set myCell = myRow.cells(0)
MsgBox myCell.Width
'Add a new cell to the first row.
Set myCell = myTable.rows(0).insertCell(myRow.cells.Length)
End Sub
```

Managing Source Control Projects in Microsoft FrontPage

Microsoft FrontPage provides two methods of versioning, FrontPage Light Weight source control (also known as Microsoft Office-style locking or FrontPage-based locking) and Microsoft Visual SourceSafe. FrontPage Light Weight source control is the default versioning method for source control projects in FrontPage.

Versioning provides a measure of control over users who maintain pages on your Web sites. Both source control methods provide checkin, checkout, undocheckout. Visual SourceSafe provides other versioning capabilities, such as version tracking and rollback features.

Creating a source control project

To create a new source control project, you set the **RevisionControlProject** property to the path of the project (for Visual SourceSafe) or to <FrontPagebased Locking> as shown in the following statement.

ActiveWeb.RevisionControlProject = "<FrontPage-based Locking>"

Assuming that the **String** "\$/Coho Winery" is a valid Visual SourceSafe project, the following statement assigns the active Web site to a Visual SourceSafe project.

```
ActiveWeb.RevisionControlProject = "$/Coho Winery/Inventory"
```

The following example creates a source control project and checks out two files.

Note To run this example, you must have a Web site called "C:\My Documents\My Web Sites\Coho Winery". You may create two files called "index.htm" and "footnote.htm" or substitute file names of your choice.

```
Private Sub CreateSourceControl()
   Dim myWeb As WebEx
   Dim myProject As String
   Dim myFile1 As WebFile
   Dim myFile2 As WebFile
   Set myWeb = Webs.Open("C:\My Documents\My Web Sites\Coho Winery"
   Set myFile1 = myWeb.RootFolder.Files("index.htm")
   Set myFile2 = myWeb.RootFolder.Files("footnote.htm")
   myProject = "<FrontPage-based Locking>"
   myWeb.RevisionControlProject = myProject
   myFile1.Checkout
   myFile2.Checkout
End Sub
```

The **Checkout** method provides a Boolean force checkout argument for administrators.

Removing a source control project

Once a project is completed, you may decide to remove versioning. To do this, set the **RevisionControlProject** property to an empty **String** ("") as shown in the following statement.

myWeb.RevisionControlProject = ""

Switching between FrontPage Light Weight and Visual SourceSafe projects

To switch between these two types of versioning, you must first set the **RevisionControlProject** property to an empty **String** ("") as shown in the following statement.

myWeb.RevisionControlProject = ""

Lists Collection

Multiple objects Lists

Represents the collection of all **List** objects in the current Web site. Lists allow information to be shared and exchanged between different users and different Web sites.

This object is supported only by Web pages or sites that are based on Microsoft SharePoint Services.

Using the Lists collection

Use the **Lists** property of the **WebEx** object to return the collection of all lists in the Web site. Use **Lists.item** (*index*), where *index* is either the name of the list or its numeric position within the collection, to return a single **List** object.

The following example displays the names of all lists in the active Web site. If the active Web site does not contain any lists, a message is displayed to the user.

```
Sub ListAllLists()
'Displays the names of all lists in the collection
    Dim lstWebList As List
    Dim strName As String
    'Check if any lists exist
    If Not ActiveWeb.Lists Is Nothing Then
        'Cycle through lists
        For Each lstWebList In ActiveWeb.Lists
            'add list names to string
            If strName = "" Then
                strName = lstWebList.Name & vbCr
            Flse
                strName = strName & lstWebList.Name & vbCr
            End If
        Next
        'Display names of all lists
        MsgBox "The names of all lists in the current Web site are:"
               & vbCr & strName
    Else
        'Other wise display message to user
        MsqBox "The current Web site contains no lists."
    End If
```

```
End Sub
```

Use the **Lists**.**Add** method to add a new list to the **Lists** collection. The following example adds a new list of type **fpBasicList** called NewShare to the active Web site.

```
Sub NewList()
'Adds a new list to the current Web site
```

End Sub

MetaTags Collection

WebFile MetaTags

An array of **property key/value** pairs. Each item in the **MetaTags** collection represents a META tag contained on an HTML page in Microsoft FrontPage. There is no MetaTag object.

Note META tags generated by FrontPage, such as META tags for a theme or border, won't show up in the **MetaTags** collection. The **MetaTags** collection is only propagated after the file is saved. For example, if you add a new META tag to a page by using the Code view or by programmatically using the Page object model in Microsoft Visual Basic for Applications, you won't be able to view the **property key/value** pairs until after you save the page. To save space in the META dictionary, you can use the following methods to disable the META tag store:

- Set the DisableMetaTagStore key on a per-service basis
- Set the DisableMetaTagStore key as a global setting under HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Shared Tools\Web Server Extensions\All Ports
- Set the vti_disablemetatagstore property

The DisableMetaTagStore key functions in the same way as other server settings; see the *Server Extensions Resource Kit* for more information about server settings.

Using the MetaTags Collection

Use the **MetaTags** property to return the **MetaTags** collection.

Note In the meta data for FrontPage, the **http-equiv** attribute can be used in place of the **name** attribute. FrontPage doesn't use the value of the **http-equiv** attribute in response message headers. The syntax for META data is as follows:

```
<META
CONTENT=description
HTTP-EQUIV=text
NAME=text
TITLE=text
URL=url
>
```

You can also return a list of the META tags that exist in the active Web site by accessing the file structure through the root folder of the **ActiveWeb** object, as shown in the following example.

Note To run this example, create a form with a text box called txtMetaTags (set to multiple lines) and a command button called cmdGetMetaTagInfo, and then copy the example into the code window.

```
Private Sub cmdGetMetaTagInfo_Click()
Dim myWeb As WebEx
Dim myFiles As WebFiles
Dim myFile As WebFile
Dim myMetaTags As MetaTags
Dim myMetaTag As Variant
Dim myFileName As String
Dim myMetaTagName As String
Dim myReturnInfo As String
Set myWeb = ActiveWeb
Set myFiles = myWeb.RootFolder.Files
With myWeb
    For Each myFile In myFiles
        Set myMetaTags = myFile.MetaTags
        For Each myMetaTag In myMetaTags
            myFileName = myFile.Name
            myMetaTagName = myMetaTag
```

```
myReturnInfo = myFileName & ": " & myMetaTagName
txtMetaTags.Text = txtMetaTags.Text & myReturnInfo
Next
Next
txtMetaTags.SetFocus
txtMetaTags.CurLine = 0
End With
End Sub
```

Use the **<u>Application</u>** property to return the **<u>Application</u>** object. The following statement returns the **<u>Application</u>** object.

```
myApplication = ActiveWeb.RootFolder.Files(0).MetaTags. _
Application
```

Use the **Count** property to return the number of META elements in the collection. The following statement returns the number of META elements in the tenth file of the **ActiveWeb** object.

```
myMetaTagCount = ActiveWeb.RootFolder.Files(9).MetaTags.Count
```

Use **Items**(*index*), where *index* is the **property key** value as a string, of an item in the **MetaTags** collection to return the **property key/value** pair. The following example returns the program identification tag from the META tags in the first file of the **ActiveWeb** object.

```
myMetaTagOne = ActiveWeb.RootFolder.Files(0).MetaTags("ProgId")
```

Common **property key** values are "generator" and "progid." For more information about **property key** values, see the table in the **<u>Properties</u>** object.

Use the **Parent** property when you want to return the file container for the **MetaTags** collection. For example, the following example returns the **Url** property of the **WebFile** container object that is associated with the META tags for the first file of the **ActiveWeb** object.

```
myMetaTagParent = _
ActiveWeb.RootFolder.Files(0).MetaTags.Parent.Url
```

NavigationNodes Collection

Multiple objects ^L<u>NavigationNodes</u> ^L<u>NavigationNode</u> ^LMultiple objects

A collection of **NavigationNode** objects within the navigational structure of a Microsoft FrontPage-based Web site. Each **NavigationNode** object represents a pointer to a page on a Web site. The **NavigationNode** object is a member of the **NavigationNodes** collection.

Using the NavigationNodes object

Use the **NavigationNode** property to return the **NavigationNode** object for a **WebFile** object. For more information on returning the collection of child nodes within the navigational structure of a Web site, see the <u>Children</u> property. The following example builds a list of the labels that are associated with each **NavigationNode** object in the **NavigationNodes** collection. The procedure exits when it reaches the end of the navigational structure.

```
Private Sub GetNavigationNode()
    Dim myWeb As WebEx
    Dim myWebFiles As WebFiles
    Dim myWebFile As WebFile
    Dim myNavNodeLabel As String
    Dim myLabel As String
    On Error Resume Next
    Set myWeb = ActiveWeb
    Set myFiles = myWeb.RootFolder.Files
    With myFiles
        For Each myFile In myFiles
            myLabel = myFile.NavigationNode.Label
            If Err <> 0 Then Exit Sub
            myNavNodeLabel = myNavNodeLabel & myLabel & vbCRLF
        Next
    End With
End Sub
```

Use **Children**(*index*), where *index* is the index number of an item in the collection of child nodes, to return a single **NavigationNode** object. The following example returns the first **NavigationNode** object in the collection — which is the home page.

```
Set myNavNodeOne = ActiveWeb.RootNavigationNode.Children(0)
```

Use the **Add** method to add a **NavigationNode** object to the **NavigationNodes** collection. The following example adds a node to the rightmost position in the current navigational structure.

Note After you finish modifying your navigational structure, you must apply the changes using the **ApplyNavigationStructure** method before the navigational structure is updated in FrontPage.

PageWindows Collection

Multiple objects PageWindows
PageWindow
Multiple objects

A collection of **PageWindowEx** objects. Each **PageWindowEx** object represents an open Web page in a Microsoft FrontPage application window. The **PageWindowEx** object is a member of the **PageWindows** collection.

Using the PageWindows object

Use the **PageWindows** property to return the **PageWindows** collection. The following statement returns the **PageWindows** object to the myPages variable.

```
myPages = WebWindows(0).PageWindows
```

Use **PageWindows**(*index*), where *index* is the index number of an item in the **PageWindows** collection, to return a single **PageWindowEx** object. The following statement returns the first **PageWindowEx** object in the collection.

```
Set myPageWindow = WebWindows(0).PageWindows(0)
```

Use the **Add** method to add a page window to the **PageWindows** collection. The following example opens the specified page in mywebOne and adds the page to the **PageWindows** collection.

```
Dim myPageWindows As PageWindows
Set myPageWindows = ActiveWeb.WebWindows(0).PageWindows
myPageWindows.Add("C:\My Web Sites\myWebOne\bugrep.htm")
```

Use the **Application** property to return the **Application** object from the **PageWindows** collection. The following example returns the **Application** object from the **PageWindows** collection.

```
myAppName = WebWindows.PageWindows.Application.Name
```

Use the **Close** method to close a **PageWindowEx** object or the collection of open **PageWindows**. The following statement closes the fourth **PageWindowEx** object for the first **WebWindowEx** object.

```
WebWindows(0).PageWindows(3).Close
```

Use **Close**(*index*), where *index* is the index number of an item in the **PageWindows** collection, to close a single **PageWindowEx** object as shown in the following statements. Both statements close the same page window. In the first statement, you close the page window using the index number for the **Close** method, while in the second statement, you close the page window by specifying the index number for the page you want to close.

```
WebWindows(0).PageWindows.Close(2)
WebWindows(0).PageWindows(2).Close
```

Use **Close()** to close all of the open **PageWindowEx** objects in the **PageWindows** collection. The following statement closes all of the open pages in the **PageWindows** collection.

```
WebWindows(0).PageWindows.Close()
```

Use the **Count** property to return the total number of page windows in the **PageWindows** collection. The following example returns the number of page windows in the collection of **PageWindows**.

```
myPageCount = WebWindows(0).PageWindows.Count
```

Use the **Parent** property when you want to return the container for the **PageWindows** collection. The following statement returns the **WebWindowEx** container object for the first **PageWindowEx** object using the **Parent** property.

```
Set myParent = ActiveWeb.WebWindows(0).PageWindows.Parent
```

Properties Collection

Multiple objects Properties

A collection of meta data as **Property** objects. Each item in the **Properties** collection represents an item associated with an individual **WebEx**, **WebFile**, or **WebFolder** object. For more information, see the table of **PropertyKeys** in <u>Using the PropertyKeys</u>. Choose a topic from the following list to go directly to that topic.

Using the Properties object

Using the Properties collection within a Web site

Using the property keys

Using the Properties object

Use the **Add** method to add a property to the **Properties** collection. The following statement adds a copyright statement to the **Properties** collection.

```
ActiveWeb.Properties.Add "Copyright", _
"Copyright 1999 by Coho Winery")
```

You can also add and remove categories and approval levels.

Use the **ApplyChanges** method to apply a **Property** object that has been added to the collection of **Properties**. The following statements add and apply a copyright property to the **Properties** collection.

```
ActiveWeb.Properties.Add "Copyright", _
"Copyright 1999 by Coho Winery")
ActiveWeb.Properties.ApplyChanges
```

Use the **Count** property to return the total number of properties in the **Properties** collection. The following example checks that the number of properties in the **Properties** collection has increased since the **Copyright** property was added and applied to the collection, and then goes on to add the copyright to a Web site page.

Note To run this example, you must have a Web site called "C:\My Documents\My Web Sites\Coho Winery", or you may substitute an alternative Web site URL.

```
Private Sub copyrightAdd()
   Dim myWeb As WebEx
   Dim myCopyright As String
   Dim myCount As Integer
   Dim myMessage As String
   myCopyright = "Copyright 1999 by Coho Winery"
   myCount = ActiveWeb.Properties.Count
   myMessage = "No new properties have been added."
   Set myWeb = Webs.Open("C:\My Web Sites\Coho Winery")
   myWeb.Activate
   ActiveWeb.Properties.Add "Copyright", myCopyright
```

```
If myCount <> ActiveWeb.Properties.Count - 1 Then
    MsgBox myMessage
    Exit Sub
End If
ActiveWeb.RootFolder.Files("Zinfandel.htm").Open
ActiveDocument.body.insertAdjacentText "BeforeEnd", _
    ActiveWeb.Properties("Copyright")
ActivePageWindow.Save
ActiveWeb.Close
End Sub
```

Use the **Delete** method to delete a single property. The following statement deletes the **Copyright** property from the **Properties** collection.

```
ActiveWeb.Properties.Delete("Copyright")
```

Use the **Parent** property when you want to return the container for the **Properties** collection. For example, the following statement returns the URL of the parent Web site.

```
myParentURL = ActiveWeb.Properties.Parent.Url
```

Using the Properties collection within a Web site

Use **Properties**(*index*), where *index* is the **PropertyKey** of the item in the **Properties** collection, to return a single property. The following example returns the value of the **PropertyKey** vti_hassearchbot.

```
Private Sub CheckForSearchBot()
   Dim myProperties As Properties
   Dim myFoundSearchBot As Boolean
   Set myProperties = ActiveWeb.Properties
   With myProperties
        myFoundSearchBot = .Item("vti_hassearchbot")
   End With
End Sub
```

Important The values for properties are variant— that is, each value can be a string, a Boolean, or an array. You can use the **Typename()** function to determine the type of the property value.

Note To run the following example you must create a form that contains a text box called txtCategories. Also, notice that myCategories is declared as a variant type in this example. In a previous example, myFoundSearchBot was declared Boolean— but could also have been declared as a string.

```
Private Sub GetWebPropertyCategories()
   Dim myProperties As Properties
   Dim myCategories As Variant
   Dim myCategory As Variant
   Dim txtCategories As String
   Set myProperties = ActiveWeb.Properties
   With myProperties
      myCategories = .Item("vti_categories")
      For Each myCategory In myCategories
        txtCategories = txtCategories & "|" & myCategory
        Next
   End With
End Sub
```

Using the property keys

The property keys can be a valuable tool for organizing information about your Web sites. For example, you can add categories to the master list of categories on your Web site.

Important The category names can be added to the categories list using case sensitivity, but will not display with case sensitivity in the User Interface (UI). Because of this, a situation may occur where a user creates a category on the client, such as "Web Administrators", and a duplicate category exists on the server, such as "web administrators". In this case, in the UI, Microsoft FrontPage will assign both categories to the "web administrators" category and neither of these can be deleted from the Web site (in the UI). Programmatically, the case sensitivity will correctly display in the Immediate and Local windows of the Visual Basic Editor.

The following example first adds a category to the properties for the active Web site, deletes the Waiting category from the same list (vti_categories), and then adds the new category to each of the files in the active Web site.

Note When creating a new category, you must first add the new category to the Web site before you add it to any of the files. The procedure, AddCategories, adds a new category to the Web site and the next procedure, AddCategoryToFiles, adds the same category to all of the files in the Web site.

```
Private Sub AddCategories()
   Dim myWeb As WebEx
   Dim myCategory(2) As String
   Dim myItem As Variant
   Set myWeb = ActiveWeb
   myCategory(0) = "+web administrators"
   myCategory(1) = "-waiting"
   ActiveWeb.Properties("vti_categories") = myCategory
   ActiveWeb.Properties.ApplyChanges
   'List all of the items in vti_categories in the Immediate window
   For Each myItem In myWeb.Properties("vti_categories")
        Debug.Print myItem
   Next
End Sub
```

```
Private Sub AddCategoryToFiles()
    Dim myWeb As WebEx
    Dim myCategories(1) As String
    Dim myFiles As WebFiles
    Dim myFile As WebFile
    Dim myItem As Variant
    Set myWeb = ActiveWeb
    Set myFiles = myWeb.RootFolder.Files
    myCategories(0) = "+web administrators"
    For Each myFile In myFiles
        myFile.Properties("vti_categories") = myCategories
        myFile.Properties.ApplyChanges
        'List all the items in vti_categories in the Immediate windo
        For Each myItem In myFile.Properties("vti_categories")
            Debug.Print myItem
        Next
    Next
End Sub
```

Note Although you can create new categories, such as "web administrators", that aren't included in the categories that are automatically installed with FrontPage, these new categories won't be visible in the User Interface (UI) until they are added to the Web site. Notice that in the previous example, the new category was added to the Web site first and will be visible in the UI.

The following table provides detailed information about the property keys available in FrontPage. Notice that all property keys in FrontPage start with "vti_". It is suggested that you prefix the property keys you want to add with a short name, such as an abbreviation for the name of your company, to reduce the chances of naming conflicts with other properties.

Property Key	Description	Permission
vti_approvallevel	The approval level set for a page. The available levels are AssignedTo, ReviewStatus, and Categories.	Read/write
	List of available approval levels that can be applied to documents in a Web site. This property key is a string vector (or an array of	

vti_approvallevels	strings). Used to populate the Web- wide list of known "review statuses" (also known as "approval levels") that are set for a document. The approval levels you can set are AssignedTo, ReviewStatus, and Categories.	Read-only
vti_approvedby	The authenticated name of the person who approved the page.	Read-only
vti_assignedby	The authenticated name of the person who assigned the page to a user.	Read-only
vti_assigneddate	This is the date that the page was last assigned to a user.	Read-only
vti_assignedto	The authenticated name of the person to whom the page is assigned.	Read-only
	Returns the author's name in string format. The identification of the authenticated author who has, or is, making changes to the document.	
vti_author	Note This is different from the Author property stored as part of Microsoft Office document properties, which can be set by the user to any arbitrary value.	Read-only
vti_casesensitiveurls	True if the Web site is running on a server where URLs are case sensitive, such as a server running UNIX. For example, if this property is set to True , then a file named MyFile.htm and a file named myFile.htm are treated as two separate files. The master list of categories. This property key is a string vector (or	Read-only

vti_categories	an array of strings) that is used for both Web-level and document-level meta data. At the Web-site level, the property key contains a list of all known categories that exist in the Web site. At the document level, it contains a list of all the categories to which the specified document belongs.	Read/write
	Note These categories are automatically updated whenever a category is added or modified in any way.	
vti_clientvercutoff	You can use this property key to set a version cutoff string. This is most commonly used in a multi-version environment to prevent older versions of the application from running on a specific server. For more information, see the <i>FrontPage Server Extensions</i> <i>Resource Kit.</i>	Read/write
vti_dataconns	List of data connections used by database features in FrontPage.	Read-only
vti_defaultcharset	The default character set for the Web site or page.	Read-only
vti_defaultlanguage	The default language for the Web site or page.	Read-only
vti_description	Provides a multi-line text box for the user to write comments in.	Read/write
vti_donotpublish	Set to True if the page is marked as "draft" or "not to be published."	Read/write
vti_filesize	The size of the file in bytes.	Read-only
	This property key corresponds to the generator tag within an HTML	

vti_generator	document. The value of the vti_generator property key contains the name of the application that "generated" the document, which may be different from the application that was invoked to handle the page. For example, if you created or edited a page with Microsoft Word, and then retrieved the value of the vti_generator property key in Microsoft FrontPage Visual Basic for Applications for that document, the value of the vti_generator property key would be Microsoft Word even though the procedure call was generated by FrontPage. Valid values are "Microsoft Word", "Microsoft Access", "Microsoft FrontPage", and so on. Note FrontPage defers to Word when there's a theme conflict if the vti_generator property key contains "Microsoft Word".	Read-only
vti_globalpage	Marks a page as global, that is, a top-level page in the navigation structure, on the same level as, but not equal to, the home page. True if the page contains framesets.	Read/write
vti_hasframeset	This property key is used when constructing link bar links. When the page that is being linked is a frameset, the link's target attribute is forced to "_top". This causes the new page to replace the current frameset to prevent recursive	Read-only
	framesets at browse time.	
-----------------------	--	------------
vti_hasruntimebots	True if the page has run-time bots on it.	Read-only
vti_htmlextensions	A concatenated string of extensions used to specify a Web file, such as htm.html.asp.	Read-only
vti_httpdversion	The version of FrontPage that is running on the httpd server.	Read-only
vti_imagemapformat	This property key sets the URL format used by the server-side image map processor. Valid values are NCSA and CERN. This property key is only useful for prior versions of FrontPage. FrontPage now uses client-side image maps.	Read/write
vti_imagemapurlprefix	Sets the server-relative URL of the server-side image-map processor for the selected image-map format. If the value of the vti_imagemapurlprefix property key is set to an empty string (""), server-side image maps are handled automatically by the FrontPage Server Extensions. For other formats, provide the name and	Read/write
vti insecureserverurl	location of the image-map processor. To specify client-side image maps, set this variable to an empty string. For more detailed information, see the <i>FrontPage</i> <i>Server Extensions Resource Kit</i> . Retrieves the http:// URL (unsecured URL) for a Web site. Provides compatibility with older browsers that may not have secure	Read-only
	browsing capability. For more information, see the <i>FrontPage</i>	J

	Server Extensions Resource Kit.	
vti_isbrowsable	True if the page folder is browsable. Use Folder.IsBrowsable to set this value	Read-only
vti_ischildweb	True if the specified folder is the root of a subsite. Use Folder.MakeWeb and Folder.RemoveWeb to modify.	Read-only
vti_isexecutable	Set this property to True to enable if this folder is marked for executable content on the server. Use Folder.IsExecutable to set this value.	Read/write
vti_isscriptable	Set this property to True if scripting is enabled for this folder. Use Folder.IsScriptable to set this value.	Read-write
vti_longfilenames	True if the server supports file names that are longer than the standard 8.3 file format. (The 8.3 file format only supports file names with eight characters and extensions with three characters, such as "filename.txt".)	Read-only
vti_metatags	This property key corresponds to the contents within the META tags in an HTML document. All of the information contained in this property key is added to the META tags collection.	Read-only
vti_modifiedby	Authenticated name of the author who last modified the page. The image file associated with the button that links to the home page.	Read-only
vti_navbuttonhomelabel	Note This image is retrieved from	Read/write

	the theme associated with the file and is not customizable.	
vti_navbuttonhomelabeltext	The text that overlays the vti_navbuttonhomelabel.	
	Note This text is customizable by the user.	Read/write
	The text that overlays the vti_nvabuttonuplabel.	
vti_navbuttonlabeltext	Note This text is customizable by the user.	Read/write
vti_navbuttonnextlabel	The image file associated with the button that links to the next page in the navigation structure.	
	Note This image is retrieved from the theme associated with the file and is not customizable.	Read/write
vti_navbuttonnextlabeltext	The text that overlays the vti_navbuttonnextlabel.	
	Note This text is customizable by the user.	Read/write
vti_navbuttonprevlabel	The image file associated with the button that links to the previous page in the navigation structure.	
	Note This image is retrieved from the theme associated with the file and is not customizable.	Read/write
	The text that overlays the vti_navbuttonprevlabel.	

vti_navbuttonprevlabeltext	Note This text is customizable by the user.	Read/write
vti_navbuttonuplabel	The image file associated with the button that links to the page one level up in the navigation structure. Note This image is retrieved from the theme associated with the file and is not customizable.	Read/write
vti_noclientimagemaps	True if client image maps are prohibited.	Read-only
vti_nonnavpage	True if the page isn't associated with a navigation node within the navigation structure.True if the Web site is not being stored under source control.Prevents adding a file to source control if this property is set to True.	Read/write
vti_nosourcecontrol	Note If a file has previously been added to a source control project with the vti_nosourcecontrol property set to False , and the property is subsequently set to True , the file will not be removed from the source control project.	Read/write
	This property key corresponds to the "originator" META tag in an HTML document. The value of this property key contains the name of the application that created the document. You should never overwrite this property key with the name of another application. The	

vti_originator	vti_generator property key should be used to track the application that last "generated" the HTML page. Simply editing a document in FrontPage will change the value of the vti_generator property key to "Microsoft FrontPage". Valid values are "Microsoft Word", "Microsoft Access", "Microsoft FrontPage", and so on.	Read-only
	Note FrontPage defers to Word when there's a theme conflict if the vti_generator property key contains "Microsoft Word".	
vti_privatetext	The value of this property key must be an integer. Zero or not present is the default. A non-zero value means that the page won't be added to any text indexes. This property key only works with the built-in text indexing in FrontPage and is not recognized by Microsoft Internet Information Services (IIS). This corresponds to the "progid" META tag in an HTML document. The value of this property key determines which application opens the document when a user clicks (or	Read/write
vti_progid	double-clicks) the document's file name in Windows Explorer. Valid values are "FrontPage.Editor.Document", "Word.Document", "Excel.Sheet", "PowerPoint.Slide", "Access.Application".	Read-only

Note FrontPage defers to Word

	when there's a theme conflict if the vti_generator property key contains "Microsoft Word".	
vti_scriptlanguage	Indicates which scripting language is enabled on the server.	Read-only
vti_secureserverurl	Retrieves the https://URL (secured URL) for a Web site. Provides a secure environment for confidential transactions over the Internet. For more information see the <i>FrontPage Server Extensions</i> <i>Resource Kit.</i>	Read-only
	The IP address of the server.	
vti_serveripaddress	Note An IP address beginning with 127 represents a local server.	Read-only
vti_serverlanguage	The language of the operating system. Used for localization purposes.	Read-only
vti_showhiddenpages	True if hidden pages are displayed.	Read-only
vti_sourcecontrolcheckedoutby	Authenticated name of the author who last checked out a page.	Read-only
vti_sourcecontrolerror	The error last returned by the source control system. This error is added to the META tag information if an error occurs while the file is being added to a source control project.	Read-only
vti_sourcecontrolproject	The name of the source control project. For a Microsoft Visual SourceSafe project, the string would include the "\$" symbol, as in "\$/myProject"	Read-only
	The value of this property key is the type of source control in use on that	

vti_sourcecontrolsystem	Web site. Can be Microsoft Visual SourceSafe or Microsoft Frontpage Light Weight source control. Valid values for these two types of source control are "VSS" and "LW", respectively. This property key is set when a source control project is started. For more information on source control projects, see <u>Managing Source Control Projects</u> .	Read-only
vti_sourcecontrolversion	A numbered string that contains the version of the source control system that is in use.	Read-only
vti_textextensions	A concatenated string of extensions that are commonly used to denote text files. For example, the concatenated string might contain ".txt.rpt"	Read-only
vti_themedefault	The default theme of the Web site.	Read-only
vti_themesusecss	True if CSS is used to present the theme.	Read/write
vti_timecreated	The timestamp when the page was created. The timestamp is in Universal Time Coordinate (UTC) or Greenwich time.	Read-only
vti_timelastmodified	The time that the page was last modified. The timestamp is in UTC time. This is only set when the author edits the page directly.	Read-only
vti_timelastwritten	The time that the page was last rewritten. The timestamp is in UTC time.	Read-only
vtititle	The value of the text between the <title> tags on the page.</title>	Read-only
vti_usernames	The list of known user names for a specified Web site.	Read-only
	The type of Web site server. Valid	

	values include, "MSIIS" for	
vti_webservertype	Microsoft Internet Information	Read-only
	Services and "PWS" for Personal	
	Web Server.	
vti_welcomenames	A concatenated list of file names that can be used as the file name for the home page, such as "index.htm index.html default.htm".	Read-only

Themes Collection

Multiple objects L<u>Themes</u>

A collection of **Theme** objects. Each **Theme** object represents a theme associated with an HTML page or a Web site in Microsoft FrontPage. The **Theme** object is a member of the **Themes** collection.

Using the Themes collection

Use the **Themes** property to return the **Themes** collection. The **Themes** property can be used with either the **Application** or the **WebEx** object. The **Themes** property for the **Application** object is the collection of themes available in FrontPage. The **Themes** property for the **WebEx** object is the collection of themes applied to the files within a Web site, or the collection of themes applied to a Web site. The following statement illustrates both the **Themes** and **Count** properties and returns the number of items in the **Themes** collection that are available in FrontPage.

myTotalThemeCount = Application.Themes.Count

The following statement returns the number of themes within the active Web site.

myWebThemeCount = ActiveWeb.Themes.Count

Use **Themes**(*index*), where *index* is the index number of a theme item, to return a single **Theme** object. The following statement returns the first theme in the **Themes** collection.

```
myThemeOne = Application.Themes(0)
```

Use the **Application** property to return the **Application** object. The following example returns the version and build number of FrontPage from within the **Themes** collection.

Note To run this procedure you must have at least one open Web site.

```
Private Sub GetBuildNumber()
   Dim myThemes As Themes
   Dim myBuild As String
   Set myThemes = ActiveWeb.Themes
   myBuild = myThemes.Application.Build
End Sub
```

Use the **Parent** property when you want to return the container for the **Themes** collection. For example, the following statement returns the URL associated with the parent container of the **Themes** collection.

myParentUrl = ActiveWeb.Themes.Parent.Url

WebFiles Collection

Multiple objects ^L<u>WebFiles</u> ^L<u>WebFile</u> ^LMultiple objects

A collection of **WebFile** objects. Each **WebFile** object represents all of the open files in a Web site. The **WebFile** object is a member of the **WebFiles** collection.

Using the WebFiles object

Use the **Files** property to return the **WebFiles** collection. The following statement returns the **WebFile** objects in the **WebFiles** collection.

Set myWebFiles = Application.Webs(0).RootFolder.Files

Use **WebFiles**(*index*), where *index* is the ordinal number of an item in the **WebFiles** collection, to return a single **WebFile** object. The following statement returns the first **WebFile** object in the collection.

Set myWebFile = ActiveWeb.RootFolder.Files(0)

Use the **Add** method to add a **WebFile** object to the **WebFiles** collection. The following statement adds a new **WebFile** object to the collection of **WebFiles**.

myWebFiles.Add ("C:\New Web WebFiles\Sales Statistics.htm")

Use the **Application** property to return the **Application** object from within the **WebFiles** collection. The following statement returns the **Application** object from the **WebFiles** collection.

myApp = ActiveWeb.RootFolder.Files.Application

Use the **Count** property to return the number of open files in the **WebFiles** collection. The following statement returns the number of files in the collection of **WebFiles**.

myCount = Webs(0).RootFolder.WebFiles.Count

Use the **Delete** method to delete a **WebFile** object. The following statement deletes the third file in the **WebFiles** collection.

```
Webs(0).RootFolder.Files(2).Delete
```

Use the **Parent** property when you want to return the container for the **WebFiles** collection. The following statement returns the URL of the folder for the second file in the active Web site.

myWebFileParent = ActiveWeb.RootFolder.Files(1).Parent.Url

WebFolders Collection

Multiple objects <u>WebFolder</u> Multiple objects

A collection of **WebFolder** objects. Each **WebFolder** object within the **WebFolders** collection represents a folder in a Web site. The **WebFolder** object is a member of the **WebFolders** collection.

Note All of the methods that involve changing the location of a folder, such as **Copy** or **Move**, only change the location within the current Web site; you cannot move a folder from one Web site to a another Web site.

Using the WebFolders object

Use **WebFolders**(*index*), where *index* is the index number of an item in the **WebFolders** collection, to return a single **WebFolder** object. The following example returns the first **WebFolder** object in the collection.

Set myFolder = ActiveWeb.RootFolder.Folders(0)

Use the **Add** method to add a new **WebFolder** object to the **WebFolders** collection in a Web site. Both of the following statements add a **WebFolder** to the collection of **WebFolders** in the active Web site— parentheses are not required for the folder name, as shown in the second statement.

Note The *FolderUrl* argument within the quotes ("Coho Winery") should only include the new folder name, not the entire URL, unless you are adding a new URL to the location designated as the *FolderUrl*. The program will fail if the entire URL is included for existing URLs.

```
ActiveWeb.RootFolder.Folders.Add ("Coho Winery")
ActiveWeb.RootFolder.Folders.Add "Coho Winery"
```

Use the **Count** property to return the number of total navigation nodes in the **WebFolders** collection. The following statement returns the number of Web folders in the Coho Winery Web site.

```
Webs("C:\Web Server One\Coho Winery").Folders.Count
```

Use the **Delete** method to delete a folder from a Web site. The following statements delete the tenth **WebFolder** object. The second statement uses the name of the folder instead of the index number to designate the folder to delete.

```
ActiveWeb.RootFolder.Folders(9).Delete
ActiveWeb.RootFolder.Folders("TempFolder").Delete
```

Use the **Copy** method to copy a **WebFolder** object. The following example copies a folder (WebFolders(4)) to another folder on the active Web site

(Chardonnay Inventory). For purposes of this example, WebFolders(4) is a folder named Inventory in the Coho Winery Web site. This folder contains the entire wine inventory— but the Web designer wanted to feature the sale on Chardonnay wines and created a temporary folder that will be edited to contain only Chardonnay wine.

```
Private Sub CopyInventory()
   Dim myFolder As WebFolder
   Set myFolder = ActiveWeb.RootFolder.Folders(4)
   myFolder.Copy ("C:\Coho Winery\Chardonnay Inventory, False, True
End Sub
```

Use the **Parent** property when you want to return the container for the **WebFolders** collection. The following statement returns the container for the fourth folder.

```
myParent = Webs.RootFolder.Folders(3).Parent
```

Webs Collection

<u>Application</u> <u>Webs</u> <u>Web</u> Multiple objects

A collection of <u>WebEx</u> objects. Each **WebEx** object represents a Web site, which can either be disk-based (on a local hard drive) or server-based (on a Web server). The **WebEx** object is a member of the **Webs** collection.

Using the Webs collection

Use the **Webs** property to return the **Webs** collection. You can also use the **Application** property to return the **Application** object. The following statement uses the **Application** object to return the first item in the **Webs** collection or use the second statement to return the entire collection of Web sites.

```
Set myWebOne = Application.Webs(0)
Set myWebs = Application.Webs
```

Use **Webs**(*index*), where *index* is the index number of an item in the **Webs** collection, to return a single **WebEx** object. The following statement returns the third Web site in the collection of open **WebEx** objects.

Set myGetWebThree = Webs(2)

Use the **Add** method to add an item to the list of available items in the **Webs** collection. The following statement adds the Coho Winery Web site to the **Webs** collection. If it doesn't exist, FrontPage will create a new Web site at the specified path and open it.

Webs.Add("C:\My Documents\My Web Sites\Coho Winery")

Use the **Application** property to return information about the application from within the **Webs** collection. The following statement returns the version number of the **Application** object.

ActiveWeb.Webs.Application.Version

Use the **Count** property to return the number of open Web sites in the **Webs** collection. The following statement returns the number of open Web sites.

Webs.Count

Use the **Delete** method to permanently delete a Web site from the **Webs** collection. The following statement deletes the Coho Winery Web site.

Webs.Delete("C:\My Documents\My Web Sites\Coho Winery")

Use the **Open** method to open a Web site. The following statement opens a Web site and adds it to the collection of items in the **Webs** collection.

Webs.Open("C:\My Documents\My Web Sites\Coho Winery")

Use the **Parent** method when you want to return the container of the **Webs** collection, which is the application. The following statement returns the **Application** object.

Set myParent = Webs.Parent

WebWindows Collection

Multiple objects L<u>WebWindows</u> <u>WebWindow</u> Multiple objects

A collection of <u>WebWindowEx</u> objects. Each **WebWindow** object represents an open Web site in Microsoft FrontPage. The **WebWindowEx** object is a member of the **WebWindows** collection.

Using the WebWindows Collection

Use **Items**(*index*), where *index* is the index number of an item in the **WebWindows** collection, to return a single **WebWindowEx** object. The following example returns the **Caption** property for the fourth item in the **WebWindows** collection.

```
Function ReturnWebWindowCaption() As String
   Dim myCaption As String
   Dim myWebWindow As WebWindowEx
   Set myWebWindow = Application.WebWindows(3)
   myCaption = myWebWindow.Caption
   ReturnWebWindowCaption = myCaption
End Sub
```

Use the **WebWindows** property to return the **WebWindows** collection. The following example closes all of the open **WebWindowEx** objects in the **WebWindows** collection except the **ActiveWebWindow** object.

```
Private Sub CloseWebWindows
   Dim myWebWindows As WebWindowEx
   Dim myWebWindow As WebWindowEx
   Dim myActiveWebWindow As WebWindowEx
   Set myWebWindows = Application.WebWindows
   Set myActiveWebWindow = ActiveWebWindow
   For Each myWebWindow In myWebWindows
        If myWebWindow.Caption <> myActiveWebWindow.Caption Then _
            myWebWindow.Close
   Next
End Sub
```

Use the **Application** property to return the **Application** object. If you're already working with the **WebWindows** collection and you'd like to check the version number of the application, you can easily access it through your With myWebWindows statement as shown in the following example.

```
With myWebWindows
myWebWindowCount = myWebWindows.Count
```

```
myAppVersion = .Application.Version
If myAppVersion < "4.0" Then
    MsgBox "Please upgrade your FrontPage software."
Else
    For Each myWebWindow In myWebWindows
        myCaption = myWebWindow.Caption
        With myPageWindows
            myPageCount = PageWindows.Count
        End With
        Next
End If
End With</pre>
```

You can use the **Close** method to close an individual **WebWindowEx** object, multiple **WebWindowEx** objects, or all **WebWindowEx** objects in FrontPage. For more details on the **Close** method, see the usage described in the following table.

Important FrontPage will close the application if you use the **Close** method to close all **WebWindowEx** objects in FrontPage.

Close Method Usage	Code
Close an individual WebWindowEx object in the application	Application.WebWindows(<i>index</i>).Close
	or Application.WebWindows.Close(<i>index</i>)
Close multiple WebWindowEx objects in the application (as shown in the previous example)	<pre>For Each myWebWindow In myWebWindows If myWebWindow.Caption <> _ myActiveWebWindow.Caption Then _ myWebWindow.Close Next</pre>
Close <i>all</i> WebWindowEx objects in the application	Application.WebWindows.Close
Close an individual	Webs(<i>index</i>).WebWindows(<i>index</i>).Close

WebWindowEx object in a Web site

	<pre>Set myWeb = Web(index).WebWindows</pre>
Close multiple WebWindowEx objects in a Web site	For Each myWebWindow In myWebWindows If myWebWindow.Caption _ <> myActiveWebWindow.Caption Then _ myWebWindow.Close
	Next

Close a collection in a Web Webs(*index*).WebWindows.Close site

Closing all **WebWindowEx** The expression, objects in FrontPage functions Application.WebWindows.Close is the same as, the same as the **Quit** method Application.Quit.

Use the **Count** property to return the number of **WebWindowEx** objects in the collection. The following example returns the number of **WebWindowEx** objects.

Web.WebWindows.Count

Use the **Parent** property when you want to return the container for the **WebWindows** collection. The following statement returns the **Application** object.

Application.WebWindows.Parent.Name

Application Object

Application ^LMultiple objects

Represents the Microsoft FrontPage application. The **Application** object includes properties and methods that return top-level objects. For example, the **ActiveDocument** property returns a document object that references the FrontPage Page object model that is compatible with Microsoft Internet Explorer 4.0 and later.

Using the Application Object

Use the **Application** property to return the **Application** object. You can use the **Application** property from any of the objects in FrontPage. The following example accesses the **Application** object, and then displays the **Open** dialog box.

Application.FileDialog(msoFileDialogOpen).Show

Many of the properties and methods that return the most common user-interface objects, such as the **ActiveDocument** property, can be used without the **Application** object qualifier. For example, instead of writing Application.ActiveDocument.Title, you can write ActiveDocument.Title. Properties and methods that can be used without the **Application** object qualifier are considered "global." To view global properties and methods in the Object Browser, click <globals> at the top of the list in the **Classes** box of the Object Browser.

Remarks

To use Automation to control FrontPage from another application, use the **CreateObject** or **GetObject** function to return a FrontPage **Application** object. The following Microsoft Word Visual Basic for Applications (VBA) example starts FrontPage, opens an existing Web site, and closes the Web site.

```
Private Sub StartFrontPage()
    Dim myNewFP As Variant
    Set myNewFP = CreateObject("FrontPage.Application")
    myNewFP.Webs.Open ("C:\MyWebs\Adventure Works")
    myNewFP.Webs.Close "(C:\MyWebs\Adventure Works")
    Set myNewFP = Nothing
End Sub
```

BasicList Object

BasicList ^LMultiple objects

Contains information about the basic list type used within Microsoft FrontPage. The **BasicList** object allows users to share and categorize information between Web sites.

This object is supported only by Web pages or sites that are based on Microsoft SharePoint Services.

Using the BasicList object

Use **Lists.item**(*index*), where *index* is either the name of the basic list or its numeric position within the collection, to return a single **Basic List** object. The following example displays the names of all basic lists in the active Web site. If the Web site contains no lists, a message is displayed to the user.

```
Sub ListAllLists()
'Displays the names of all basic lists in the collection
    Dim lstWebList As List
    Dim strName As String
    Dim blnFound As Boolean
    'Set found flag to false
    blnFound = False
    'Check if any lists exist and, if so cycle through them
    If Not ActiveWeb.Lists Is Nothing Then
        For Each lstWebList In ActiveWeb.Lists
            If lstWebList.Type = fpListTypeBasicList Then
                'Set boolean flag to found and names to string
                blnFound = True
                If strName = "" Then
                    strName = lstWebList.Name & vbCr
                Else
                    strName = strName & lstWebList.Name & vbCr
                End If
            End If
        Next
        If blnFound = True Then
            'Display names of all basic lists
            MsgBox "The names of all basic lists in the current Web
                   & vbCr & strName
        Else
            MsgBox "There are no basic lists in the current Web."
        End If
    Else
        'Otherwise display message to user
        MsgBox "The current Web contains no lists."
    End If
End Sub
```

Use the **Lists** collection's **Add** method to create a new list of type **fpListTypeBasicList**. The following example creates a new list called NewShare.

```
Sub NewList()
'Adds a new list to the current Web site
Dim objApp As FrontPage.Application
Dim objLists As Lists
Set objApp = FrontPage.Application
Set objLists = objApp.ActiveWeb.Lists
'Add new list and displays a message to the user.
objLists.Add Name:="NewShare", _
ListType:=fpListTypeBasicList, _
Description:="List of Shared files"
MsgBox "A new list was added to the Lists collection."
```

End Sub

Discussion Object

Discussion ^LMultiple objects

Contains information about a Discussion list in a Microsoft FrontPage Web site.

This object is supported only by Web pages or sites that are based on Microsoft Windows SharePoint Services.

Using the Discussion object

Use the **Item** property for the **Lists** collection to return a single **Discussion** object. The following example assumes the first list in the active Web site is a **Discussion** object, and then returns it.

Dim objDiscussion As Discussion

```
Set objDiscussion = ActiveWeb.Lists(0)
```

DocumentLibrary Object

DocumentLibrary ^LMultiple objects

Represents the collection of documents in the current Web site.

This object is supported only by Web pages or sites that are based on Microsoft SharePoint Services.

Using the DocumentLibrary object

Use **Lists.Item**(*index*), where *index* is either the name of the document library or its numeric position within the collection, to return a single **DocumentLibrary** object. The following example displays the names of all document libraries in the active Web site. If the Web site contains no document libraries, a message is displayed to the user.

```
Sub ListAllLibraries()
'Displays the names of all document libraries
'in the collection.
    Dim lstWebList As List
    Dim strName As String
    Dim blnFound As Boolean
    'Set found flag to false
    blnFound = False
    'Check if any lists exist and is so, cycle through them
    If Not ActiveWeb.Lists Is Nothing Then
        For Each lstWebList In ActiveWeb.Lists
            If lstWebList.Type = fpListTypeDocumentLibrary Then
                'Set boolean flag to found and add name to string
                blnFound = True
                If strName = "" Then
                    strName = lstWebList.Name & vbCr
                Else
                    strName = strName & lstWebList.Name & vbCr
                End If
            End If
        Next
        If blnFound = True Then
            'Display names of all document libraries
            MsqBox "The names of all document libraries in the curre
                   & vbCr & strName
        Else
            MsqBox "There are no document libraries in the current w
        End If
    Else
        'Otherwise display message to user
        MsqBox "The current web contains no lists."
    End If
End Sub
```

Use the **List** object's **Add** method to create a new list of type **fpListTypeDocumentLibrary**. The following example creates a new document library called Newlibrary.

```
Sub NewLibrary()
'Adds a new list to the current web site
Dim objApp As FrontPage.Application
Dim objLists As Lists
Set objApp = FrontPage.Application
Set objLists = objApp.ActiveWeb.Lists
'Add new list
objLists.Add Name:="NewLibrary", _______ListType:=fpListTypeDocumentLibrary, _______Description:="List of Shared files"
'Display message to user
MsgBox "A new library was added to the Lists collection."
```

End Sub

List Object

Multiple objects Lists

Contains information about the **List** object and the Microsoft FrontPage collaboration objects. The **List** object is a base class that defines the common members used by the different types of lists in FrontPage. For example, the collaboration objects,— the **BasicList**, **Discussion**, **DocumentLibrary**, and **Survey** objects— allow information to be shared and exchanged between different users and different web sites. The **List** object is a member of the **Lists** collection.

This object is supported only by Web pages or sites that are based on Microsoft SharePoint Services.
Using the List object

Use **Lists.Item**(*index*), where *index* is either the name of the list or its numeric position within the collection, to return a single **List** object. The following example displays the names of all lists in the active Web site. If the Web site contains no lists, a message is displayed to the user.

```
Sub ListAllLists()
'Displays the names of all lists in the collection
    Dim lstWebList As List
    Dim strName As String
    'Check if any lists exist
    If Not ActiveWeb.Lists Is Nothing Then
        'Cycle through lists
        For Each lstWebList In ActiveWeb.Lists
            'add list names to string
            If strName = "" Then
                strName = lstWebList.Name & vbCr
            Else
                strName = strName & lstWebList.Name & vbCr
            End If
        Next
        'Display names of all lists
        MsgBox "The names of all lists in the current web are:" _
               & vbCr & strName
    Else
        'Otherwise display message to user
        MsqBox "The current web contains no lists."
    End If
```

```
End Sub
```

Similarly, use the <u>WebFolder</u> object's <u>List</u> property to return the List object associated with the folder.

Use the **List** object's **<u>Fields</u>** property to return a collection of <u>**ListField**</u> objects that define the fields in the current list.

ListField Object

Multiple objects ListFields

Contains information about the fields that make up a **List** object. The fields of a list define the columns that appear in the list and present information about the items in the list. The **ListField** object is a base class that defines the common members used by the different types of fields in Microsoft FrontPage lists.

Using the ListField object

Use **ListFields.Item**(*index*), where *index* is the either name of the field or it's position within the collection to return a single **ListField** object. The following example displays the names of all fields in the current list. If the Web site contains no lists, a message is displayed to the user.

```
Sub ListAllFields()
'Displays the name of fields in the current list
    Dim objApp As FrontPage.Application
    Dim objField As ListField
    Dim strType As String
    Set objApp = FrontPage.Application
    If Not ActiveWeb.Lists Is Nothing Then
        For Each objField In objApp.ActiveWeb.Lists.Item(0).Fields
            If strType = "" Then
                'Create new string
                strType = objField.Name & vbCr
            Else
                'Add next field name to string
                strType = strType & objField.Name & vbCr
            End If
        Next objField
        MsgBox "The names of the fields in this list are: " & _
                vbCr & strType
    Else
        'Otherwise display message to user
        MsgBox "The current web site contains no lists."
    End If
End Sub
```

ListFieldAttachments Object

ListFieldAttachments ListFieldAttachments

Contains information about the attachments field when attachments are enabled for a list in a Microsoft FrontPage Web site.

Using the ListFieldAttachments object

Use the **Item** property for the **ListFields** collection to return a single **ListFieldAttachments** object. The following example returns the first field in the the first list in the **Lists** collection.

Dim objList As BasicList Dim objField As ListFieldAttachments

```
Set objList = ActiveWeb.Lists(0)
Set objField = objList.Fields(0)
```

ListFieldChoice Object

<u>ListFieldChoice</u>

L<u>Web</u>

Contains information about the choice field. The choice field allows the user to select from a specified number of options by providing a drop-down list or radio buttons in the list.

Using the ListFieldChoice object

Use **ListFields.**<u>Item</u> (*index*), where *index* is the name or ordinal position of a field of type **fpFieldChoice** to return a single **ListFieldChoice** object.

Use the **ListFields.Add** method to add a field of type **fpFieldChoice** to the **ListFields** collection. The following example adds a new field named NewChoiceField of type **fpFieldChoice**, to the **ListFields** collection. Use the **AddChoice** method to add choices to the list. Use the **clear** method to clear the choices from the list.

```
Sub DisplayChoice()
'Display choice
Dim objApp As FrontPage.Application
Dim objLstFlds As listFields
Dim objFldChoice As ListFieldChoice
Dim strName As String
Set objApp = FrontPage.Application
Set objLstFlds = objApp.ActiveWeb.Lists.Item(0).Fields
strName = "NewChoiceField"
'Add new field of type fpFieldChoice to list
objLstFlds.Add Name:=strName, Description:="New Choice value Fie
Fieldtype:=fpFieldChoice, Required:=True
MsgBox "A new Field named " & strName & " was added to the list
objApp.ActiveWeb.Lists.Item(0).Name & "."
```

End Sub

ListFieldComputed Object

ListFieldComputed ListFieldComputed

Contains information about fields created automatically by the computer. The **ListFieldComputed** object cannot be created by the user and instead is used by Microsoft FrontPage to create a reference from the list to a page in the Web site. For example, in many lists, the Title field is created by the computer and is used to reference the page corresponding to the list field.

Using the ListFieldComputed object

Use **ListFields.Item** (*index*) to return a **ListFieldComputed** object, where *index* is either the name of the field or its numeric position within the collection. The following example displays the names of all computed fields in the current list.

```
Sub ListComputedFields()
'Display the names of computed fields in the current list
    Dim objApp As FrontPage.Application
    Dim objField As ListField
    Dim strType As String
    Set objApp = FrontPage.Application
    If Not ActiveWeb.Lists Is Nothing Then
        For Each objField In objApp.ActiveWeb.Lists.Item(0).Fields
            'Check if it is a computed field
            If objField.Type = fpFieldComputed Then
                If strType = "" Then
                    'Create new string
                    strType = objField.Name & vbCr
                Else
                    'Add next field name to string
                    strType = strType & objField.Name & vbCr
                End If
            End If
        Next objField
        MsgBox "The names of the fields in this list are: " & _
                vbCr & strType
    Else
        'Otherwise display message to user
        MsgBox "The current web site contains no lists."
    End If
End Sub
```

ListFieldCounter Object

ListFieldCounter Web

Contains information about the key counter used within the list.

Remarks

This field is created automatically by Microsoft FrontPage and cannot be modified by the user.

Using the ListFieldCounter object

The following example displays the name associated with the counter field in the current list.

```
Sub ListCounterFields()
'Displays the name of counter fields in the current list
    Dim objApp As FrontPage.Application
    Dim objField As ListField
    Dim strType As String
    Dim blnFound As Boolean
    blnFound = False
    Set objApp = FrontPage.Application
    If Not ActiveWeb.Lists Is Nothing Then
        For Each objField In objApp.ActiveWeb.Lists.Item(0).Fields
            'Check if it is a computed field of type fpFieldFile
            If objField.Type = fpFieldCounter Then
                blnFound = True
                If strType = "" Then
                    'Create new string
                    strType = objField.Name & vbCr
                Else
                    'Add next field name to string
                    strType = strType & objField.Name & vbCr
                End If
            End If
        Next objField
        If blnFound = True Then
            MsgBox "The names of the fields in this list are: " & _
                    vbCr & strType
        Else
            MsgBox "There are no counter fields in the list."
        End If
    Else
        'Otherwise display message to user
        MsgBox "The current web site contains no lists."
    End If
End Sub
```

ListFieldCurrency Object

ListFieldCurrency

Web

Contains information about the field type used to view currency information within the list. The ListFieldCurrency object allows you to view information about different currency types within the currency field of the list.

Using the ListFieldCurrency object

Use **ListFields.Item** (*index*), where *index* is the name or ordinal position of a field of type **fpListFieldCurrency**, to return a single **ListFieldCurrency** object.

Use the **ListFields**.**Add** method to add a field of type **fpFieldCurrency** to the **ListFields** collection. The following example adds a new field named NewCurrencyField of type **fpFieldCurrency** to the **ListFields** collection and changes the currency type to display Canadian dollars.

```
Sub CreateCurrencyField()
'Add new Currency field
    Dim objApp As FrontPage.Application
    Dim objLstFlds As ListFields
    Dim objFldChoice As ListFieldCurrency
    Dim strName As String
    Set objApp = FrontPage.Application
    Set objLstFlds = objApp.ActiveWeb.Lists.Item(0).Fields
    strName = "NewCurrencyField"
    'Add new Field of type fpFieldCurrency to list
    objLstFlds.Add Name:=strName, Description:="New Currency Field",
                   Fieldtype:=fpFieldCurrency, Required:=True
    Set objFldChoice = objLstFlds.Item("NewCurrencyField")
    'Change currency type to Canadian
    objFldChoice.Currency = fpCurrencyFieldCanada
    MsgBox "A new Field named " & strName & " was added to the list
           objApp.ActiveWeb.Lists.Item(0).Name & "."
```

End Sub

ListFieldDateTime Object

ListFieldDateTime

└<u>Web</u>

Contains information about the field used to display dates and times within a Microsoft FrontPage list. The ListFieldDateTime object allows you to view date and time information in a variety of different configurations depending on the type of data in the list.

Using the ListFieldDateTime object

Use **ListFields.Item** (*index*), where *index* is the name or ordinal position of a field of type **fpListFieldDateTime**, to return a single **ListFieldDateTime** object.

Use the **ListFields.**Add method to add a field of type **fpListFieldDateTime** to the **ListFields** collection. The following example adds a new field named NewDateTimeField of type **fpListFieldDateTime** to the **ListFields** collection.

End Sub

ListFieldFile Object

ListFieldFile W



Contains information about any files contained in the list.

Remarks

This field is created automatically by Microsoft FrontPage and cannot be modified by the user.

Using the ListFieldFile object

The following example displays the names of all fields of type **fpFieldFile** in the active list. If no fields of this type exist, or if the Web contains no lists, a message is displayed to the user.

```
Sub ListFileFields()
'Displays the name of file fields in the current list
    Dim objApp As FrontPage.Application
    Dim objField As ListField
    Dim strType As String
    Dim blnFound As Boolean
    blnFound = False
    Set objApp = FrontPage.Application
    If Not ActiveWeb.Lists Is Nothing Then
        For Each objField In objApp.ActiveWeb.Lists.Item(0).Fields
            'Check if it is a computed field of type fpFieldFile
            If objField.Type = fpFieldFile Then
                blnFound = True
                If strType = "" Then
                    'Create new string
                    strType = objField.Name & vbCr
                Else
                    'Add next field name to string
                    strType = strType & objField.Name & vbCr
                End If
            End If
        Next objField
        If blnFound = True Then
            MsgBox "The names of the fields in this list are: " & _
                    vbCr & strType
        Else
            MsgBox "There are no file fields in the list."
        End If
    Else
        'Otherwise display message to user
        MsgBox "The current web contains no lists."
    End If
End Sub
```

ListFieldInteger Object

ListFieldInteger

^LWeb

Contains information about fields created automatically by the computer. The ListFieldinteger object cannot be created by the user and instead is used by Microsoft FrontPage to create an ID for each item in the list. For example, in a typical list, the ID field is created by the computer as a unique identifier for each item in the list.

Using the ListFieldInteger object

Use **ListFields.Item** (*index*) to return a **ListFieldInteger** object, where *index* is either the name of the field or its numeric position within the collection. The following example displays the names of all Integer fields in the current list. If the Web site contains no lists, or if the list contains no Integer fields, a message is displayed to the user.

```
Sub ListIntegerFields()
'Displays the name of Integer fields in the current list
    Dim objApp As FrontPage.Application
    Dim objField As ListField
    Dim strType As String
    Dim blnFound As Boolean
    blnFound = False
    Set objApp = FrontPage.Application
    If Not ActiveWeb.Lists Is Nothing Then
        For Each objField In objApp.ActiveWeb.Lists.Item(0).Fields
            'Check if it is a computed field of type fpFieldInteger
            If objField.Type = fpFieldInteger Then
                blnFound = True
                If strType = "" Then
                    'Create new string
                    strType = objField.Name & vbCr
                Else
                    'Add next field name to string
                    strType = strType & objField.Name & vbCr
                End If
            End If
        Next objField
        If blnFound = True Then
            MsqBox "The names of the fields in this list are: " &
                    vbCr & strType
        Else
            MsgBox "There are no Integer fields in the list."
        End If
    Else
        'Otherwise display message to user
        MsqBox "The current Web site contains no lists."
    End If
End Sub
```

ListFieldLookup Object

ListFieldLookup ^LMultiple objects

Contains information about the Lookup field. The **ListFieldLookup** object allows you to search for information within the given Web site based on a specified field.

Using the ListFieldLookup object

Use **ListFields.Item** (*index*), where *index* is the name or ordinal position of a field of type **fpFieldLookup**, to return a single **ListFieldLookup** object.

Use the **ListFields.**<u>Add</u> method to add a field of type **ListFieldLookup** to the **ListFields** collection.

The following example adds a new field named NewFileLookupField of type **fpFieldLookup** to the **ListFields** collection.

End Sub

ListFieldMultiLine Object

ListFieldMultiLine

└<u>We</u>b

Contains information about the field used to display information containing more than one line of text. For example, the **ListFieldMultiLine** object can be used to display descriptions and summaries, which often require more than a single line.

Using the ListFieldMultiLine object

Use **ListFields.Item** (*index*), where *index* is the name or ordinal position of a field of type **fpFieldMultiLine**, to return a single **ListFieldMultiLine** object.

Use the **ListFields.**<u>Add</u> method to add a field of type **fpFieldMultiLine** to the **ListFields** collection.

The following example adds a new field named Description of type **fpFieldMultiLine** to the **ListFields** collection. The subroutine displays the name of the new field as well as the number of lines it will contain.

```
Sub CreateMultiLine()
'Add new MultiLine Field
    Dim objApp As FrontPage.Application
    Dim objLstFlds As ListFields
    Dim objLstFldMulti As ListFieldMultiLine
    Dim strName As String
    Set objApp = FrontPage.Application
    Set objLstFlds = objApp.ActiveWeb.Lists.Item(0).Fields
    strName = "Description"
    'Add new Field of type fpFieldMultiLine to list
    objLstFlds.Add Name:=strName, Description:="Description Field",
                   Fieldtype:=fpFieldMultiLine
    Set objLstFldMulti = objLstFlds.Item(strName)
    MsgBox "A new field named " & strName & " was added to the list
           objApp.ActiveWeb.Lists.Item(0).Name & ". It contains " &
           obiLstFldMulti.NumberOfLines & " lines."
```

End Sub

ListFieldNumber Object

ListFieldNumber Web

Contains information about how numbers are displayed in Microsoft FrontPage list fields. The **ListFieldNumber** object allows you to configure the way in which you view numbers in FrontPage lists.

Using the ListFieldNumber object

Use **ListFields.Item** (*index*), where *index* is the name or ordinal position of a field of type **fpFieldNumber**, to return a single **ListFieldNumber** object.

Use the **ListFields**.**Add** method to add a field of type **fpFieldNumber** to the **ListFields** collection. The following example adds a new field named Total of type **fpFieldNumber** to the **ListFields** collection. The subroutine displays the name of the new field and the name of the list to which it was added.

Use the **ListFieldNumber** object's **DisplayFormat** property to change the way in which the data will be displayed in the field.

End Sub

ListFieldRatingScale Object

ListFieldRatingScale

^LWeb

Contains information about the rating scale list field for a list in a Microsoft FrontPage Web site. The **ListFeldRatingScale** object is only available for survey lists.

Using the ListFieldRatingScale Object

Use the **Item** property for the **ListFields** collection to return a single **ListFieldRatingScale** object. The following example returns the first field in the first list in the **Lists** collection. This example assumes that the first list in the active Web is a Web survey list.

Dim objList As BasicList Dim objField As ListFieldRatingScale

Set objList = ActiveWeb.Lists(0)
Set objField = objList.Fields(0)

ListFields Collection

Multiple objects ^L<u>ListFields</u>

Represents a collection of **ListField** objects that define the text fields used within Microsoft FrontPage lists. The **ListField** object is a base class that defines the common members used by the different types of fields in FrontPage. For example, the **ListFieldCurrency** and **ListFieldNumber** objects allow you to customize the way in which currency and numeric information is displayed.

Using the ListFields collection

Use **Fields** .**Item** (*index*), where *index* is either the name of the list or its position within the collection, to return a single **ListField** object. The following example displays the names of all fields in the first list of the active Web site. If the Web site contains no lists, a message is displayed to the user.

Use the **Add** method to add a new **ListField** object to the **ListFields** collection.

```
Sub ListFields()
'Display the names of fields in the current list
    Dim objApp As FrontPage.Application
    Dim objField As ListField
    Dim strType As String
    Set objApp = FrontPage.Application
    If Not ActiveWeb.Lists Is Nothing Then
        For Each objField In objApp.ActiveWeb.Lists.Item(0).Fields
            If strType = "" Then
                'Create new string
                strType = objField.Name & vbCr
            Else
                'Add next field name to string
                strType = strType & objField.Name & vbCr
            End If
        Next objField
        MsgBox "The names of the fields in this list are: " & _
                vbCr & strType
    Else
        'Otherwise display message to user
        MsgBox "The current web contains no lists."
    End If
End Sub
```
ListFieldSingleLine Object

ListFieldSingleLine

-<u>Web</u>

Contains information about the single line field. The **fpFieldSingleLine** object is used to display information that typically requires no more than a single line of text. For example, the Name and Title fields typically use the ListFieldSingleLine object to display information.

This object is supported only by Web pages or sites that are based on Microsoft SharePoint Services.

Using the ListFieldSingleLine object

The following example creates a new field of type **ListFieldSingleLine** named Alternative Name. The subroutine displays the name of the new field and the name of the list to which it was added.

End Sub

ListFieldTrueFalse Object

└<u>Web</u> ListFieldTrueFalse

Contains information about the **Boolean** field in a Microsoft FrontPage list. The ListFieldTrueFalse object allows you to provide the user with a simple, binary user-interface option.

This object is supported only by Web pages or sites that are based on Microsoft SharePoint Services.

Using the ListFieldTrueFalse object

The following example creates a new field of type **fpFieldTrueFalse** named Default Page. The subroutine displays the name of the new field and the name of the list to which it was added.

```
Sub CreateTrueFalseField()
'Add new TrueFalse field
Dim objApp As FrontPage.Application
Dim objLstFlds As ListFields
Dim strName As String
Set objApp = FrontPage.Application
Set objLstFlds = objApp.ActiveWeb.Lists.Item(0).Fields
strName = "Default Page"
'Add new Field of type fpFieldTrueFalse to list
objLstFlds.Add Name:=strName, Description:="Determines if this i
"default page", Fieldtype:=fpFieldTrueFalse
MsgBox "A new field named " & strName & " was added to the list
objApp.ActiveWeb.Lists.Item(0).Name & "."
```

End Sub

ListFieldURL Object

ListFieldURL

^LWeb

Contains information about the Microsoft FrontPage field used to display URLs. The ListFieldURL object allows you to customize the way URLs appear in the list fields. The URL can be displayed as an image or as a link. Use the **DisplayFormat** property to change the view settings.

This object is supported only by Web pages or sites that are based on Microsoft SharePoint Services.

Using the ListFieldURL object

Use **ListFields.Item** (*index*), where *index* is the name or ordinal position of a field of type **fpField URL**, to return a single **ListFieldURL** object.

The following example creates a new field of type **fpFieldURL** and displays the name of the new field and the name of the list into which it was inserted.

End Sub

NavigationNode Object

Multiple objects ^L<u>NavigationNode</u> ^LMultiple objects

Represents a node in the navigational structure of a Web site. The **NavigationNode** object is a member of the **NavigationNodes** collection. Within the **NavigationNodes** collection, individual **NavigationNode** objects are indexed beginning with zero.

Important From the **NavigationNode** object, you can access all other navigation nodes in a Web site. The **RootNavigationNode** object, created by default each time you create a new Web site, provides the basis for the navigation structure, which is accessed through the <u>Children</u> property. The first child node of the navigation structure is usually the home page of the Web site, which can be accessed through the <u>HomeNavigationNode</u> property. However, the first child node of the root navigation node can be any page, and may not contain a **HomeNavigationNode** object at all.

Using the NavigationNode object

You can use the **NavigationNode** property to return the **NavigationNode** object. The following example builds a list of navigation node labels for the **WebFile** object of the **WebFiles** collection.

```
Private Sub GetNavigationNode()
    Dim myWeb As WebEx
    Dim myWebFiles As WebFiles
    Dim myWebFile As WebFile
    Dim myNavNodeLabel As String
    Dim myLabel As String
    On Error Resume Next
    Set myWeb = ActiveWeb
    Set myFiles = myWeb.RootFolder.Files
    With myFiles
        For Each myFile In myFiles
            myLabel = myFile.NavigationNode.Label
            If Err <> 0 Then Exit Sub
            myNavNodeLabel = myNavNodeLabel & myLabel & vbCRLF
        Next
    End With
End Sub
```

The **Children** property returns the collection of child nodes within the navigation structure of a Web site. The following statement returns the number of child nodes within the navigation structure of the active Web site.

Use **Children**(*index*), where *index* is the index number of a navigation node item, to return a single **NavigationNode** object. The following statement returns the file name of the first navigation node in the **NavigationNodes** collection.

myNavNodeName _

= ActiveWeb.RootFolder.Files(0).NavigationNode.Children(0).File.

The **File** property returns the **File** object that is associated with the **NavigationNode** object. The following statement returns **True** if the file is open.

myNavFile = ActiveWeb.RootFolder.Files(3).NavigationNode.File.IsOpen

The **Home** property returns the **Home** object associated with the current navigation node and references information such as the **Children**, **File**, **Label**, **Next**, **Prev**, and other properties for the home page. The following statement returns the URL of the **Home** property for the **NavigationNode** object.

You can return the **Label** property to set or return text that can be used as buttons within the navigation structure, or used for text in a link bar. The following example returns the label for the home page.

myLabel = ActiveWeb.RootFolder.Files(0).NavigationNode.Label

Use the **Next**, **Parent**, **Prev**, or **Url** properties to return navigation nodes associated with the specified property. The following example returns the URL that is associated with the previous **NavigationNode** object.

myPrevNode = ActiveWeb.RootFolder.Files(1).NavigationNode.Prev.Url

Use the **Web** property to return the **WebEx** object associated with the current navigation node. The following example returns the **WebEx** object for the current navigation node.

```
myNavNodeWeb = __
ActiveWeb.RootFolder.Files(2).NavigationNode.Web.Url
```

Use the **Move** method to move a navigation node from one child node to another. The following example moves a navigation node to a child location on a sibling node in the same Web site.

Private Sub MoveNavNode()
Dim myNodes As NavigationNodes

Dim myNode As NavigationNode

Set myNodes = ActiveWeb.RootNavigationNode.Children
Set myNode = myNodes(4)
myNode.Move myNodes, myNodes(2)
ActiveWeb.ApplyNavigationStructure
End Sub

PageWindowEx Object

PageWindowEx ^LMultiple objects

Represents an open editor session and encapsulates the Microsoft FrontPage Page object model that is compatible with the Document object model for Microsoft Internet Explorer 4.0 and later. The **PageWindowEx** object is a member of the **PageWindows** collection and represents all of the open page windows in the specified Web site. Within the **PageWindows** collection, individual **PageWindowEx** objects are indexed beginning with zero.

Note You can also substitute the **Caption** property of the **PageWindow** object instead of the index number when accessing a **PageWindowEx** object. This only works when the **PageWindowEx** object has already been saved. The following statements return the **PageWindowEx** object for an open page window that has a caption of "C:\My Web Sites\Coho Winery\Zinfandel.htm").

Set objPageWindow = ActiveWebWindow.PageWindow("Zinfandel.htm")
Set objPageWindow = WebWindows(0).PageWindow("Zinfandel.htm")

Using the PageWindow object

Use **PageWindows**(*index*), where *index* is the index number of a page window item, to return a single **PageWindowEx** object. The following statement returns the file URL of the first page window item in the **PageWindows** collection.

PgePageOne = WebWindows(0).PageWindows(0).Document.Url

Use the **ActiveFrameWindow** property to return the active frame of a **PageWindowEx** object. The following statement returns the **ActiveFrameWindow** object.

Note The active frame is the frame that currently has the focus; it is shown in FrontPage with a blue border surrounding it.

```
Set objActiveFrame _
    = WebWindows(1).ActivePageWindow.ActiveFrameWindow
```

Use the **FrameWindow** property to return an **FPHTMLWindow2** object, which can then be used to return the **frames** collection and the content of each of the **frames** collection pages.

```
objFrameWindow = WebWindows(0).ActivePageWindow.FrameWindow
```

Use the **Document** property to return the document associated with the **PageWindowEx** object. The following statement returns the document associated with the first **PageWindowEx** object of the first **WebWindowEx** object.

```
Set objDoc = WebWindows(0).PageWindows(0).Document
```

You can use the **IsDirty** property to determine if the **PageWindowEx** object is dirty— that is, if it has been modified since the last refresh or save. The **isDirty** property returns **True** if the **PageWindowEx** object is dirty. The following example saves the **PageWindowEx** object if the first item in the **PageWindows** collection is dirty.

```
Private Sub CheckPageWindowIsDirty()
Dim objPageWin As PageWindowEx
```

```
Set objPageWin = WebWindows(0).PageWindows(0)
If objPageWin.IsDirty = True Then
        objPageWin.Save
End If
End Sub
```

You can use the **ViewMode** property to set the view for the page as shown in the following statement. For more information on the enumerated constants available for this property, see the table under the <u>ViewMode</u> property.

```
WebWindows(1).PageWindows(1).ViewMode = fpPageViewHtml
```

Survey Object

Survey ^LMultiple objects

Contains information about the Microsoft FrontPage **Survey** object that allows users to vote on issues and share information.

This object is supported only by Web pages or sites that are based on Microsoft SharePoint Services.

Using the Survey object

Use **Lists.Item**(*index*), where *index* is is the name or ordinal position of a **List** object of type **fpListTypeSurvey**, to return a single **Survey** object.

The following example lists the names of all surveys in the active Web site. If the Web site contains no surveys or the Web site contains no lists, a message is displayed to the user.

```
Sub ListAllSurveys()
'Displays the names of all survey objects in the collection
    Dim lstWebList As List
    Dim strName As String
    Dim blnFound As Boolean
    'Set found flag to false
    blnFound = False
    'Check if any lists exist
    If Not ActiveWeb.Lists Is Nothing Then
        'Cycle through lists
        For Each lstWebList In ActiveWeb.Lists
            If lstWebList.Type = fpListTypeSurvey Then
                'Set boolen flag to found
                blnFound = True
                'add list names to string
                If strName = "" Then
                    strName = lstWebList.Name & vbCr
                Else
                    strName = strName & lstWebList.Name & vbCr
                End If
            End If
        Next
        If blnFound = True Then
            'Display names of all survey objects
            MsgBox "The names of all survey objects in the current w
                   & vbCr & strName
        Else
            MsqBox "There are no survey objects in the current Web s
        End If
    Else
        'Otherwise display message to user
        MsgBox "The current Web site contains no lists."
    End If
```

End Sub

Use the **Lists** collection's **Add** method to create a new list of type **fpListTypeSurvey**. The following example creates a new survey called NewSurvey.

End Sub

System Object

Application LSystem

Provides access to system information such as the operating system, screen resolution, or registry.

Using the System Object

Use the **System** property to return the **System** object. The following statement returns the name of the calling application.

```
mySysApp = System.Application.Name
```

Use the **Build** and **Version** properties to return build and version information about the operating system.

myVer = System.Version
myBld = System.Build

The horizontal and vertical resolution can be used to determine if a graphic can be displayed on a client's machine. The following statements return the resolution settings.

currHorizRes = System.HorizontalResolution
currVertRes = System.Vertical.Resolution

Use the **OperatingSystem** property to return the name of the current operating system, as shown in the following statement.

thisOps = System.OperatingSystem

Use the **Parent** property to return the parent of the specified object. The following statement returns parent information for the **System** object.

```
Private Sub GetSystemParentInfo()
   Dim mySys As System
   Dim mySysUserName As String
   Set mySys = System
   With mySys
        mySysUserName = .Parent.UserName
   End With
End Sub
```

Use the **LanguageDesignation** property to return a three-letter abbreviation for the language used for the operating system. The following statement returns "enu" as the language designation abbreviation for the English (US) language.

currSystemLanguage = System.LanguageDesignation

Use the **ProfileString** property to return or set an entry in the Windows registry. If used without parameters, the **ProfileString** property defaults to the following key:

HKEY_CURRENT_USERS\Software\Microsoft\FrontPage

The parameters for the **ProfileString** property are:

- *RegistrySection*, where *RegistrySection* is a registry subtree such as HKEY_CURRENT_USER or HKEY_LOCAL_MACHINE.
- *RegistryKey*, where *RegistryKey* is the next level below the section or subtree with such key names as Software or Network.

The following example returns the Identifier for the CentralProcessor subkey.

myRegKeyInfo = mySys.ProfileString(myRegSec, "Identifier")
End Sub

Theme Object

Multiple objects L<u>Themes</u>

Represents a theme in Microsoft FrontPage. The **Theme** object is a member of the **Themes** collection. The **Themes** collection represents all the themes on the local machine or all themes applied to a specific Web site. Within the **Themes** collection, individual **Theme** objects are indexed beginning with zero.

Using the Theme object

Use the **Format** property to return format information for a theme. The following statement returns the format for the first theme applied to the active Web site.

```
myThemeFormat = ActiveWeb.Themes(1).Format
```

Use the **Label** property to return the name that is displayed in the Theme list box. This is the full name of the theme. The following statement returns the label for the theme.

```
myThemeLbl = ActiveWeb.Themes(1).Label
```

Use the **Name** property to return or apply a theme to a Web page. The **Name** property contains the directory name for the theme, which is an abbreviated version of the name of the theme as displayed in the Theme list box. The following example lists the names of all themes in the themes collection in the body of the active document.

Note To run this example, you must have a Web site open that has a theme applied to the active page, the entire Web site, or a specific file in the Web site.

```
Sub ListThemes()
Dim objTheme As Theme
For Each objTheme In Application.Themes
        ActiveDocument.body.insertAdjacentText "beforeend", objTheme
        Next
```

End Sub

Use the **Version** property to return the version number of the theme. The following example returns the version for a theme.

```
myThemeVersion = Theme.Version
```

WebEx Object

Multiple objects L<u>Web</u> Multiple objects

Represents a Microsoft FrontPage Web site. The **WebEx** object is a member of the <u>Webs</u> collection, which represents all of the open Web sites in FrontPage. FrontPage provides the ability to create multiple **WebEx** objects on a Web server. Within the **Webs** collection, individual **WebEx** objects are indexed beginning with zero. The directory hierarchy of a Web site in FrontPage is similar to a folder hierarchy. Any **WebFolder** *can* represent a Web site, but every **WebFolder** does not necessarily represent a Web site. The Web folder hierarchy provides the link to folders and files on a Web server directory.

Using the Web object properties

Use the **Web** property to return the **WebEx** object. The following example checks the Web site's operating system for the capability of processing long file names.

Note To run this example, create a form with a command button called cmdCheckLongFilenames, a text box called txtLongFilenames, and copy the example into the code window.

```
Private Sub cmdCheckLongFilenames()
Dim objPageWin As PageWindow
Set objPageWin = ActivePageWindow
With objPageWin
If .Web.AllowsLongFilenames = True Then
txtlongFilenames = _______This operating system uses long file names."
Exit Sub
Else
txtlongFilenames = ______This operating system only uses short file names."
End If
End With
End Sub
```

Use **Webs**(*index*), where *index* is the ordinal position of a Web site in the **Webs** collection, to return a single **WebEx** object. The following example returns the URL of the first Web site in the **Webs** collection.

Application.Webs(0).Url

Use the **ActiveWebWindow** property to return the selected **WebWindowEx** object. From the **WebWindowEx** object, you can access the **ActiveDocument**, **ActivePageWindow**, or **Application** properties, along with properties such as **Caption**, **PageWindows**, **Parent**, **ViewMode**, **Visible**, and **Web**. The following example returns the creation date and file size of the active document.

Note Although **Date** is an available type in Microsoft Visual Basic for Applications (VBA), the **WebWindowEx** object returns the date in string format and does not automatically convert the string to a date format.

```
Private Sub ActiveDocDateSize()
   Dim objWebWindow As WebWindowEx
   Dim strFileSize As String
   Dim strCreateDate As String
   Set objWebWindow = ActiveWebWindow
   With objWebWindow
      strFileSize = .ActiveDocument.fileSize
      strCreateDate = .ActiveDocument.fileCreatedDate
   End With
End Sub
```

The **RevisionControlProject** and **IsUnderRevisionControl** properties return the status of the **WebEx** object's revision state. You can control versioning in Microsoft FrontPage through Microsoft Visual SourceSafe or through Microsoft Office-style locking. For more information on source control projects and Office-style locking, see <u>Managing Source Control</u>.

If a revision control project does not correspond to a valid Visual SourceSafe project, FrontPage defaults to Office-style locking. The following example returns the **RevisionControlProject** and **IsUnderRevisionControl** properties, and includes a source control project example.

Note To run this example, create a module and copy the example into the code window. You must have a Web site open.

```
Private Sub SourceControl()
Dim objWeb As WebEx
Set objWeb = ActiveWeb
If Not(objWeb.IsUnderRevisionControl) Then
        objWeb.RevisionControlProject = "<FrontPage-based Locking>"
End If
End Sub
Private Sub ReturnRevisionState()
Dim objWeb As WebEx
Dim strRevCtrlProj As String
```

```
Dim blnIsUnderRevCtrl As Boolean
Set objWeb = ActiveWeb
With objWeb
    RevCtrlProj = .RevisionControlProject
    blnIsUnderRevCtrl = .IsUnderRevisionControl
End With
End Sub
```

Use the **RootFolder** and **RootNavigationNode** properties to determine the root folder or root navigation node. The **RootFolder** property returns a pointer to the root folder of a Web site. The **RootNavigationNode** property returns the **NavigationNode** object from which you can access all other navigation nodes in a Web site. The **RootNavigationNode** object is created by default when you create a Web site, and provides the basis for the navigation structure, which is accessed through the **Children** property. The first child node of the navigation structure is the home page of the Web site. The following example returns the name of the root folder and the URL of the **RootNavigationNode** object.

```
Private Sub GetRootInfo()
   Dim objWeb As WebEx
   Dim strRootFolder As String
   Dim strHomeNavNode As String
   Set objWeb = ActiveWeb
   With objWeb
      strRootFolder = .RootFolder.Name
      strHomeNavNode = .RootNavigationNode.Children(0).Url
   End With
End Sub
```

Use the **SharedBorders** property to set the shared borders for a Web site either on or off. The following statement sets the **SharedBorders** property to **True** and turns shared borders on for the specified Web site.

```
ActiveWeb.SharedBorders(fpBorderTop) = True
```

Use the **WebWindows** property to return the collection of **WebWindow** objects that are contained within the current **WebEx** object. The following statement returns a count of the **WebWindows** collection.

Application.WebWindows.Count

Using the Web object methods

Use the **Activate** method to place the focus on the current object. The following statements check if myAdventureWorksWeb is the active Web site; if it is not, then myAdventureWorksWeb is activated.

```
If ActiveWeb <> myAdventureWorksWeb Then
        objAdventureWorksWeb.Activate
End If
```

Use the **ApplyNavigationStructure** method to apply a newly created or modified navigation structure to a Web site. The following statement applies a navigation structure to a Web site, where the variable for the Adventure Works Web site is webAdventureWorksWeb.

```
myAdventureWorksWeb.ApplyNavigationStructure
```

Use the **CancelRequests** method to cancel all server requests. The following statement cancels all server requests for the Adventure Works Web site, with webAdventureWorksWeb as the **Web** object variable.

Note The client will stop all requests to the server; however, the server may have already started a transaction, in which case it will continue until the transaction is finished and then the remaining requests (if any) will be cancelled.

myAdventureWorksWeb.CancelRequests

Use the **LocateFile** or **LocateFolder** methods to return a **WebFile** or a **WebFolder** object within a Web site. The following example locates a folder for a disk-based Web site.

Application.Web.LocateFolder("C:\My Web Sites\Adventure Works\images

Use the **Publish** method to publish a Web site to a Web server. The following statement publishes the Adventure Works Web site to a Personal Web Server site.

```
Dim objWeb As WebEx
Set objWeb = Application.Web
With objWeb
   .Publish _
    "http://myServer/wwwroot", fpPublishAddToExistingWeb
```

The FpWebPublishFlags enumerated types can be concatenated as shown in the following statement.

```
myWeb.Publish _
    "http://myServer/wwwroot", fpPublishAddToExistingWeb + _
    fpPublishCopySubwebs
```

WebFile Object

Multiple objects L<u>WebFile</u> Multiple objects

Represents a file in a Microsoft FrontPage-based Web site. The **WebFile** object is a member of the **WebFiles** collection. The **WebFiles** collection represents all of the files in a specified **WebFolder** object. Within the **WebFiles** collection, individual **WebFile** objects are indexed beginning with zero. The **WebFile** object is similar to a file in a directory-based hierarchy. FrontPage provides the ability to create multiple **Web** objects on a Web server. Any **WebFolder** *can* represent a Web site, but every **WebFolder** does not necessarily represent a Web site.

Using the File object

Use **WebFiles**(*index*), where *index* is the ordinal number of a Web page, to return a single **WebFile** object. The following example returns the file name of the first Web page in the **WebFiles** collection.

ActiveWeb.RootFolder.Files(0).Name

Use the **File** object to return information about a file on a Web site. The following example returns the **Name**, **Title**, and **Url** properties of each **File** object on the active Web site.

Note To run this program, you must have a least one Web site open.

```
Private Sub GetWebFileInfo()
    Dim myWeb As WebEx
    Dim myFiles As WebFiles
    Dim myFile As WebFile
    Dim myFileName As String
    Dim myTitle As String
    Dim myUrl As String
    Set myWeb = ActiveWeb
    Set myFiles = myWeb.RootFolder.Files
    With myWeb
        For Each myFile In myFiles
            myFileName = myFile.Name
            myTitle = myFile.Title
            myUrl = myFile.Url
        Next
    End With
End Sub
```

Use the **IsOpen** property to check if a file is currently open in **Page** view. The following example returns the **IsOpen** property for a specified **File** object. Notice that the **Edit** method is used to open the file in this example. For more information on using these methods, see the <u>Edit</u> method.

Note You must have a Web site open to run this program.

```
Private Sub CheckForOpenFile()
    Dim myWeb As WebEx
    Dim myFiles As WebFiles
    Dim myFile As WebFile
    Dim myFileToOpen As String
    Dim myMessage As String
    Dim myFileName As String
    Set myWeb = ActiveWeb
    Set myFiles = myWeb.RootFolder.Files
    myFileToOpen = "index.htm"
    myMessage = "This file is currently open."
   With myWeb
        For Each myFile In myFiles
            myFileName = myFile.Name
            If myFileName = myFileToOpen Then
                If myFile.IsOpen = True Then
                    MsgBox (myMessage)
                    Exit Sub
                Else
                    myFile.Edit fpPageViewNormal
                    Exit Sub
                End If
           End If
        Next
    End With
End Sub
```

Use the **Checkin**, **Checkout**, and **UndoCheckout** methods to manage file resources through source control on a Web site. The following statement checks out the first file in the active Web site.

Note You must have a source control project set up in order for this to work.

```
myFileCheckedOut = ActiveWeb.RootFolder.Files(1).Checkout
```

Similar to file management features in Microsoft Visual SourceSafe, FrontPage also provides an **UndoCheckout** method that you can use to return a file to its original state. The following statement returns the file to its original state.

myFileCheckedOut = ActiveWeb.RootFolder.Files(1).UndoCheckout

You can use the **CheckedoutBy** property before attempting to check out a file to see if the file is currently checked out and by whom. The following statement returns the logon alias of the person who checked out a file or is null if the file isn't currently checked out.

```
myWhoCheckedOutFile = ActiveWeb.RootFolder.Files(0).CheckedoutBy
```

Use the **Properties** property to return information about a Web site, such as the type of Web server (vti_webservertype) or if the Web site has a search bot (vti_hassearchbot). The **Properties** property returns a collection of key-value pairs used to maintain the meta information. The following statement returns **True** for the variable mySearchBot if the Web site has a search bot.

```
mySearchBot = ActiveWeb.Properties.Item("vti_hassearchbot")
```

Use the **MetaTags** property to return information about the meta tags contained in the HTML coding of a file. The **MetaTags** property returns a collection of meta tags for a **File** object, such as the generator of the file. The following example returns the file name and meta tags for each file in a Web site.

Note To run this program, you must have a least one Web site open.

```
Private Sub GetMetaTags()
    Dim myWeb As WebEx
    Dim myMetaTag As Variant
    Dim myFiles As WebFiles
    Dim myFile As WebFile
    Dim myMetaTags As MetaTags
    Dim myFileName As String
    Dim myMetaTagName As String
    Set myWeb = ActiveWeb
    Set myFiles = myWeb.RootFolder.Files
    With myWeb
        For Each myFile In myFiles
            Set myMetaTags = myFile.MetaTags
            For Each myMetaTag In myMetaTags
                myFileName = myFile.Name
                mvMetaTagName = myMetaTag
            Next
        Next
    End With
```

End Sub

Use the **SharedBorders** property to return the shared borders on the current Web page or to set new shared borders. The following statement returns the top shared border of the first file in the **Files** collection of the active Web site.

You can also set shared borders on a Web page, as shown in the following statement.

```
ActiveWeb.RootFolder.Files(0).SharedBorders(fpBorderTop) = True
```

Use the **ThemeProperties** property to return information about whether the theme uses vivid colors or active graphics. The following example returns the properties of an applied theme and adds vivid colors to the current theme properties if vivid colors haven't been applied to the specified object.

```
Private Sub CheckThemeProperties()
   Dim myFile As WebFile
   Set myFile = ActiveWeb.RootFolder.Files(0)
   If myFile.ThemeProperties(fpThemeActiveGraphics) Then
      myFile.ApplyTheme myFile.ThemeProperties(fpThemeName), myFil
   Else
      myFile.ApplyTheme myFile.ThemeProperties(fpThemeName), myFil
   End If
End Sub
```

Using File methods

Use the **Copy**, **Delete**, **Edit**, **Move**, or **Open** methods to manage your Web pages. There's a subtle distinction between the **Edit** and **Open** methods. With the **Edit** method, you can open and modify a FrontPage-compatible file into a **PageWindow** object. With the **Open** method, you can open both FrontPage-compatible files and any other type of file such as image or text files, into the file's associated editor. When you use the **Open** method to open a file type that is not FrontPage-compatible, FrontPage does not return a file object. The following example opens a file, deletes a file, and moves a file.

Note To run this example, you must have a Web site called "C:\My Documents\My Webs Sites\Coho Winery".

```
Private Sub OpenFile()
    Dim mvWeb As WebEx
    Dim myFile As WebFile
    Set myWeb = Webs.Open("C:\My Documents\My Webs Sites\Coho Winery
    myWeb.Activate
    Set myFile = myWeb.RootFolder.Files("index.htm")
    myFile.Open
End Sub
Private Sub DeleteFile()
    Dim myWeb As WebEx
    Dim myFile As WebFile
    Set myWeb = ActiveWeb
    Set myFile = myWeb.RootFolder.Files(0)
    myFile.Delete
End Sub
Sub MoveFile()
    Dim myWeb As WebEx
    Dim myFile As WebFile
    Set myWeb = ActiveWeb
    Set myFile = myWeb.RootFolder.Files(0)
    myFile.Move "New Filename", True, True
End Sub
```
WebFolder Object

Multiple objects ^L<u>WebFolder</u> ^LMultiple objects

Represents a folder in a Microsoft FrontPage-based Web site. The **WebFolder** object is a member of the <u>WebFolders</u> collection.

Note The **Folder** object is a pointer to the **WebFolder** object.

The **WebFolders** collection represents all of the folders in a specified Web site. Within the **WebFolders** collection, individual **WebFolder** objects are indexed beginning with zero. The **WebFolder** object is similar to a folder in a directorybased hierarchy; however, the relationship between **WebFolder** objects and **Web** objects is unique. FrontPage provides the ability to create multiple **WebEx** objects on a Web server. Any **WebFolder** *can* represent a Web site, but every **WebFolder** does not necessarily represent a Web site. The folder hierarchy provides the link to folders and files on a Web server directory. The navigation structure provides the underlying structure for the **Web** objects within individual FrontPage-based Web sites.

Using the WebFolder object

Use **WebFolders**(*index*), where *index* is the **property key** of a folder, to return a single **WebFolder** object. The following example returns the file name of the first folder item in the **WebFolders** collection.

```
ActiveDocument.WebFolders(0).Name
```

Use the collection properties such as **Files**, **Folders**, or **Properties**, to return the collection object for the specified item. The following statements return the first specified item in the collection for the active Web site.

```
myFileOne = ActiveWeb.RootFolder.Files(0)
myFolderOne = ActiveWeb.RootFolder.Folders(0)
myPropertyOne = ActiveWeb.Properties("vti_author")
```

Use such properties as **IsExecutable**, **IsReadable**, **IsRoot**, and so on, to check for the specified state of the folder. If you have CGI scripts that you'd like to execute, you can add the scripts to a folder and set the **IsExecutable** property of that folder to **True**. When you have content in a folder that you'd like others to browse, you can set the **IsReadable** property to **True**. If you want to check whether the current folder is the root folder, you can use the **IsRoot** property.The following example checks if files in the current **WebFolder** object are executable, read-only, or located in a root folder.

```
Private Sub GetFolderInfo()
   Dim myWeb As WebEx
   Dim myFolder As WebFolder
   Dim myIsExe As Boolean
   Dim myIsReadable As Boolean
   Dim myIsRoot As Boolean
   Set myWeb = ActiveWeb
   Set myFolder = myWeb.RootFolder.Folders(1)
   With myFolder
        myIsExe = .IsExecutable
        myIsReadable = .IsReadable
        myIsRoot = .IsRoot
   End With
End Sub
```

The **IsExecutable**, **IsReadable**, and **IsWriteable** properties return information about the state of the folder. The following examples show how to set the **IsExecutable** and **IsReadable** properties and read the **IsWriteable** property.

Note You cannot set the **IsWriteable** property, however you can set the **IsExecutable** and **IsReadable** properties for a **WebFolder** object.

```
Sub FolderProperties ()
Dim myFolder As WebFolder
Set myFolder = ActiveWeb.RootFolder.Folders(0)
If myFolder.IsWritable Then
    MsgBox "Folder, " & myFolder.Url & " is writable"
End If
If Not(myFolder.IsReadable) Then
    MyFolder.IsReadable = True
End If
If myFolder.IsExecutable Then
    MyFolder.IsExecutable Then
    MyFolder.IsExecutable = False
End If
End Sub
```

Folders (or **WebFolders** collection) in FrontPage serve two purposes. They can be folders that help manage the contents of a Web site or they can be entire Web sites. A Web site can have multiple sub Web sites below it. The **IsWeb** property returns **True** if the folder in question is a Web subsite. The following example uses the **IsWeb** property to determine if a folder is a Web subsite and, if so, opens the Web site.

Note To run this example, you must have a Web site called "C:\My Documents\My Web Sites\Coho Winery", or you may substitute an alternative Web site URL.

```
Private Sub CheckFolder()
   Dim myFolder As WebFolder
   Set myFolder = ActiveWeb.RootFolder.Folders("Coho Winery")
   If myFolder.IsWeb = True Then
        Webs.Open myFolder.Url
   End If
End Sub
```

Use the **Url** property to return the URL of the current **WebFolder** object. The following statement returns the absolute URL for the eighth folder in the active Web site.

myUrl = ActiveWeb.RootFolder.Folders(7).Url

Use the **Copy**, **Delete**, and **Move** methods to maintain your Web site structure. The following statement copies a **WebFolder** object from one folder to another folder, updates the links during the copy process, and forces an overwrite if the file already exists.

myFolder.Copy("C:\My Web Sites\New Adventure Products", True, True)

WebPackage Object

<u>WebPackage</u>

Represents a Web package that has been created in Microsoft FrontPage Visual Basic for Applications. The **WebPackage** object is an in-memory object only and does not correspond to any FrontPage User Interface element. Instead, use the **WebPackage** object to work with a Web package once you've created it in code.

Using the WebPackage object

Use the **<u>CreatePackage</u>** method to create a WebPackage object. The following example creates a new Web package.

```
Dim objPackage As WebPackage
```

Set objPackage = ActiveWeb.CreatePackage("New Web Package")

Use the **Add** method to add files to the Web package. The following example adds three files to the **WebPackage** object created in the previous code.

```
objPackage.Add objWeb.Url & "/test.htm", fpDepsDefault
objPackage.Add objWeb.Url & "/test2.htm", fpDepsNone
objPackage.Add objWeb.Url & "/test3.htm", fpDepsImages
```

Use the **Subject**, **Author**, **Comany**, and **Title** properties to add information about a Web package. The following example specifies the subject, author, and company for the **WebPackage** object created above. (When you create a Web package, the **Title** parameter for the **CreatePackage** method becomes the value of the **Title** property. You can change the title of a Web package by setting the **Title** property to a new value.)

```
objPackage.Author = "John Smith"
objPackage.Company = "Fourth Coffee"
objPackage.Subject = "This is a new Web package for Fourth Coffee."
```

Use the **<u>Remove</u>** method to remove files that were added by using the **Add** method. The following example removes one of the files added above.

```
objPackage.Remove objWeb.Url & "/test3.htm", fpDepsImages
```

Use the **Save** method to save, or export, a Web package. The following example saves the **WebPackage** object created above.

```
objPackage.Save "c:\NewWebPackage.fwp", True
```

WebWindowEx Object

WebWindowEx ^LMultiple objects

Represents a Microsoft FrontPage application window in which a Web site is opened. The **WebWindowEx** object is a member of the <u>WebWindows</u> collection. The **WebWindows** collection represents all of the open application windows in a specified Web site or within FrontPage. Within the **WebWindows** collection, individual **WebWindowEx** objects are indexed beginning with zero. Each Web site that is opened in FrontPage is contained in a new **WebWindowEx** object, unless it is opened in a windowless environment by setting the **Visible** property of the **WebWindowEx** object to **False**. For more information on windowless environments, see <u>Coding in a Windowless</u> <u>Environment</u>.

Using the WebWindow object

Use the **WebWindow** property to return information about an open **WebWindowEx** object. You can also use the **PageWindows** property to return information about the collection of open pages in a **WebWindowEx** object. Use **WebWindows**(*index*), where *index* is the index number of an application window item, to return a single **WebWindow** object. The following statement returns the **ViewMode** property of the first Web site in the **WebWindows** collection.

```
myViewMode = WebWindows(0).ViewMode
```

You can also use the **ViewMode** property to switch between view modes by setting the view mode as shown in the following statement, which switches the current view mode to Navigation view.

ActiveWebWindow.ViewMode = fpWebViewStructure

The **Activate** method puts the focus on the specified **WebWindowEx** object. The following statements activates the first Web sites in the collection of open windows.

```
myWebWindow = WebWindows(0)
myWebWindow.Activate
```

The **ActivePageWindow** property returns the active **PageWindowEx** object. The following statements return the URL and the caption of the active **PageWindowEx** object. The value returned for the caption in this case is a file name, such as "Index.htm".

```
urlThisDoc = WebWindow.ActivePageWindow.Document.Url
fileName = WebWindow.ActivePageWindow.Caption
```

You can also return the **Caption** property from the **WebWindowEx** object. In this case, the text that is returned reflects the text in the title bar of the FrontPage application window, which consists of the application name and the URL of the

specified **WebWindowEx** object, such as "Microsoft FrontPage – C:\My Documents\My Web Sites\Adventure Works". The following statement returns the value of the **Caption** property of the **WebWindowEx** object.

thisCaption = WebWindow.Caption

Use the <u>Close</u> method to close a **WebWindowEx** object. The following statement closes the specified **WebWindow**.

```
Set myWebWindowOne = WebWindows(0)
myWebWindowOne.Close
```

Use the **ViewMode** property to return or set one of the values shown in the following table. You can also use these enumerated values to switch views in FrontPage.

Enumermated Constant	Value	Corresponding View in FrontPage
FpWebViewLinks	0	Hyperlinks view
FpWebViewFolders	1	Folders view
FpWebViewStructure	2	Navigation view
FpWebViewPage	3	Page view
fpWebViewAllFiles	4	View a list of every file in Reports view
FpWebViewTodo	5	View a To Do list in Tasks view
FpWebViewBrokenLinks	6	View a list of broken hyperlinks in Reports view
FpWebFiewSiteSummary	7	Site Summary view in Reports view

The following statement sets the **ViewMode** property to **fpWebViewPage**.

WebWindows(0).ViewMode = fpWebViewPage

Use the **Visible** property to return or set a Boolean value for the state of a **WebWindowEx** object. The **Visible** property returns **True** if a **WebWindowEx** object is visible. The following statement sets a **WebWindowEx** object to an invisible state.

WebWindow.Visible = False

Use the **Web** property to return information about the **WebEx** object. The following statement returns the number of properties for the specified Web sites.

myProperties = ActiveWeb.WebWindows(0).Web.Properties.Count

Activate Method

Activates the specified **PageWindowEx**, **WebEx**, or **WebWindowEx** object. When you have multiple objects open, you can use the **Activate** method to work with a specific **PageWindowEx**, **WebEx**, or **WebWindowEx** object.

expression.Activate

expression An expression that returns a **PageWindowEx**, **WebEx**, or **WebWindowEx** object.

Remarks

The **Activate** method has no effect on a windowless **PageWindowEx** object.

Example

This statement activates the home page for the Adventure Works Web site.

If myWeb.Url = "C:\My Web Sites\Adventure Works" Then myWeb.Activate



Add Method

Add method as it applies to the ListFields object.

Adds a new **ListField** object to the **ListFields** collection.

expression.Add(Name, Description, FieldType, Required, DefaultValue)

expression Required. An expression that returns one of the above objects.

Name Required. A **String** that represents the name of the field.

Description Optional. A **String** that represents a description of the field.

DefaultValue Optional. A Variant that defines the default value.

FieldType Optional. An <u>FpFieldType</u> constant that represents the type of the new field.

FpFieldType can be one of these FpFieldType constants.fpFieldAttachmentsfpFieldChoicefpFieldComputedfpFieldCounterfpFieldCurrencyfpFieldDateTimefpFieldFilefpFieldIntegerfpFieldIntegerfpFieldMultiLinefpFieldNumberfpFieldRatingScalefpFieldSingleLine defaultfpFieldTrueFalse

fpFieldURL

Required Optional. A **Boolean** that determines if this is a required field. **True** if the field is required.

Add method as it applies to the **Lists** object.

Adds a new **List** object to the **Lists** collection.

expression.Add(Name, ParentFolder, ListType, Description)

expression Required. An expression that returns a **Lists** object.

Name Required. A **String** that represents the name of the new list.

ParentFolder Optional. A **String** that represents the parent folder associated with the **List**.

ListType Optional. An <u>FpListType</u> constant that represents the type of list.

FpListType can be one of these FpListType constants. fpListTypeBasicList *default* fpListTypeDiscussion fpListTypeDocumentLibrary fpListTypeSurvey

Description Optional. A **String** that represents a description of the **List** object.

Add method as it applies to the **NavigationNodes** object.

Adds a new file to the list of available items in the **NavigationNodes** collection. Use this method to add a new file to the navigation structure. For more information on using navigation nodes, see the **RootNavigationNode** property for the **Web** object, or the **Children** property for the **NavigationNode** object.

expression.Add(Url, NodeLabel, ModificationType, LeftSibling)

expression Required. An expression that returns a **NavigationNodes** collection.

Url Required **String**. A string that contains the path for the Web server where the file will be stored. This can be any absolute URL, such as http://web server or file://file system for disk-based Webs.

NodeLabel Required **String**. A string of text used to identify the **NavigationNode** object when viewing the navigation structure in Navigation view. The *NodeLabel* argument is used only as an aid to identification.

ModificationType Required **<u>FpStructModType</u>**. The node modification type.

FpStructModType can be one of these FpStructModType constants.

fpStructBaseOnSibling Use this setting if you want to add a new node to the right of the node designated in the left sibling argument.

fpStructLeftmostChild The leftmost node in the current navigation structure. **fpStructRightmostChild** The rightmost node in the current navigation structure.

LeftSibling Optional Variant. An index into the NavigationNodes collection. It can be either a string that represents a URL, or a number that represents a node in the collection.

Remarks

Adding a new file onto the Web server (using the **WebFiles.Add** method) doesn't imply that you are automatically introducing the file into the navigation structure. A **NavigationNode** object must be created separately for the file. To create a new **NavigationNode** object, use the **Add** method for the **NavigationNodes** collection.

Note When a template is used to create a new Web, navigation nodes are automatically created for the files that have been added to the Web by the template.

Add method as it applies to the **PageWindows** object.

Adds a new **<u>PageWindowEx</u>** object to the list of available open items in the **PageWindows** collection.

Note Opening a new or existing file object by using the **Add** method for the **PageWindowEx** object also adds the **PageWindowEx** object that is associated with the opened file to the **PageWindows** collection.

expression.Add(FileUrl)

expression Required. An expression that returns a **PageWindows** collection object.

FileUrl Optional String. A string that contains the path for the Web server where the page will be stored. This can be any absolute URL for a file, such as http://web server/file or file://file system/file for disk-based Webs.

Remarks

You can use one of three methods when you want to open HTML pages in Microsoft FrontPage Page view— the **Add**, **Edit** or **Open** method. When you want to open, edit, then save a file in Page view that exists either on a file server or on a file system on your hard disk, use the **Add** method for the **PageWindows** object, as shown in the following statements.

Note It doesn't matter where the files exist; they could reside on a hard disk, server, or a FrontPage Web.

```
Dim myFile As String
myFile = "C:\Adventure Works HTML Files\Hiking.htm"
ActiveWeb.ActiveWebWindow.PageWindows.Add (myFile)
```

With this method, you haven't added the file to a FrontPage-based Web— you've just opened it. If myFile is part of a Web, and the Web is currently not open, FrontPage will also open the Web.

When you want to open and edit an HTML file that exists on a Web, use the **Edit** method for the **Files** collection in the root folder, as shown in the following statement.

```
ActiveWeb.RootFolder.Files("Hiking.htm").Edit
```

You can use the **Open** method in the same way. However, you should reserve the **Open** method for opening files that are not HTML files, such as Microsoft Word documents, image files, and so on.

You can also use the **Add** method to open a new unsaved **PageWindow** object. You can use either of the following statements to open an unsaved page window.

```
Set myUnsavedPageWindow = ActiveWebWindow.PageWindows.Add()
Set myUnsavedPageWindow = ActiveWebWindow.PageWindows.Add("")
```

Note You can use the expression ActiveWebWindow.PageWindows.Add("C:\My

Documents\My Webs\index.htm") as a valid expression as long as index.htm is a valid FrontPage-based file that resides in My Webs. However, if index.htm does not reside in My Webs, your code will fail. To add a new page, you must follow the procedure described earlier in this section.

Add method as it applies to the **Properties** object.

Adds a new property to the list of available items in the **Properties** collection.

expression.Add(PropertyKey, PropertyValue)

expression Required. An expression that returns the **Properties** collection.

PropertyKey Required **String**. A string that contains the name of the property that you want to add. For more information, see the table of properties in the **Properties** collection topic.

PropertyValue Required Variant. The value of the property.

Remarks

You can programmatically add and remove categories and approval ratings for the **Properties** collection.

Add method as it applies to the **WebFiles** object.

Adds a new **WebFile** object to the list of available items in the **WebFiles** collection. A **WebFile** object is not restrictive and can be any type of file; it is not restricted to an HTML file type— it could be an image file, a movie, or an executable file.

Note Use this method to add a new file to a FrontPage-based Web.

expression.Add(FileUrl, ForceOverwrite)

expression Required. An expression that returns a **WebFiles** collection.

FileUrl Required String. A string that contains the URL for the file such as "Inventory.htm". This can be any absolute URL for a file, such as http://web server/file or file://file system/file for disk-based Webs.

ForceOverwrite Optional **Boolean**.

Remarks

Accessing a single **WebFile** object through the **WebFiles** collection provides reference to a **WebFile** object that doesn't have access to the Page object model until the Web file is opened. Once the file is open, the **PageWindow** object associated with the file provides access to the Page object model that is compatible with Microsoft Internet Explorer 4.0 and later. For further information on using the Page object model in your Web pages, see <u>Exploring the Object Model in FrontPage</u>.

Add method as it applies to the WebFolders object.

Adds a new **WebFolder** object to the list of available items in the **WebFolders** collection.

expression.Add(FolderUrl)

expression Required. An expression that returns a **WebFolders** collection.

FolderUrl Required **String**. A string that contains the URL for the folder, such as the Images folder in C:\My Webs. This can be any absolute URL for a folder, such as http://web server/folder or file://file system/folder for disk-based webs.

Add method as it applies to the **WebPackage** object.

Returns a **Boolean** that represents whether the specified file was successfully added to the Web package.

expression.Add(Url, flags)

expression Required. An expression that returns a <u>WebPackage</u> object.

Url Required **String**. The path and file name of the file to be added.

flags Optional **FpDependencyFlags**. Specifies how to handle dependencies for the specified file. Dependencies are included only if they exist within the current Web site.

FpDependencyFlags can be a combination of one or more of the following **FpDependencyFlags** constants.

fpDepsDefault	Includes all images, link bars, hyperlinks, lists, shared borders, and themes.	
fpDepsImages	Includes all images.	
fpDepsLinkbars	Includes all link bars.	
fpDepsLinks	Includes all pages to which there are hyperlinks.	
fpDepsLists	Includes lists that may be needed in order for the page to render correctly.	
fpDepsNone	Includes no dependencies.	
fpDepsRecurse	Includes all files that are in a specified folder.	
fpDepsSharedBorders Includes all shared borders.		
fpDepsThemes	Includes all themes.	

Remarks

Use the **CreatePackage** method to create a new Web package. Then use the **Add** method to add pages and their specified dependencies to the Web package. You can create Web packages from files in Web sites based on Microsoft Windows SharePoint Services and in disk-based Web sites.

Add method as it applies to the **Webs** object.

Adds a new Web to the list of available items in the **Webs** collection.

Security Avoid using hard-coded passwords in your applications. If a password is required in a procedure, request the password from the user, store it in a variable, and then use the variable in your code. For recommended best practices on how to do this, see <u>Security Notes for Microsoft Office Solution</u> <u>Developers</u>.

expression.Add(WebUrl, UserName, Password, WebOpenFlags)

expression Required. An expression that returns a **Webs** collection object.

WebUrl Required **String**. A string that contains the path for the Web server where the Web will be stored. This can be any absolute URL for a Web, such as http://web server or file://file system for disk-based Webs.

UserName Optional **String**. The user's logon name for the Web server.

Password Optional **String**. The user's password for the Web server.

WebOpenFlags Optional. An <u>FpWebOpenFlags</u> constant that represents the behavior of the new Web.

FpWebOpenFlags can be one of these FpWebOpenFlags constants. **fpOpenInWindow** *default* **fpOpenNoWindow**

Add method as it applies to the **WebWindows** object.

Adds a new **WebWindowEx** object to the **WebWindows** collection.

expression.Add(ViewModeEx)

expression Required. An expression that returns a **WebWindows** collection.

ViewModeEx Required. An <u>**FpWebViewModeEx**</u> enumerated constant that represents the information displayed in the new window.

FpWebViewModeEx can be one of these FpWebViewModeEx constants. **fpWebViewExAccessibility** fpWebViewExAllFiles **fpWebViewExAssignedTo** fpWebViewExBrokenLinks fpWebViewExBrowserTypes **fpWebViewExCategories fpWebViewExCheckoutStatus** fpWebViewExComponentErrors **fpWebViewExCSSLinks** fpWebViewExDailyPageHits **fpWebViewExDailySummary fpWebViewExFolders fpWebViewExLinks** fpWebViewExMasterPages fpWebViewExMonthlyPageHits **fpWebViewExMonthlySummary fpWebViewExNavigation fpWebViewExOlderFiles fpWebViewExOsTypes fpWebViewExPage** fpWebViewExPublishStatus fpWebViewExRecentlyAddedFiles fpWebViewExRecentlyChangedFiles **fpWebViewExReferringDomains fpWebViewExReferringURLs**

fpWebViewExRemoteSite fpWebViewExReviewStatus fpWebViewExSearchStrings fpWebViewExSharedBorders fpWebViewExSiteSummary fpWebViewExSlowPages fpWebViewExThemes fpWebViewExThemes fpWebViewExTodo fpWebViewExUnlinkedFiles fpWebViewExUsageSummary fpWebViewExVisitingUsers fpWebViewExWeeklyPageHits fpWebViewExWeeklyPageHits

Example

As it applies to the **NavigationNodes** object.

This example adds a new node called footnote.htm to the list of items in the **NavigationNodes** collection.

Note To run this example, you must have a Web site called "C:\My Documents\My Webs\Coho Winery" that contains a file called footnote.htm. Or, you may substitute an alternative Web site URL or file name.

```
Private Sub AddNewNavNode()
Dim myHome As NavigationNode
Dim myNewNode As NavigationNode
Dim myFileUrl As String
myFileUrl = "C:\My Documents\My Webs\Coho Winery\footnote.htm"
Set myHome = ActiveWeb.HomeNavigationNode
Set myNewNode = _______myHome.Children.Add (myFileUrl, ______"Footnote", fpStructLeftmostChild)
ActiveWeb.ApplyNavigationStructure
End Sub
```

As it applies to the **PageWindows** object.

This example adds the Inventory page for Coho Winery to the list of items in the **PageWindows** collection. When you add a page using the *FileUrl* argument, the page that you want to add must exist as a file— you cannot create a new page using this argument. To create an *unsaved* new page, see the description in the **Add** method.

Note To run this example, you must have a Web site called "C:\My Documents\My Webs\Coho Winery" that contains a file called Inventory.htm. Or, you may substitute an alternative Web site URL or file name.

Private Sub AddPage() Dim myPageWindows As PageWindows Dim myPage As String

```
Set myPageWindows = ActiveWeb.ActiveWebWindow.PageWindows
myPage = "C:\My Documents\My Webs\Coho Winery\Inventory.htm"
myPageWindows.Add (myPage)
```

As it applies to the **Properties** object.

End Sub

This example adds a new file to the list of items in the **Properties** collection.

Note To run this example, you must have a Web site called "C:\My Documents\My Webs\Coho Winery\Zinfandel.htm" that contains a file called footnote.htm. Or, you may substitute an alternative Web site URL or file name.

```
Private Sub CopyrightAdd()
Dim myWeb As WebEx
Dim myCopyright As String
Dim myCopyrightProperty As Variant
myCopyright = "Copyright 1999 by Coho Winery"
Set myWeb = Webs.Open("C:\My Webs\Coho Winery")
myWeb.Activate
ActiveWeb.Properties.Add "Copyright", myCopyright
ActiveWeb.RootFolder.Files("Zinfandel.htm").Open
ActiveWeb.RootFolder.Files("Copyright")
ActiveWeb.Properties("Copyright")
ActiveWeb.Properties("Copyright")
ActiveWeb.Close
End Sub
```

As it applies to the **Webs** object.

This example adds a new item to the list of files in the **Webs** collection.

```
Webs.Add ("C:\My Documents\My Webs\Coho Winery")
```

As it applies to the **WebFiles** object.

This example adds a new **WebFile** object to the list of items in the **Files** collection.

ActiveWeb.RootFolder.Files.Add ("C:\New Web Files\Sales Statistics.h

As it applies to the **WebFolders** object.

This example adds a folder to the list of items in the **WebFolders** collection.

ActiveWeb.RootFolder.Folders.Add ("Distribution Centers")

As it applies to the **WebPackage** object

The following example creates a new Web package and adds the page "test.htm" to the package, including all dependencies for the page, and then saves the new Web package.

```
Dim objWeb As WebEx
Dim objPackage As WebPackage
Set objWeb = ActiveWeb
Set objPackage = objWeb.CreatePackage("New Web Package")
With objPackage
    .Author = "John Smith"
    .Company = "Fourth Coffee"
    .Subject = "This is a new Web package for Fourth Coffee."
    .Add objWeb.Url & "/test.htm", fpDepsDefault
    .Save "c:\NewWebPackage.fwp", True
End With
```

AddChoice Method

Adds a new choice to the list of available choices for the current field. The field must be of type **ListFieldChoice** .

expression.AddChoice(text, Index)

expression Required. An expression that returns a **ListFieldChoice** object.

text Required. A **String** that represents the text that will appear in the drop-down list or beside a radio button.

Index Optional. A **Long** that represents the position of the choice within the list of choices.

Example

The following example adds two choices to a choice field named NewChoiceField in the first list of the active Web site. The new choices are SaleOption1, which will appear first in the list, and SaleOption2, which will appear second in the list. The relative positions of the choices are determined by the optional *Index* argument.

```
Sub AddChoice()
Dim objApp As FrontPage.Application
Dim objLstFlds As listFields
Dim objFldChoice As ListFieldChoice
Set objApp = FrontPage.Application
Set objLstFlds = objApp.ActiveWeb.Lists.Item(0).Fields
'Set a reference to the new field and
'add two new choices to the list.
Set objFldChoice = objLstFlds.Item("NewChoiceField")
objFldChoice.AddChoice text:="SaleOption1", Index:=1
objFldChoice.AddChoice text:="SaleOption2", Index:=2
```

End Sub



AddLinkBar Method

Returns a **<u>NavigationNode</u>** object that represents a link bar.

expression.AddLinkBar(NodeLabel, ModificationType, LeftSibling)

expression Required. An expression that returns a **NavigationNodes** collection.

NodeLabel Required. A **String** that represents the label or name of the link bar.

ModificationType Required. An <u>**FpStructModType**</u> constant that represents the structure of the link bar.

FpStructModType can be one of these FpStructModType constants. fpStructBaseOnSibling Base the link bar on its closest sibling node. fpStructLeftmostChild Base the link bar on its leftmost child node. fpStructRightmostChild Base the link bar on its rightmost child node.

LeftSibling Optional. A Variant that represents the left sibling of the node. This value is used to locate the new link bar in the hierarchy.

Remark

The new node will not appear in the link bar. Only the node's children will appear in the link bar.

Note The new link bar must be added to the structure before child nodes can be added to it.

Example

The following example creates a new link bar that is based on its sibling in the hierarchy of navigation nodes.

End Sub
ApplyChanges Method

Apply changes to the specified object. Changes to a property of a **WebEx**, **WebFile**, or **WebFolder** object are not applied until you use the **ApplyChanges** method for the specified object.

expression. ApplyChanges

expression An expression that returns a **Properties** collection object.

This example changes the value of the **vti_title** property for a file called Zinfandel.htm and applies the change.

```
Private Sub ChangeProperties()
   Dim myProperties As Properties
   Set myProperties = _______ActiveWeb.RootFolder.Files("Zinfandel.htm").Properties
   myProperties("vti_title") = "Rogue Cellars Wine List"
   myProperties.ApplyChanges
End Sub
```

ApplyDynamicTemplate Method

Applies a Dynamic Web Template to a document.

expression.ApplyDynamicTemplate(bzMaster, pState,)

expression Required. An expression that returns a <u>WebFile</u> object.

bzMaster Required **String**. The path and filename for the Dynamic Web Template.

pState Required **DynamicTemplateState**. Specifies the region mapping for the Dynamic Web Template. Use the <u>SetHeadMapping</u> and <u>SetBodyMapping</u> methods of the <u>DynamicTemplateState</u> object to customize region mapping.

To detach a Dynamic Web Template use the **ApplyDynamicTemplate** method with the *bzMaster* parameter set to an empty string.

The following example applies the specified Dynamic Web Template file to the specified file.

Dim objState As DynamicTemplateState Dim objFile As WebFile Set objState = Application.CreateDynamicTemplateState Set objFile = ActiveWeb.LocateFile("home.htm")

objFile.ApplyDynamicTemplate "template.dwt", objState

ApplyNavigationStructure Method

Applies the navigation structure to the specified object.

expression. Apply Navigation Structure

expression An expression that returns a <u>WebEx</u> object.

There are two details to keep in mind when programmatically creating files and navigation nodes:

- Navigation labels cannot be empty.
- Changes to the navigation structure can be lost if you don't apply the navigation structure before starting operations that affect the content of the Web site such as moving or adding files or folders.

This example adds a navigation node as the rightmost child node and then applies the changes to the navigation structure.

```
Private Sub AddNewNavNode()
   Dim myWeb As WebEx
   Dim myChildNodes As NavigationNodes
   Dim myNewNavNode As NavigationNode
   Set myWeb = ActiveWeb
   Set myChildNodes = _______myWeb.RootFolder.Files(1).NavigationNode.Children
   myNewNavNode = _______myChildNodes.Add(myWeb.Url & "Sale.htm", "Sale", ______fpStructRightmostChild)
   myWeb.ApplyNavigationStructure
End Sub
```

ApplyTemplate Method

Some of the content in this topic may not be applicable to some languages.

Applies an existing HTML template to the current Web site.

expression.ApplyTemplate(TemplateDir, fOverWrite)

expression Required. An expression that returns one of the objects in the Applies To list.

TemplateDir Required **String**. The path of the template.

fOverWrite Optional. A **Boolean** that determines if the current template will be overwritten. If **True**, the current template will be overwritten. If **False**, the current template will not be overwritten. The default value is **False**.

The following example adds a specified template to the current Web site using the **ApplyTemplate** method. The method is called with the *fApplyThemes* and the *fOverWrite* arguments set to **False**. The themes will not be applied to the new Web site and any existing template will not be overwritten.

```
Sub UseTemplate()
                        'Applies a template to the current Web site
                        'or applying themes.
    Dim objApp As FrontPage.Application
    Dim objWeb As WebEx
    Dim strPath As String
    Dim strname As String
    Set objApp = FrontPage.Application
    Set objWeb = objApp.ActiveWeb
    'Set variable to template directory.
    strPath = "C:\Program Files\Microsoft Office\Templates\"
    'Prompt the user for the file name of the template.
    strname = InputBox("Enter the file name of the template you wish
    'Add the template name to the path in order to
                                'create a full path name.
    strPath = strPath & strname
    'Apply the template to the new Web site.
    objWeb.ApplyTemplate TemplateDir:=strPath, _
        fOverWrite:=False
```

End Sub



ApplyTheme Method

Applies the value contained in the *ThemeName* argument to the property named in the *ThemeProperties* argument. For example, a theme can be applied to a **WebFile**, **WebFiles**, **PageWindowEx**, or **WebEx** object in a Microsoft FrontPage-based Web site.

expression.ApplyTheme(ThemeName, ThemeProperties)

expression An expression that returns an object in the Applies To list.

ThemeName Required **String**. A string that contains the name of the theme that you want to apply to a file. The *ThemeName* parameter can be one of the following:

aftrnoon	concrete	modular	strtedge
arcs	corporat	nature	studio
arctic	cypress	network	sumipntg
artsy	deepblue	papyrus	sunflowr
axis	echo	passport	tabs
balance	eclipse	piechart	technolo
bars	edge	pixel	topo
blank	evergreen	poetic	travel
blends	expeditn	profile	water
blitz	folio	quad	watermar
blocks	glacier	radial	waves
bluecalm	global	refined	willow
blueprnt	highway	ricepapr	zero
boldstri	ice	ripple	
breeze	indust	rmnsque	
canyon	inmotion	sandston	
capsules	iris	satin	
cascade	journal	sky	
checkers	layers	slate	
citrus	level	sonora	
classic	loosegst	spiral	
compass	mdshapes	spring	

ThemeProperties Optional <u>**FpThemeProperties**</u>. The properties associated with the theme.

FpThemeProperties can be one of these FpThemeProperties constants.fpThemeActiveGraphicsfpThemeBackgroundImagefpThemeCSSfpThemeDefaultSettingsfpThemeNamefpThemeNoBackgroundImagefpThemeNoCSSfpThemeNormalColors defaultfpThemeNormalGraphicsfpThemePropertiesAllfpThemePropertiesNonefpThemeVividColors

The following code applies the Sumi Painting theme to a file with active graphics.

```
Dim strTheme As String
strTheme = "sumipntg"
Call WebFile.ApplyTheme(strTheme, fpThemeActiveGraphics)
```

To change more than one theme property when applying the theme, use the plus sign (+), as shown in the following example.

```
strTheme = "sumipntg"
WebFile.ApplyTheme(strTheme, _
fpThemeVividColors + fpThemeActiveGraphics)
```

This method is essentially the same one you'd use for applying a theme to a **PageWindowEx** or **WebEx** object.

This example contains a function, ApplyThemeToFilesInFolder, and a procedure that you can modify to apply any of the available themes. This example applies the Artsy theme to all files in a specified folder.

Note To run this example, copy the code into a module in the Microsoft Visual Basic Editor and run the ChangeToArtsy procedure.

```
Function ApplyThemeToFilesInFolder(myThemeName As String, _
        myFolderObject As WebFolder) As Boolean
    Dim myFile As WebFile
    Dim myTheme As Theme
    On Error GoTo ERR
    For Each myFile In myFolderObject.Files
        Call myFile.ApplyTheme(myThemeName, fpThemePropertiesAll)
    Next myFile
   ApplyThemeToFilesInFolder = True
    Exit Function
ERR:
    MsgBox "An error occurred: " & ERR.Description, vbCritical, "Err
    ApplyThemeToFilesInFolder = False
Exit Function
End Function
Private Sub ChangeToArtsy()
    ApplyThemeToFilesInFolder "artsy", ActiveWeb.RootFolder
End Sub
```

CancelRequests Method

Immediately cancels all requests to the **WebEx** object without saving.

expression.CancelRequests

expression An expression that returns a **WebEx** object.

The **CancelRequests** method can be used to stop a process that may be taking too long, may appear to be in an infinite loop, or that may be coming from a questionable source.

```
Private Sub CancelRequestsToWeb_Click()
    Dim myWeb As WebEx
    Set myWeb = ActiveWeb
    myWeb.CancelRequests
End Sub
```

Checkin Method

Checks the specified <u>WebFile</u> object into the source control project.

Note You must have a source control project in place before using this method. For information about source control projects, refer to <u>Managing Source Control</u> <u>Projects</u>.

expression.Checkin(Comment, KeepCheckedout)

expression An expression that returns a **WebFile** object.

Comment Optional **String**. A description string.

KeepCheckedout Optional **Boolean**. **True** keeps the file checked out. Default value is **False**.

The *KeepCheckedout* argument provides the ability to have the file remain in a checkedout state while the user checks the file in to Microsoft Visual SourceSafe to record the changes. This does not apply to Microsoft FrontPage Light Weight source control.

The program in this example performs the following items:

- Checks out a file and puts the page in edit mode.
- Adds a welcome message to the page.
- Checks if the open page has been modified.
- Saves the page if it has been modified.
- Closes the file and checks it into the existing source control project.

Note To run this example, you must have a source control project in place on a Web site with a file called "C:\My Documents\My Web Sites\Rogue Cellars\Zinfandel.htm." Or, you may substitute a Web site and file of your choice.

```
Private Sub CheckinFile()
    Dim myWeb As WebEx
    Dim myFile As WebFile
    Dim myPageWindow As PageWindowEx
    Dim myWelcome As String
    Set myWeb = Webs("C:/My Web Sites/Rogue Cellars")
    myWelcome = "Welcome to my Web Site!"
    Set myFile = myWeb.RootFolder.Files("Zinfandel.htm")
    myFile.Checkout
    Set myPageWindow = myFile.Edit(fpPageViewNormal)
    With myPageWindow
            myPageWindow.Document.body.insertAdjacentText("BeforeEnd
                myWelcome)
        If myPageWindow.IsDirty = True Then myPageWindow.Save
            .Close
    End With
    myFile.Checkin
End Sub
```

Checkout Method

Checks the specified **WebFile** object out to the source control project.

Note You must have a source control project in place before using this method. For information about source control projects, refer to <u>Managing Source Control</u> <u>Projects</u>.

expression.Checkout(ForceCheckout)

expression An expression that returns a **WebFile** object.

ForceCheckout Optional **Boolean**. Forces a checkout, even if the file is already checked out. **True** forces a checkout of the file. Default value is **False**.

The *ForceCheckout* argument provides the administrator with the ability to force a checkout in cases where a file has been checked out by a user who is unavailable to check the file back in.

The program in this example performs the following:

- Checks out a file from an existing source control project and puts the file in edit mode.
- Adds a welcome message to the document.
- Checks if the open page has been modified.
- Saves the page, if it has been modified.
- Closes the file and checks it into the existing source control project.

Note To run this example, you must have a source control project in place on a Web site with a file called "C:\My Documents\My Web Sites\Rogue Cellars\Zinfandel.htm". Or, you may substitute an alternative Web site and file name.

```
Private Sub CheckoutFile()
    Dim myWeb As WebEx
    Dim myFile As WebFile
    Dim myPageWindow As PageWindowEx
    Dim myWelcome As String
    Set myWeb = Webs("C:/My Web Sites/Rogue Cellars")
    myWelcome = "Welcome to my Web Site!"
    Set myFile = myWeb.RootFolder.Files("Zinfandel.htm")
    myFile.Checkout
    Set myPageWindow = myFile.Edit(fpPageViewNormal)
    With myPageWindow
        myPageWindow.Document.body.insertAdjacentText("BeforeEnd", _
                myWelcome)
        If myPageWindow.IsDirty = True Then myPageWindow.Save
            .Close
    End With
    myFile.Checkin
End Sub
```

ClearChoices Method

Removes the available choices for a choice field.

expression.ClearChoices

expression Required. An expression that returns one of the objects in the Applies To list.



Close Method

Close method as it applies to the **PageWindowEx** object.

Closes the specified **<u>PageWindowEx</u>** object.

expression.Close(ForceSave, PromptUser)

expression Required. An expression that returns the above object.

ForceSave Optional **Boolean**. **True** forces the specified file to be saved before the Close method is completed. Default is **False**.

PromptUser Optional **Boolean**. **True** prompts the user before closing the page. Default is **False**.

Close method as it applies to the **PageWindows** object.

Closes the specified pages in the **PageWindows** collection, or, if **Null**, closes all open pages in the **PageWindows** collection.

expression.Close(Index, ForceSave, PromptUser)

expression Required. An expression that returns one of the above objects.

Index Optional **Variant**. Refers to an individual item in the **PageWindows** collection. Can be any number corresponding to an item in the collection, with the index starting at zero.

ForceSave Optional **Boolean**. **True** forces the specified file to be saved before the **Close** method is completed. Default is **False**.

PromptUser Optional **Boolean**. **True** prompts the user before closing the pages. Default is **False**.

Close method as it applies to the **WebWindows** object.

Closes the specified **<u>WebWindowEx</u>** object.

expression.Close(Index)

expression Required. An expression that returns one of the above objects.

Index Optional **Variant**. Refers to an item in the **WebWindows** collection. Can be any number corresponding to an item in the collection, with the index starting at zero.

Close method as it applies to the **WebEx** and **WebWindowEx** objects.

Closes the specified object.

expression.Close

expression Required. An expression that returns one of the above objects.

As it applies to the **PageWindowEx** object.

The following example closes the active page window.

```
Sub CloseWindow()
'Closes the active page window
Dim objApp As FrontPage.Application
Set objApp = FrontPage.Application
If Not objApp.ActivePageWindow Is Nothing Then
objApp.ActivePageWindow.Close ForceSave:=True
End If
```

End Sub

As it applies to the **PageWindows** collection.

The following example closes the first page window of the first Web site in the **WebWindows** collection.

```
Sub CloseWindow()
'Closes a page window
Dim objApp As FrontPage.Application
Dim objPgeWindows As PageWindows
Set objApp = FrontPage.Application
Set objPgeWindows = objApp.ActiveWeb.WebWindows(0).PageWindows
objPgeWindows.Close Index:=0, ForceSave:=True
```

End Sub

As it applies to the **WebWindows** object.

The following example closes all open Web windows.

```
Sub CloseWindow()
'Closes all Web page windows.
```

```
Dim objApp As FrontPage.Application
```

Dim objPgeWindows As WebWindows

```
Set objApp = FrontPage.Application
Set objWebWindows = objApp.ActiveWeb.WebWindows
objWebWindows.Close
```

End Sub

```
As it applies to the WebEx and WebWindowEx objects.
```

The following example closes the active Web site (if one exists).

```
Sub CloseWindow()
'Closes the active document
Dim objApp As FrontPage.Application
Set objApp = FrontPage.Application
If Not objApp.ActiveWeb Is Nothing Then
objApp.ActiveDocument.Close
End If
```

End Sub



ConvertToField Method

Changes a field from one type to another and returns the object specified in the *Type* parameter.

expression.ConvertToField(Type)

expression Required. An expression that returns one of the objects in the Applies To list.

Type Required **FpFieldType**. Specifies the type of field to which to convert the specified field.

FpFieldType can be one of the following **FpFieldType** constants.

fpFieldAttachments	Returns a ListFieldAttachments object.
fpFieldChoice	Returns a ListFieldChoice object.
fpFieldComputed	Returns a ListFieldComputed object.
fpFieldCounter	Returns a ListFieldCounter object.
fpFieldCurrency	Returns a ListFieldCurrency object.
fpFieldDateTime	Returns a ListFieldDateTime object.
fpFieldFile	Returns a ListFieldFile object.
fpFieldInteger	Returns a ListFieldInteger object.
fpFieldLookup	Returns a ListFieldLookup object.
fpFieldMultiLine	Returns a ListFieldMultiline object.
fpFieldNumber	Returns a ListFieldNumber object.
fpFieldRatingScale	Returns a ListFieldRatingScale object.
fpFieldSingleLine	Returns a ListFieldSingleLine object.
fpFieldTrueFalse	Returns a ListFieldTrueFalse object.
fpFieldURL	Returns a ListFieldURL object.

The following chart specifies whether one field type can be converted to another field type and how the conversion works if special conversion is necessary.

From Field/ ToField	Text	Choice	Note	Note (Rich Text Enabled)	Number
Text	N/A	Yes	Yes	Yes	Yes; converts numbers and sets other values to NULL
Choice	Yes	N/A	Yes	Yes	Yes; converts numbers and sets other values to NULL
Note	Yes; converts and truncates text to less than 255 characters.	Yes; converts and truncates text to 255 characters.	N/A	Yes	Yes; converts numbers and sets other values to NULL
Note (Rich Text Enabled)	No	No	Yes	N/A	No
Number	Yes	Yes	Yes	Yes	N/A
Currency	Yes	Yes	Yes	Yes	Yes

DateTime	Yes	Yes	Yes	Yes	No
Boolean	Yes; converts to 0 or 1.				
Multi- value choice	Yes	Yes	Yes	Yes	Yes

Copy Method

Copies the specified object to a designated URL. During the copy process you can choose to update hyperlinks or force a file overwrite if the specified object has the same name as the designated object.

expression.Copy(DestinationUrl, Unused, ForceOverwrite)

expression An expression that returns a **WebFile** or **WebFolder** object.

DestinationUrl Required **String**. The target URL.

Unused Optional **Boolean**. This parameter is unused. Setting it or not setting it will have no effect on the functionality of the **Copy** method.

ForceOverwrite Optional **Boolean**. Specifies whether to force a file overwrite when a file or folder is found with the same name. Set the argument to **True** to force a file overwrite. Default value is **False**.

The **Copy** method only copies files or folders within the same Web site. You cannot copy across Web sites. However, you can use the **SaveAs** method for the **PageWindowEx** object to save a page that has its file currently located in one Web site to save the page to a file in another Web site. Or, you can use the **Add** method for the **WebFile** object to add a file that is currently located in one Web site to another Web site.
This example copies a file from the Coho Winery folder to an Inventory folder within the same Web site.

Note To run this example, you must have a Web site called "C:\My Documents\My Web Sites\Coho Winery." Or, you may substitute an alternative Web site and file name.

```
Private Sub FileCopy()
   Dim myFile As WebFile
   Set myFile = ActiveWeb.RootFolder.Files("Zinfandel.htm")
   myFile.Copy "C:\My Web Sites\Coho Winery\Inventory\Zinfandel.htm
End Sub
```

CreateDynamicTemplateState Method

Returns a **DynamicTemplateState** object that represents a set of region mappings to use when applying or updating a Dynamic Web Template.

expression.CreateDynamicTemplateState

expression Required. An expression that returns an <u>Application</u> object.

The following example creates a reference to a Dynamic Web Template.

Dim objState As DynamicTemplateState

Set objState = Application.CreateDynamicTemplateState

CreatePackage Method

Returns a <u>WebPackage</u> object that represents a collection of pages, files, and folders and their related dependencies, such as images, cascading style sheets, and JavaScript files.

expression.CreatePackage(Title)

expression Required. An expression that returns a <u>WebEx</u> object.

Title Required **String**. The name of the Web package. This value becomes the value of the **Title** property for the **WebPackage** object.

Remarks

Use the **CreatePackage** method to create the **WebPackage** object. Use the <u>Add</u> method to add pages to the Web package. Then use the <u>Save</u> method to save the new Web package to disk. Use the <u>Remove</u> method to remove files that were added by using the **Add** method.

You can create Web packages from files in Web sites based on Microsoft Windows SharePoint Services and in disk-based Web sites.

The following example creates a new Web package and adds the page "test.htm" to the package, including all dependencies for the page, and then saves the new Web package.

```
Dim objWeb As WebEx
Dim objPackage As WebPackage
Set objWeb = ActiveWeb
Set objPackage = objWeb.CreatePackage("New Web Package")
With objPackage
    .Author = "John Smith"
    .Company = "Fourth Coffee"
    .Subject = "This is a new Web package for Fourth Coffee."
    .Add objWeb.Url & "/test.htm", fpDepsDefault
    .Save "c:\NewWebPackage.fwp", True
End With
```

CreateSearchInfo Method

Returns a <u>SearchInfo</u> object that represents a custom find or find and replace operation.

expression.CreateSearchInfo

expression Required. An expression that returns an <u>Application</u> object.

The following example finds the next occurrence of the P element in the active document.

Dim objSearch As SearchInfo Dim blnFound As Boolean Dim objRange As IHTMLTxtRange

Set objSearch = Application.CreateSearchInfo
objSearch.Find = "p"
objSearch.Action = fpSearchFindTag

Set objRange = Application.ActiveDocument.selection.createRange blnFound = Application.ActiveDocument.Find(objSearch, Nothing, objRa If blnFound = True Then objRange.Select

DecodeURL Method

Returns a **String** that represents a decoded Web address for the specified encoded Web address.

expression.DecodeURL(bstrEncodedURL)

expression Required. An expression that returns an <u>Application</u> object.

bstrEncodedURL Required **String**. The encoded Web address to decode.

Remarks

Decoding a URL includes replacing "%20" with spaces. Use the **EncodeURL** method to encode a Web address.

The following example decodes the specified Web address.

Dim strDecodedURL As String

```
strDecodedURL = Application.DecodeURL _
    ("http://www.fourthcoffee.com/our%20best%20coffee.htm")
```



Delete Method

Delete method as it applies to the NavigationNodes object.

Deletes an individual navigation node from the list of available nodes in the **NavigationNodes** collection.

expression.Delete(Index)

expression Required. An expression that returns a **NavigationNodes** object.

Index Optional **Variant**. Refers to an item in the navigation structure. Can be any number corresponding to an item in the navigation structure, with the index starting at zero.

Delete method as it applies to the **Properties** object.

Deletes a property from the list of available properties in the **Properties** collection.

expression.Delete(PropertyKey)

expression Required. An expression that returns a **Properties** object.

PropertyKey Required **String**. A string that represents the property name.

Delete method as it applies to the **WebEx** object.

Deletes a Web site from the list of available Web sites in the **Webs** collection.

expression.Delete(WebDeleteFlags)

expression Required. An expression that returns one of the above objects.

WebDeleteFlags Optional <u>FpWebDeleteFlags</u>. Determines what is deleted from the current Web site. Default is **fpDeleteEntireWeb**.

FpWebDeleteFlags can be one of these FpWebDeleteFlags constants. fpDeleteEntireWeb *default* fpDeleteFrontPageInfoFromWeb

Delete method as it applies to the **WebFiles** and **WebFolders** objects.

Deletes a task from the list of available tasks in the **WebFiles** collection, or a folder or folders from the list of available folders in the **WebFolders** collection.

expression.Delete(Index)

expression Required. An expression that returns one of the above objects.

Index Required **Variant**. Refers to an item in the **WebFiles** or **WebFolders** collection. Can be any number corresponding to an item in the collection, with the index starting at zero.

Delete method as it applies to the Webs object.

Deletes a Web site from the list of available Web sites in the **Webs** collection.

expression.Delete(Index, WebDeleteFlags)

expression Required. An expression that returns a **Webs** object.

Index Required **Variant**. Refers to an item in the **Webs** collection. Can be any number corresponding to an item in the collection, with the index starting at zero.

WebDeleteFlags Optional **FpWebDeleteFlags**. Determines what is deleted from the current Web site. Default is **fpDeleteEntireWeb**.

FpWebDeleteFlags can be one of these FpWebDeleteFlags constants. fpDeleteEntireWeb *default* fpDeleteFrontPageInfoFromWeb

Delete method as it applies to all other objects in the Applies To list.

Deletes the specified object from a Web site.

expression.**Delete**

expression Required. An expression that returns one of the above objects.

As it applies to the **NavigationNodes** object.

This example deletes the fourth navigation node of the second file in the active Web site.

Note You must apply the navigation structure to the Web site in order for the changes to be applied to the Web site.

```
Private Sub DeleteNavNode()
   Dim myWeb As WebEx
   Dim myChildNodes As NavigationNodes
   Dim intResponse As Integer
   Set myWeb = ActiveWeb
   Set myChildNodes = _
        myWeb.RootFolder.WebFiles(1).NavigationNode.Children
   intResponse = MsgBox("Are you sure you want to " & _
        "delete this navigation node?", vbYesNo)
   If intResponse = vbYes Then
        Call myChildNodes.Delete(3)
        myWeb.ApplyNavigationStructure
   End If
End Sub
```

As it applies to the **Properties** object.

This example deletes the SaleText property from the Sales.htm file.

```
Private Sub DeleteProperty()
   Dim myFile As WebFile
   Dim myProp As String
   Dim intResponse As Integer
   myProp = "SaleText"
   Set myFile = ActiveWeb.RootFolder.Files("Sales.htm")
   intResponse = MsgBox("Are you sure you want to delete the " & _
        myProp & " property?", vbYesNo)
```

```
If intrespons = vbYes Then
    myFile.Properties.Delete myProp
End If
End Sub
```

As it applies to the **WebEx** object.

This example deletes a temporary Web site called TempWeb.

```
Note To run this example, you must have a Web site called "C:\My Documents\My Web Sites\TempWeb". Or, you may substitute an alternative Web site URL.
```

```
Private Sub DeleteWeb()
    Dim myWeb As WebEx
    Dim myTempWeb As WebEx
    Dim myFolders As WebFolders
    Dim myFolder As WebFolder
    Dim myWebToDelete As String
    Dim intResponse As String
    Set myWeb = Webs.Open("C:\My Documents\My Webs")
    Set myFolders = myWeb.RootFolder.Folders
    myWebToDelete = "TempWeb"
    For Each myFolder In myFolders
        If myFolder.IsWeb = True Then
            If myFolder.Name = myWebToDelete Then
                intResponse = MsgBox("Are you sure you want to delet
                    "the " & myFolder.Name & " sub Web site?", vbYes
                If intResponse = vbYes Then
                    Set myTempWeb = Webs.Open(myFolder.Name)
                    myTempWeb.Delete
                End If
            End If
        End If
    Next
    ActiveWebWindow.Close
End Sub
```

As it applies to the **WebFiles** collection.

This statement deletes a file in the active Web site.

Note To run this example, you must have a file called "C:\My Documents\My Web Sites\TempFile.htm". Or, you may substitute an alternative file name.

```
Private Sub DeleteWebFile()
   Dim intResponse As Integer
   intResponse = MsgBox("Are you sure you want " & _
        "to delete this file?", vbYesNo)
   If intResponse = vbYes Then
        ActiveWeb.RootFolder.Files.Delete "TempFile"
   End If
End Sub
```



Edit Method

The **Edit** method is used to open Microsoft FrontPage compatible files in a page window. These files include file formats such as HTML, CSS, and ASP. To open files of other types, use the **Open** method.

Note HTML files without extensions will not open with the **Edit** method.

expression.Edit(ViewMode)

expression An expression that returns a **WebFile** object.

ViewMode Optional <u>FpPageViewMode</u>.

FpPageViewMode can be one of these FpPageViewMode constants. **fpPageViewDefault fpPageViewHtml fpPageViewNoFrames fpPageViewNormal fpPageViewNoWindow fpPageViewPreview**

This example shows how to use the **Edit** method to open a file for editing.

Note To run this program, you must have a Web site open that contains a file called "RedWines.htm." Or, you may substitute a file of your choice.

```
Private Sub ModifyFile()
   Dim myFile As WebFile
   Set myFile = ActiveWeb.RootFolder.Files("RedWines.htm")
   myFile.Edit
End Sub
```

EncodeURL Method

Returns a **String** that represents the encoded Web address for the specified Web address.

expression.EncodeURL(bstrDecodedURL)

expression Required. An expression that returns an <u>Application</u> object.

bstrDecodedURL Required **String**. The Web address to encode.

Remarks

Encoding a URL includes replacing spaces with "%20". Use the **DecodeURL** method to decode a Web address.

the following example encodes the specified Web address.

Dim strEncodedURL As String

```
strEncodedURL = Application.EncodeURL _
    ("http://www.fourthcoffee.com/our best coffee.htm")
```



ImportWebPackage Method

Returns an **<u>FpPkgImportResult</u>** that represents whether the specified package was imported or whether there were problems with the import.

FpPkgImportResult can be one of the following **FpPkgImportResult** constants.

fpPkgImportCancelled	Import was cancelled by the user.
fpPkgImportComplete	Import was completed successfully.
fpPkgImportErrorInPackage	Import was cancelled because an error in the package was found (for example, an invalid manifest or damaged files).
fpPkgImportFailed	Import failed for an unknown reason.
fpPkgImportNotTrusted	Import was cancelled because the package was not signed by a trusted certificate or the <i>FpPkgTrustLevel</i> parameter was not set to fpPkgTrustAll .
fpPkgImportServerNotSupported	Import was cancelled because the server on which the Web site is located does not support Web packages.
fpPkgImportStopped	Import was cancelled because a file, folder, or list conflict caused the import process to stop as a result of the value of the <i>fpConflictOpts</i> parameter

expression.**ImportWebPackage(packageFileName, urlDeployTo**, *FpPkgTrustLevel***,** *fpConflictOpts***)**

expression Required. An expression that returns one of the objects in the Applies To list.

packageFileName Required **String**. The path and file name of the package to be imported.

urlDeployTo Required **String**. The path to which to import the files included

in the Web package.

FpPkgTrustLevel Required <u>FpPkgTrustLevel</u>. Specifies whether to trust all Web packages or only those that are digitally signed by a trusted source.

FpPkgTrustLevel can be one of the following **FpPkgTrustLevel** constants.

fpPkgTrustAll fpPkgTrustCertificateStore

fpConflictOpts Required <u>FpPkgImportConflictOpts</u>. Specifies how to handle conflicts.

FpPkgImportConflictOpts can be one of the following **FpPkgImportConflictOpts** constants.

fpPkgFileConflictMask fpPkgListConflictMask

fpPkgOnConflictSkip	Skips importing any files if a conflict arises when importing a Web package, and continues the import process.
fpPkgOnConflictStop	Stops the import process if any conflict arises when importing a Web package, and continues the import process.
fpPkgOnFileConflictOverwrite	If a file conflict exists, overwrites that file.
fpPkgOnFileConflictSkip	Skips importing a file in the Web package that conflicts with a file in the Web site, and continues the import process.
fpPkgOnFileConflictStop	Stops the import process if a file if the Web package conflicts with an existing file in the Web site.
fpPkgOnListConflictMergeOrRename	Merges lists if they are compatible; otherwise, the Web lists are backed up and imported, and the import process continues.

fpPkgOnListConflictMergeOrSkip	Merges lists if compatible; otherwise, the import process skips the conflicting lists and continues.
fpPkgOnListConflictMergeOrStop	Merges lists if compatible; otherwise, the import process stops.
fpPkgOnListConflictRename	Backs up all conflicting lists and continues the import process.
fpPkgOnListConflictSkip	Skips deployment of a list if a list conflict exists.
fpPkgOnListConflictStop	Stops the import process of a Web package if a list conflict exists.

Remarks

You can import Web packages only into Web sites based on Microsoft Windows SharePoint Services.

The following example imports the specified Web package into a new folder in the active Web site.

```
Dim objWeb As WebEx
Dim objFolder As WebFolder
Set objWeb = ActiveWeb
Set objFolder = objWeb.AllFolders.Add("NewWebPackageFolder")
```

objWeb.ImportWebPackage "c:\NewWebPackage.fwp", objFolder.Url, _
fpPkgTrustCertificateStore, fpPkgOnListConflictSkip

LocateFile Method

Returns the specified **WebFile** object.

expression.LocateFile(FileUrl)

expression An expression that returns a **WebEx** object.

FileUrl Required **String**. Default value is the file portion of the URL.

This example locates a file in the root directory of the Web site and puts the file in edit mode.

Note You must have a Web site open and a file called "Zinfandel.htm," or you may substitute a file of your choice.

```
Private Sub LocateAFile()
   Dim myFile As WebFile
   Set myFile = Webs(0).LocateFile("Zinfandel.htm")
   myFile.Edit
End Sub
```

In most cases, you would probably use the entire URL for the *String* argument of the **LocateFile** method— for example, if you wanted to locate the file First_Qtr.htm in C:/My Documents/My Web Sites/Rogue Cellars/Inventory/First_Qtr.htm. Any time a folder exists in a level deeper than the root directory of the Web site, use the entire URL as shown in the following example.

```
Private Sub GetFile()
   Dim myFile As String
   Dim myFileFound As WebFile
   myFile = _______"C:/My Documents/My Web Sites/Rogue Cellars/Inventory/First_____
   Set myFileFound = Webs(0).LocateFile(myFile)
End Sub
```

However, there is a shortcut. For example, if you want to locate a file in an images folder that resides in the root directory of the Web site, you can use a relative address by using a forward slash followed by the subfolder and file name as shown in the following statement.

```
Set myFileFound = Webs(0).LocateFile("images/JPG/myJPGFileList.htm")
```

Note You cannot substitute a backslash in a relative address.

LocateFolder Method

Returns the specified **WebFolder** object.

expression.LocateFolder(FolderUrl)

expression An expression that returns a **WebFolder** object.

FolderUrl Required String. The default value is the folder portion of the URL.

This example locates a folder in the root directory of a Web site.

Note You must have a Web site open for all of these examples.

```
Private Sub LocateAFolder()
   Dim myFolderFound As WebFolder
   Set myFolderFound = Webs(0).LocateFolder("images")
End Sub
```

In most cases, you would probably use the entire URL for the *String* argument of the **LocateFolder** method. For example, a folder may be several levels deep in the folder hierarchy, such as C:/My Documents/My Web Sites/Rogue Cellars/Inventory/First_Quarter— and you want to locate First_Quarter. Any time a folder exists in a level deeper than the root directory of the Web site, use the entire URL as shown in the following example.

```
Private Sub GetFolder()
   Dim myFolder As String
   Dim myFolderFound As WebFolder
   myFolder = _______"C:/My Documents/My Web Sites/Rogue Cellars/Inventory/First_
   Set myFolderFound = _______Webs(0).LocateFolder(myFolder)
End Sub
```

However, there is a shortcut. For example, if you want to locate an images folder that resides in the root directory of the Web site, you can use a relative URL by using a forward slash followed by the subfolder name as shown in the following statement.

```
Set myFolderFound = Webs(0).LocateFolder("images/JPG")
```

Note You cannot substitute a backslash in a relative URL.


LocatePage Method

LocatePage method as it applies to the **Application** object.

Returns a **PageWindowEx** object for the specified object.

expression.LocatePage(DocumentUrl, ViewMode)

expression Required. An expression that returns an <u>Application</u> object.

DocumentUrl Required. A **String** that represents the document or file portion of the entire URL. This can be any absolute URL, such as "http://web server/file" or "file://file system/file" for disk-based Web sites. The default value is the file portion of the URL.

ViewMode Optional. An <u>FpPageViewMode</u> constant that represents the view mode.

FpPageViewMode can be one of these FpPageViewMode constants.
fpPageViewDefault default
fpPageViewHtml
fpPageViewNoFrames
fpPageViewNormal
fpPageViewNoWindow
fpPageViewPreview

LocatePage method as it applies to the WebEx object.

Returns the **PageWindowEx** object associated with the current Web site.

expression.LocatePage(FileUrl, ViewMode)

expression Required. An expression that returns a **WebEx** object.

FileUrl Required. A **String** that represents the document or file portion of the entire URL.

ViewMode Optional. An <u>FpPageViewMode</u> constant that represents the view mode.

FpPageViewMode can be one of these FpPageViewMode constants. **fpPageViewDefault** *default* **fpPageViewHtml fpPageViewNoFrames fpPageViewNormal fpPageViewNoWindow fpPageViewPreview**

This example locates a page in the root Web site and a subsite.

```
Private Sub LocatePages()
   Dim myRootPage As PageWindowEx
   Dim myWebPage As PageWindowEx
   Set myRootPage = ______Application.LocatePage("Zinfandel.htm", fpPageViewNormal)
   Set myWebPage = ______Webs(1).LocatePage("Zinfandel.htm", fpPageViewNormal)
End Sub
```

MakeAbs Method

Returns a **String** that represents an absolute URL for the **String** specified in the **URL** parameter, using the **URLBase** parameter as the starting point. If the URL is already absolute, the URL is returned unchanged. For more information about absolute and relative URLs, refer to <u>Understanding Absolute and Relative URL</u> <u>Addressing</u>.

expression.MakeAbs(UrlBase, Url)

expression An expression that returns a **Application** object.

UrlBase Required Variant. A base URL. Can be a string or a <u>WebEx</u>, <u>WebFolder</u>, <u>WebFile</u>, <u>NavigationNode</u>, or <u>IHTMLDocument2</u> object.

Url Required **String**. A string that contains the entire URL for the Web site. This can be any URL for a Web site, such as "http://web server/folder" or "file://file system/folder" for disk-based Web sites.

This example changes a relative URL to an absolute URL.

Note To run this example, you must have a Web site and a file called "C:\My Documents\My Web Sites\Rogue Cellars\Zinfandel.htm." Or, you may substitute an alternative Web site URL or file name.

```
Private Sub MakeURLAbsolute()
   Dim myBaseURL As WebEx
   Dim myAbsAddress As String
   Dim myLocalUrl As String
   myBaseURL = Webs.Open("C:\My Web Sites")
   myLocalUrl = "Zinfandel.htm"
   myAbsAddress = MakeAbs(myBaseURL, myLocalUrl)
End Sub
```

MakeRel Method

Returns a **String** that represents a relative URL for the **String** specified in the *URL* parameter, using the *URLBase* parameter as the starting point. If the URL is already relative to the *URLBase* parameter, the URL is returned unchanged. For more information about absolute and relative URLs, refer to <u>Understanding</u> <u>Absolute and Relative URL Addressing</u>.

expression.MakeRel(UrlBase, Url)

expression An expression that returns an <u>Application</u> object.

UrlBase Required Variant. A base URL. Can be a string or a <u>WebEx</u>, <u>WebFolder</u>, <u>WebFile</u>, <u>NavigationNode</u>, or <u>IHTMLDocument2</u> object.

Url Required **String**. A string that contains the entire URL. This can be any URL for a Web site, such as "http://web server/folder" or "file://file system/folder" for disk-based Web sites.

This example changes an absolute URL to a relative URL, adds a hyperlink to the active document using the relative URL, and then saves the changes to the document.

Note To run this example, you must have a Web site called "C:\My Documents\My Web Sites\Rogue Cellars." You must also have two files, one called "Zinfandel.htm" and the other called "index.htm," which has an absolute URL (the default state). Or, you may substitute an alternative Web site URL and file names.

```
Private Sub MakeURLRelative()
    Dim myFile As WebFile
    Dim myFile2 As WebFile
    Dim myBaseURL As WebEx
    Dim myDoc As FPHTMLDocument
    Dim myRelAddress As String
    Dim myRelAddress2 As String
    Set myBaseURL = Webs.Open("C:\My Documents\My Web Sites\Rogue Ce
    Set myFile = myBaseURL.RootFolder.Files("Zinfandel.htm")
    Set myFile2 = myBaseURL.RootFolder.Files("index.htm")
    Set myDoc = myFile.Edit(fpPageViewNormal).Document
   myRelAddress = MakeRel(myBaseURL, myFile2.Url)
    myRelAddress2 = """" & myRelAddress & """"
    Call myDoc.body.insertAdjacentHTML("BeforeEnd", "<a href=" _
            & myRelAddress2 & ">" & myRelAddress & "</a>")
    ActivePageWindow.Save
End Sub
```

MakeWeb Method

Creates a new Web site from an existing folder. To create a new Web site without first creating a folder, see the <u>Add</u> method.

Security Avoid using hard-coded passwords in your applications. If a password is required in a procedure, request the password from the user, store it in a variable, and then use the variable in your code. For recommended best practices on how to do this, see Security Notes for Microsoft Office Solution Developers.

expression.MakeWeb(UserName, Password)

expression An expression that returns a <u>WebFolder</u> object.

UserName Optional **String**. The logon name of the user. You can use this option to create a default user name for the Web site.

Password Optional **String**. The password of the user. You can use this option to create a default password for the Web site.

This example creates a new Web site from an existing folder named "Distributors, " which is a folder in the Rogue Cellars Web site. This example assumes that there is a Web site on your local computer named Rogue Cellars that contains a folder named Distributors. Alternatively, you can substitute a different Web site and folder name.

```
Private Sub MakeWeb()
   Dim myWeb As WebEx
   Dim myFolder As WebFolder
   Set myWeb = Webs("C:\My Web Sites\Rogue Cellars")
   myWeb.Activate
   Set myFolder = Active.RootFolder.Folders("Distributors")
   myFolder.MakeWeb
End Sub
```



Move Method (Web Object Model)

Move method as it applies to the WebFile and WebFolder objects.

Moves the specified object from its current location to a designated URL.

expression.Move(DestinationUrl, UpdateLinks, ForceOverwrite)

expression An expression that returns one of the above objects.

DestinationUrl Required **String**. The target URL, such as "C:\My Documents\My Web Sites\Adventure Works".

UpdateLinks Required **Boolean.True** to update links during the move process.

ForceOverwrite Required **Boolean**. **True** to overwrite duplicate files or folders.

Move method as it applies to the **NavigationNode** object.

Moves a navigation node from one location to another in the navigation structure. Returns a **NavigationNode** object that represents the node after it has been moved.

expression.Move(NodeCollection, NewLeftSibling)

expression An expression that returns a **NavigationNode** object.

NodeCollection Required **NavigationNodes**. The target navigation collection.

NewLeftSibling Optional Variant. The navigation node that will precede the new node in the navigation structure. If it is not specified, the node will become the last node in the target node collection specified in the *NodeCollection* parameter.

As it applies to the **WebFile** object.

The following statement moves a file from one position in the file structure to another.

```
myFile.Move("C:\My Documents\My Web Sites\Adventure Works\Images", _
True, False)
```

As it applies to the **NavigationNode** object.

The following example moves a node from the fifth position in the navigation structure to the fourth position in the navigation structure by designating the third node as the new left sibling.

```
Private Sub MoveNavNode()
   Dim myNodes As NavigationNodes
   Dim myNode As NavigationNode
   Set myNodes = ActiveWeb.RootNavigationNode.Children
   Set myNode = myNodes(4)
   myNode.Move(myNodes,2)
   ActiveWeb.ApplyNavigationStructure
```

End Sub

OnTime Method

Starts a background timer that runs a macro on the specified date at the specified time.

expression.OnTime(When, Name, Tolerance)

expression Required. An expression that returns an <u>Application</u> object.

When Required **Variant**. The time at which the macro is to be run. Can be a string that specifies a time (for example, "4:30 pm" or "16:30"), or it can be a serial number returned by a function such as **TimeValue** or **TimeSerial** (for example, TimeValue("2:30 pm") or TimeSerial(14, 30, 00)). You can also include the date (for example, "6/30 4:15 pm" or TimeValue("6/30 4:15 pm")).

Use the sum of the return values of the **Now** function and either the **TimeValue** or **TimeSerial** function to set a timer to run a macro a specified amount of time after the statement is run. For example, use Now+TimeValue("00:05:30") to run a macro 5 minutes and 30 seconds after the statement is run.

Name Required **String**. The name of the macro to be run. Use the complete macro path to ensure that the correct macro is run (for example, "Project.Module1.Macro1"). For the macro to run, the document or template must be available both when the **OnTime** method is run and when the time specified by *When* arrives.

Tolerance Optional **Variant**. The maximum time (in seconds) that can elapse before a macro that wasn't run at the time specified by **When** is canceled. Macros may not always run at the specified time. For example, if a dialog box is being displayed, the macro will be delayed until Microsoft FrontPage has completed the task. If this argument is 0 (zero) or omitted, the macro is run regardless of how much time has elapsed since the time specified by **When**.

Remarks

Microsoft FrontPage can maintain only one background timer set by the **OnTime** method. If you start another timer before an existing timer runs, the existing timer is canceled.

This example runs the macro named "Macro1" in the current module at 3:55 P.M.

Application.OnTime When:=Timevalue("15:55:00"), Name:="Macro1"

This example runs the macro named "Macro1" 15 seconds from the time the example is run. The macro name includes the project and module name.

```
Application.OnTime When:=Now + TimeValue("00:00:15"), _
    Name:="Project1.Module1.Macro1"
```

This example runs the macro named "Start" at 1:30 P.M. The macro name includes the project and module name.



Open Method

Open method as it applies to the **WebFile** object.

Opens a file in a Web site.

expression.Open

expression Required. An expression that returns a **WebFile** object.

Open method as it applies to the **Webs** object.

Opens a Web site. Returns a **WebEx** object.

expression.**Open**(*szWebUrl*, *UserName*, *Password*, *WebOpenFlags*)

expression Required. An expression that returns a **Webs** object.

szWebUrl Required **String**. The base URL of the Web site, such as "C:\My Web Sites". This can be any absolute URL, such as "http://web server " or "file://file system " for disk-based Web sites.

UserName Optional **String**. The logon name of the user.

Password Optional **String**. A designated string of characters to validate access to the specified Web site.

WebOpenFlags Optional <u>FpWebOpenFlags</u>.

Note Avoid using hard-coded passwords in your applications. If a password is required in a procedure, request the password from the user, store it in a variable, and then use the variable in your code. For recommended best practices on how to do this, see Security Notes for Microsoft Office Solution Developers.

FpWebOpenFlags can be one of these FpWebOpenFlags constants. fpOpenInWindow *default* fpOpenNoWindow

The following example opens the Rogue Cellars Web site and the Oktoberfest Sale file, and performs the following tasks:

- Adds text to the file by creating a property to hold the text.
- Accesses the **Page object model** using the **ActiveDocument** property and the **insertAdjacentText** method.
- Adds text to the page by substituting mySaleProp for the text parameter in the **insertAdjacentText** method.
- Closes Microsoft FrontPage.

Note To run this example, you must have a Web site called "C:\My Documents\My Web Sites\Rogue Cellars", or you may substitute an alternative Web site URL and file name.

```
Private Sub AddSaleText()
Dim objWeb As Web
Dim objFile As WebFile
Dim strSaleProp As String
Dim strSaleText As String
strSaleText = "Vintage Wines for Oktoberfest Sale!!!"
Set objWeb = Webs.Open("C:\My Documents\My Web Sites\Rogue Cella
Set objFile = ActiveWeb.RootFolder.Files("Sale.htm")
objFile.Properties.Add "SaleText", mySaleText
strSaleProp = objFile.Properties("SaleText")
objFile.Open
ActiveDocument.body.insertAdjacentText "BeforeEnd", strSaleProp
WebWindows.Close
```

End Sub



Publish Method

Publishes a Web site to a Web server.

expression.Publish(DestinationUrl, PublishFlags, UserName, Password)

expression An expression that returns a **WebEx** object.

DestinationUrl Required **String**. A string that contains the entire target URL for the Web site, such as "http://wwwroot/Adventure Works". This can be any URL for a Web site, such as http://web server /folder or file://file system /folder for disk-based Web sites.

PublishFlags Optional <u>FpWebPublishFlags</u>.

FpWebPublishFlags can be one of these FpWebPublishFlags constants. **fpPublishAddToExistingWeb fpPublishCopyAllFiles fpPublishCopySubwebs fpPublishIncremental fpPublishLogInTempDir fpPublishNoDeleteUnmatched fpPublishUseLastPublishTime fpPublishNone** *default*

UserName Optional **String**. The name of the user who is publishing the Web site.

Password Optional **String**. The password of the user.

Note Avoid using hard-coded passwords in your applications. If a password is required in a procedure, request the password from the user, store it in a variable, and then use the variable in your code. For recommended best practices on how to do this, see Security Notes for Microsoft Office Solution Developers.

The following example publishes the active Web site.

Note If the Web site you are publishing to is an existing Web site, you must use the argument **fpPublishAddToExistingWeb**, otherwise your Web site won't be published. If the Web site you are publishing to doesn't exist, don't use the **fpPublishAddToExistingWeb** argument because your Web site won't be published.

```
Private Sub PublishMyWeb()
   Dim myWeb As WebEx
   Dim myBaseURL As String
   Dim myPublishParam As FpWebPublishFlags
   Set myWeb = Application.ActiveWeb
   myBaseURL = "http://www.Adventure-Works.com"
   myPublishParam = fpPublishAddToExistingWeb
   myWeb.Publish myBaseURL, myPublishParam
End Sub
```

Quit Method

Quits the application. This method does not save any changes that have not been previously saved using the <u>Save</u> or <u>SaveAs</u> command, but immediately exits the active application.

expression.Quit

expression An expression that returns an **Application** object.

This example quits the application without saving any changes that were made since the previous <u>Save</u> or <u>SaveAs</u> command was executed.

Private Sub QuitApp() Application.**Quit** End Sub

RecalcHyperlinks Method

Recalculates all meta data on the server for the specified Web site. This operation will rebuild all hyperlinks, titles for Web pages, themes on pages, and so on in a Microsoft FrontPage Web site.

Note This operation may take a long time to complete depending on the amount of meta data on the server.

expression.RecalcHyperlinks

expression An expression that returns a **WebEx** object.

The following example recalculates the hyperlinks for the active Web site.

```
Private Sub RecalcLinks()
   Dim myWeb As WebEx
   Set myWeb = Application.ActiveWeb
   myWeb.RecalcHyperlinks
End Sub
```



Refresh Method (Web Object Model)

Refresh method as it applies to the **WebEx** object.

Refreshes the specified **WebEx** object.

expression.Refresh(FetchAll)

expression Required. An expression that returns a **WebEx** object.

FetchAll Optional **Boolean**. **True** to retrieve all information for all documents regardless of view. When set to **False** this argument retrieves only the information necessary to support the current view. However, if the **Reports** or **Hyperlink** view is open, **False** refreshes all documents. Default is **False**.

Refresh method as it applies to the **PageWindowEx** object.

Refreshes the specified page with an option to save changes.

expression.Refresh(SaveChanges)

expression Required. An expression that returns a **PageWindowEx** object.

SaveChanges Optional **Boolean**. **True** to save changes.

The following statements refresh the active page and the first page of the first Web site opened.

ActivePageWindow.Refresh
WebWindows(0).PageWindows(0).Refresh

You can use the following statement to save any changes you may have made to the active page.

ActivePageWindow.Refresh(True)

You can use the following statements to refresh the active Web site and the first Web site opened.

ActiveWeb.**Refresh** Webs(0).**Refresh**



Remove Method (Web Object Model)

Returns a **Boolean** that represents whether a specified file was successfully removed from a Web package.

expression.Remove(Url, flags)

expression Required. An expression that returns one of the objects in the Applies To list.

Url Required **String**. The path and file name of the file to remove from the Web package.

flags Required <u>FpDependencyFlags</u>. Specifies which dependencies to include when removing the Web package.

FpDependencyFlags can be a combination of one or more of the following **FpDependencyFlags** constants.

fpDepsDefault	Removes all images, link bars, hyperlinks, lists, shared borders, and themes.
fpDepsImages	Removes all images.
fpDepsLinkbars	Removes all link bars.
fpDepsLinks	Removes all pages to which there are hyperlinks.
fpDepsLists	Removes lists that may be needed in order for the page to render correctly.
fpDepsNone	Removes no dependencies.
fpDepsRecurse	Removes all files that are in a specified folder.
fpDepsSharedBorders	Removes all shared borders.
fpDepsThemes	Removes all themes.

The following example creates a Web package and adds three files to it, removes the last file added to the package, and then saves the package to the local drive.

```
Set objWeb = ActiveWeb
Set objPackage = objWeb.CreatePackage("New Web Package")
With objPackage
    .Add objWeb.Url & "/test.htm", fpDepsDefault
    .Add objWeb.Url & "/test2.htm", fpDepsNone
    .Add objWeb.Url & "/test3.htm", fpDepsImages
    .Remove objWeb.Url & "/test3.htm", fpDepsImages
    .Author = "John Smith"
    .Company = "Fourth Coffee"
    .Subject = "This is a new Web package for Fourth Coffee."
    .Save "c:\NewWebPackage.fwp", True
End With
```

RemoveChoice Method

Removes the specified choice from the specified field.

expression.RemoveChoice(Index)

expression Required. An expression that returns one of the objects in the Applies To list.

Index Required. A **Long** that represents the position of the choice within the array.

The following example removes the first choice from the NewChoiceField field in the first list of the active Web site.

```
Sub RemoveChoice()
'Removes first choice from array
Dim objApp As FrontPage.Application
Dim objListFields As listFields
Dim objListField As ListFieldChoice
Set objApp = FrontPage.Application
Set objListFields = objApp.ActiveWeb.Lists.Item(0).Fields
Set objListField = objListFields.Item("NewChoiceField")
'Remove first choice in list
objListField.RemoveChoice Index:=0
```

End Sub

RemoveWeb Method

Removes a Web site.

expression.RemoveWeb(UserName, Password)

expression An expression that returns a **WebFolder** object.

UserName Optional **String**. The logon name of the user.

Password Optional **String**. The password of the user.

Note Avoid using hard-coded passwords in your applications. If a password is required in a procedure, request the password from the user, store it in a variable, and then use the variable in your code. For recommended best practices on how to do this, see Security Notes for Microsoft Office Solution Developers.
Remarks

The **RemoveWeb** method is the complement of the <u>MakeWeb</u> method. Just as the **MakeWeb** method creates the meta data for the Web site from a folder, the **RemoveWeb** method removes the meta data for the Web site from a folder, but the folder remains intact. This is different from the <u>Delete</u> method for the **WebEx** object, where the entire contents of the specified Web site are removed.

The following example removes a Web site from a folder. The folder and its contents remain intact, but the folder is no longer a Web site.

Note You must have the Web site that contains the folder open.

```
Private Sub WebRemove()
   Dim myFolders As WebFolders
   Dim myFolder As WebFolder
   Set myWebFolders = Webs(0).RootFolder.Folders
   For Each myFolder In myFolders
        If myFolder.Name = "TempWeb" Then
            myFolder.RemoveWeb
            Exit For
        End If
        Next
End Sub
```

Run Method

Runs the designated Microsoft Visual Basic macro. You can use the **Run** method to execute a specified procedure in Microsoft FrontPage. You can also use the **Run** method from within the procedure of an ActiveX control that carries out instructions to query or modify a FrontPage-based Web site.

Note You cannot pass parameters to a procedure using the **Run** method. Use the **Call** statement to pass parameters to a procedure.

expression.Run(MacroName, safeArrayOfParams)

expression An expression that returns an **Application** object.

MacroName Required **String**. The name of the macro, add-in, or script.

safeArrayOfParams Required. A *ParamArray* of type Variant.

The following example runs a macro from another procedure.

Note To run this example, you must have a Web site named Rogue Cellars or you can substitute a different Web site in place of the Rogue Cellars Web site. Copy the following procedures into a code module and run StartMacro.

```
Private Sub StartMacro()
    Dim myMacro As String
    myMacro = "OpenRogueCellars"
    Run (myMacro)
End Sub
Sub OpenRogueCellars()
    Dim myWeb As Web
    Set myWeb = Webs.Open("C:\My Web Sites\Rogue Cellars")
End Sub
```



Save Method

As it applies to the **WebPackage** object.

Saves a Web package with the specified file name.

expression.Save(strFileName, fOverWrite)

expression Required. An expression that returns a **WebPackage** object.

strFileName Required **String**. The path and file name of the Web package. Web packages have an .fwp file name extension. The **Save** method does not automatically include this file name extension, so you should specify it as part of the file name.

fOverWrite Required Long. False to not overwrite an existing file with the same file name.

As it applies to the **PageWindowEx** object.

Saves a specified page.

expression.Save(ForceOverwrite)

expression An expression that returns a **PageWindowEx** object.

ForceOverwrite Optional **Boolean**. **False** to not save over an existing file. The default value is **True**.

As it applies to the **WebPackage** object.

The following example creates a new Web package and adds the page "test.htm" to the package, including all dependencies for the page, and then saves the new Web package.

```
Dim objWeb As WebEx
Dim objPackage As WebPackage
Set objWeb = ActiveWeb
Set objPackage = objWeb.CreatePackage("New Web Package")
With objPackage
    .Author = "John Smith"
    .Company = "Fourth Coffee"
    .Subject = "This is a new Web package for Fourth Coffee."
    .Add objWeb.Url & "/test.htm", fpDepsDefault
    .Save "C:\My Documents\NewWebPackage.fwp", True
End With
```

As it applies to the **PageWindowEx** object.

The following example creates a property, adds it to a file, and then saves the page.

Note To run this example, you must have a Web site called "C:\My Documents\My Web Sites\Rogue Cellars", or you may substitute an alternative Web site URL and file name.

```
Private Sub AddSaleText()
   Dim myWeb As WebEx
   Dim myFile As WebFile
   Dim mySaleProp As String
   Dim mySaleText As String
   mySaleText = "Vintage Wines for Oktoberfest Sale!!!"
   Set myWeb = Webs.Open("C:\My Documents\My Web Sites\Rogue Cellar
   Set myFile = ActiveWeb.RootFolder.Files("Sale.htm")
   myFile.Properties.Add "SaleText", mySaleText
   mySaleProp = myFile.Properties("SaleText")
```

```
myFile.Open
ActiveDocument.body.insertAdjacentText "BeforeEnd", mySaleProp
ActivePageWindow.Save
WebWindows.Close
Sub
```

End Sub

SaveAs Method

Writes the specified page object to the destination URL.

expression.SaveAs(DestinationUrl, ForceOverwrite)

expression An expression that returns a **PageWindowEx** object.

DestinationUrl Required **String**. A string that contains the entire URL for the Web site, such as "C:\My Documents\My Web Sites\Adventure Works\index.htm". This can be any URL for a Web site, such as http://web server /folder /file or file://file system /folder /file for disk-based Web sites.

ForceOverwrite Optional **Boolean**. **False** to not save over an existing file. The default value is **True**.

The following example saves an existing file to another Web site under a new name. (It isn't necessary to change the name of the file.) The program first activates the container Web site, and then it opens the file and saves it to a different Web site with a new name.

Note You must have a file named Zinfandel.htm in the C:\My Web Sites folder, or change the name of the file in the program to match an existing file in your Web site.

```
Private Sub SaveAsNewFile()
   Dim myFile As WebFile
   Dim myPageWindow As PageWindowEx
   Webs("C:\My Web Sites").Activate
   Set myFile = ActiveWeb.RootFolder.Files("Zinfandel.htm")
   myFile.Open
   Set myPageWindow = ActivePageWindow
   myPageWindow.SaveAs ("C:\My Web Sites\Rogue Cellars\Zinfandel Sa
   myPageWindow.Close
End Sub
```



SaveReport Method

Saves a specified report to an HTML file on the user's computer.

expression.SaveReport(reportviewModeEx, DestinationUrl, Title, ForceOverwrite)

expression Required. An expression that returns a **WebWindowEx** object.

reportviewModeEx Required. An <u>**FpWebViewModeEx**</u> constant that represents the type of report you want to save.

FpWebViewModeEx can be one of these FpWebViewModeEx constants. **fpWebViewExAllFiles** fpWebViewExAssignedTo fpWebViewExBrokenLinks **fpWebViewExBrowserTypes fpWebViewExCategories** fpWebViewExCheckoutStatus fpWebViewExComponentErrors fpWebViewExDailyPageHits **fpWebViewExDailySummary fpWebViewExFolders** fpWebViewExLinks fpWebViewExMonthlyPageHits fpWebViewExMonthlySummary **fpWebViewExNavigation fpWebViewExOlderFiles fpWebViewExOsTypes fpWebViewExPage** fpWebViewExPublishStatus **fpWebViewExRecentlyAddedFiles** fpWebViewExRecentlyChangedFiles

fpWebViewExReferringDomainsfpWebViewExReferringURLsfpWebViewExReviewStatusfpWebViewExReviewStatusfpWebViewExSearchStringsfpWebViewExSiteSummaryfpWebViewExSlowPagesfpWebViewExTodofpWebViewExUnlinkedFilesfpWebViewExUsageSummaryfpWebViewExVisitingUsersfpWebViewExWeeklyPageHitsfpWebViewExWeeklySummary

DestinationUrl Required. A **String** that represents the target file name for the report.

Title Required. A String that represents the title of the new report.

ForceOverwrite Optional. A **Boolean** that indicates if any previously created report with the same file name will be overwritten by the new report. If **True**, an existing file will be overwritten. The default value is **True**.

The following example saves a report of all files in the current Web site to a file named Report1.htm on the local user's computer. The report will overwrite any existing report in the \Reports directory with the name "Report1.htm".

```
Sub ReportSave()
'Saves a specified report to a specified location.
Dim objApp As FrontPage.Application
Dim objWebwdw As WebWindowEx
Set objApp = FrontPage.Application
Set objWebwdw = objApp.ActiveWebWindow
'Save the report
objWebwdw.SaveReport reportviewModeEx:=fpWebViewExAllFiles, _
Title:="Current Project Progress", _
DestinationURL:="C:\NewProject\Reports\Report1.htm", _
ForceOverwrite:=True
```

End Sub

SetChoices Method

Set the choices for a field of type choice to the items indicated in the *ppsaChoices* argument.

expression.SetChoices(ppsaChoices)

expression Required. An expression that returns one of the objects in the Applies To list.

ppsaChoices Required **String**. An array of items that represent the choices in the list field.

ShowBordersShadingDialog Method

Displays the **Borders and Shading** dialog box and returns a **String** that represents the Cascading Style Sheet property settings for the borders and shading properties selected in the **Borders and Shading** dialog box. A value is returned when the users clicks **OK**. If the user clicks **Cancel**, an empty **String** is returned.

expression.ShowBordersShadingDialog(strCSSIn)

expression Required. An expression that returns one of the objects in the Applies To list.

strCSSIn Optional **Variant**. A **String** that represents the initial custom settings for the dialog box.

The following example displays the **Borders and Shading** dialog box, and then sets the border style for the active element to the border settings returned.

```
Dim strCSS As String
Dim strCSSIn As String
strCSSIn = "border: 3 double #00FFFF"
strCSS = Application.ShowBordersShadingDialog(strCSSIn)
If strCSS <> "" Then ActiveDocument.activeElement _
.Style.Border = strCSS
```

ShowFontDialog Method

Displays the Font dialog box and returns a **String** that represents the Cascading Style Sheet properties for the font properties selected in the **Font** dialog box. A value is returned when the users clicks **OK**. If the user clicks **Cancel**, an empty **String** is returned.

expression.ShowFontDialog(strCSSIn)

expression Required. An expression that returns one of the objects in the Applies To list.

strCSSIn Optional **Variant**. A **String** that represents the initial custom settings for the dialog box.

The following example displays the **Font** dialog box, and then sets the font style for the active element to the font settings returned.

```
Dim strCSS As String
Dim strCSSIn As String
strCSSIn = "font-size:14pt; color:#FF0000"
strCSS = Application.ShowFontDialog(strCSSIn)
If strCSS <> "" Then ActiveDocument _
.activeElement.Style.Font = strCSS
```

ShowHTMLDialog Method

Displays a custom dialog box by using the contents of an HTML page. Returns a **Variant**.

expression.ShowHTMLDialog(Url, pVarArgIn)

expression Required. An expression that returns one of the objects in the Applies To list.

Url Required **String**. The path and file name of the page to render as a dialog box.

pVarArgIn Optional Variant. Data used to set the initial settings of the custom dialog box.

The following example displays the specified Web page in a Microsoft Windows dialog box.

Application.ShowHTMLDialog("c:\test.htm")

ShowHyperlinkParameters Method

Displays the **Hyperlink Parameter** dialog box and returns a **String** that represents the hyperlink plus the parameters, separated by a question mark.

expression.ShowHyperlinkParameters(bstrPath, bstrQuery, bstrColumns, bstrColTypes)

expression Required. An expression that returns one of the objects in the Applies To list.

bstrPath Required *String*. Specifies the Web address for the hyperlink.

bstrQuery Required **String**. Specifies the query string to pass in when opening the linked page.

bstrColumns Required **String**. Specifies a comma-delimited **String** that contains the names of the database columns that are available for use in the URL.

bstrColTypes Required **String**. Specifies a comma-delimited **String** that represents the data type values of the database columns.

Remarks

The values for the *bstrColTypes* parameter include the following.

Data type	Value	Description
adArray	819	When combined with another data type, indicates an array of the other data type.
adBigInt	20	Indicates an 8-byte signed integer.
adBinary	128	Indicates a binary value.
adBoolean	11	Indicates a Boolean value.
adBSTR	8	Indicates a null-terminated character string.
adChapter	136	Indicates a 4-byte chapter value that identifies rows in a child row set.
adChar	129	Indicates a string value.
adCurrency	6	Indicates a currency value. Currency is a fixed-point number with four digits to the right of the decimal point and is stored in an eight-byte signed integer.
adDate	7	Indicates a date value. A date value is stored as a double, the whole part of which is the number of days since December 30, 1899, and the fractional part of which is the fraction of a day.
adDBDate	133	Indicates a date value (yyyymmdd).
adDBTime	134	Indicates a time value (hhmmss).
adDBTimeStamp	135	Indicates a date/time stamp (yyyymmddhhmmss plus a fraction in billionths).
adDecimal	14	Indicates an exact numeric value with a fixed precision and scale.
adDouble	5	Indicates a double-precision floating-point value.
adEmpty	0	Indicates no value.
adError	10	Indicates a 32-bit error code.
adFileTime	64	Indicates a 64-bit value representing the number of 100-nanosecond intervals since January 1, 1601.
adGUID	72	Indicates a globally unique identifier (GUID).

adIDispatch	9	Indicates a pointer to an IDispatch interface on a COM object. (ADO does not currently support this data type. Usage may cause unpredictable results.)
adInteger	3	Indicates a 4-byte signed integer.
adIUnknown	13	Indicates a pointer to an IUknown interface on a COM object. (ADO does not currently support this data type. Usage may cause unpredictable results.)
adLongVarBinary	205	Indicates a long binary value.
adLongVarChar	201	Indicates a long string value.
adLongVarWChar	203	Indicates a long null-terminated Unicode value.
adNumeric	131	Indicates an exact numeric value with a fixed precision and scale.
adPropVariant	138	Indicates an Automation PROPVARIANT.
adSingle	4	Indicates a single-precision floating point.
adSmallInt	2	Indicates a 2-byte signed integer.
adTinyInt	16	Indicates a 1-byte signed integer.
adUnsignedBigInt	21	Indicates an 8-byte unsigned integer.
adUnsignedInt	19	Indicates a 4-byte unsigned integer.
adUnsignedSmallInt	18	Indicates a 2-byte unsigned integer.
adUnsignedTinyInt	17	Indicates a 1-byte unsigned integer.
adUserDefined	132	Indicates a user-defined variable.
adVarBinary	204	Indicates a binary value.
adVarChar	200	Indicates a string value.
adVariant	12	Indicates an Automation Variant. (ADO does not currently support this data type. Usage may cause unpredictable results.)
adVarNumeric	139	Indicates a numeric value.
adVarWChar	202	Indicates a null-terminated Unicode character string.
adWChar	130	Indicates a null-terminated Unicode character string.

For more information on ActiveX data types, see ActiveX Data Object (ADO) on the Microsoft Developer Network (MSDN) Web site.

The following example displays the **Hyperlink Parameter** dialog box with the following URL and settings.

Application.ShowHyperlinkParameters "http://www.fourthcoffee.com/cof "type=black", "CategoryID,CategoryName,Description", "3,202,202"

ShowImportWebPackageDialog Method

Returns a **Boolean** that indicates whether the method successfully added the specified Web package.

expression.ShowImportWebPackageDialog(packageFileName, urlImportedTo)

expression Required. An expression that returns an **Application** object.

packageFileName Required **String**. The path and file name of the Web package to import.

urlImportedTo Required **String**. The path and file name of where to import the Web package.

Remarks

You can import Web packages only into Web sites based on Microsoft Windows SharePoint Services.

The following example displays the **Import Web Package** dialog box, and then indicates whether the specified Web package was added successfully to the specified Web site.

```
Dim blnResponse As Boolean
blnResponse = Application.ShowImportWebPackageDialog _
    ("c:\NewWebPackage.fwp", ActiveWeb.Url)
If blnResponse = True Then
    MsgBox "The Web package was added successfully."
Else
    MsgBox "Unable to add the Web package to the specified site." _
    & vbCrLf & "Contact your administrator for assistance."
End If
```

ShowPickURLDialog Method

Displays the the **Edit Hyperlink** dialog box and returns a **String** that represents the URL to the file that the user has selected in the **Edit Hyperlink** dialog box.

expression.ShowPickURLDialog(strBaseURL, strFileURL)

expression Required. An expression that returns one of the objects in the Applies To list.

strBaseURL Optional **Variant**. The base URL for the resulting hyperlink.

strFileURL Optional **Variant**. The selected page for a new hyperlink. The user may change this by selecting a different page in the **Edit Hyperlink** dialog box.

The following example displays the the **Edit Hyperlink** dialog box.

```
Dim strURL As String
Dim objSelection As IHTMLTxtRange
Dim strHyperlink As String
strURL = Application.ShowPickURLDialog _
    (ActiveWeb.Url, "c:\test.htm")
Set objSelection = ActiveDocument.selection.createRange
With objSelection
    strHyperlink = "<a href=""" & strURL & """>" & .Text & "</a>"
    .pasteHTML strHyperlink
End With
```

ShowPositionDialog Method

Returns a **String** that represents the value of the **position** attribute for a cascading style sheet.

expression.ShowPositionDialog(strCSSIn)

expression Required. An expression that returns one of the objects in the Applies To list.

strCSSIn Optional **Variant**. A **String** that represents the initial custom settings for the dialog box.

The following example sets the **position** attribute for the active element in the active document. Note that using the **setAttribute** method overwrites any other style attribute settings for the active element.

```
Dim strCSS As String
Dim strCSSIn As String
strCSSIn = "position: absolute; left: 750"
strCSS = Application.ShowPositionDialog(strCSSIn)
If strCSS <> "" Then ActiveDocument.activeElement _
.setAttribute "style", strCSS
```

SplitArgs Method

Returns a **Variant** that represents an array of the individual items in the specified string separated by the specified character string.

expression.SplitArgs(String, Tokens)

expression Required. An expression that returns one of the objects in the Applies To list.

String Required *String*. The string containing the data.

Tokens Required **String**. The character string that separates the items in the *String* parameter.

Remarks

The **SplitArgs** method is similar to the **Split** function in Microsoft Visual Basic; however, the **SplitArgs** method will not break quote-delimited strings.

The following example splits the specified string into an array of the items in the specified string that are separated by a comma.

```
Dim varArray() As Variant
varArray = Application.SplitArgs("this, is a, test", ",")
```

UndoCheckout Method

Returns the specified file to its prior state in the source control project without saving any changes that may have been made.

Note You must have a source control project in place before using this method. For information about source control projects, refer to <u>Managing Source Control</u> <u>Projects</u>.

expression.UndoCheckout

expression An expression that returns a **WebFile** object.
The following example checks a file in to its prior state (its state before checkout). The file must be part of a source control project.

Note To run this procedure, you must have a source control project in place with a Web site open and a page called Zinfandel.htm contained in the Web site. Or, substitute an alternate Web site and file name.

```
Private Sub UndoCheckout()
   Dim myWeb As WebEx
   Set myWeb = ("C:/My Web Sites/Rogue Cellars")
   myWeb.RootFolder.Files("Zinfandel.htm").UndoCheckout
End Sub
```

UpdateDynamicTemplate Method

Updates the pages that are attached to a Dynamic Web Template so that any changes to the Dynamic Web Template are applied to pages that reference the Dynamic Web Template.

expression.UpdateDynamicTemplate(type, pbzLog)

expression Required. An expression that returns one of the objects in the Applies To list.

pState Required **DynamicTemplateState**. Specifies the region mapping to use when updating the Dynamic Web Template. Use the <u>SetHeadMapping</u> and <u>SetBodyMapping</u> methods of the <u>DynamicTemplateState</u> object to customize region mapping.

pbzLog Required **String**. A **ByRef** parameter that returns a **String** containing the log entry about the update process for a file. In the case of an error, it returns information that indicates which files have failed.

Remarks

If an error occurs, the update process will terminate unless the **<u>SkipOnQuery</u>** property is set to **True**.

The following example updates the Dynamic Web Template references in the specified file.

```
Dim objState As DynamicTemplateState
Dim objFile As WebFile
Dim strLog As String
Set objState = Application.CreateDynamicTemplateState
Set objFile = ActiveWeb.LocateFile("home.htm")
```

objFile.UpdateDynamicTemplate objState, strLog

VerifyAllLinks Method

Verifies all hyperlinks in the specified Web site.

Note The Broken Hyperlinks report uses the **VerifyAllLinks** method to display any pages with broken hyperlinks.

expression.VerifyAllLinks

expression Required. An expression that returns a **WebWindowEx** object

The following example uses the **VerifyAllLinks** method to check for broken links in the active Web site, and then switches to the Broken Hyperlinks report view.

```
Sub VerifyLinks()
'Verifies broken links in the current view
Dim objApp As FrontPage.Application
Dim objWebwdw As WebWindowEx
Set objApp = FrontPage.Application
Set objWebwdw = objApp.ActiveWebWindow
'Verify all links in the current web.
objWebwdw.VerifyAllLinks
```

End Sub

ActiveDocument Property

Returns an **FPHTMLDocument** object that represents the Web page currently displayed in the Microsoft FrontPage application window.

expression. ActiveDocument

expression Required. An expression that returns one of the objects in the Applies To list.

Remarks

This object provides access to the Microsoft FrontPage Page Object Model elements that are compatible with Microsoft Internet Explorer 4.0 and later. This object model provides programmatic access to the HTML in the specified page.

The following statement returns the active document in FrontPage.

myDoc = Application.ActiveDocument

The following statement returns the active document for the specified page window.

myDoc = Application.WebWindows(0).PageWindows(0).ActiveDocument

ActiveFrameWindow Property

Returns an **FPHTMLWindow2** object that represents the document displayed in the active page window.

Note If the active page contains frames, the **ActiveFrameWindow** property returns the currently selected frame, if one is selected, or the frames page itself, if none of the frames is selected. If a page does not contain frames, the **ActiveFrameWindow** property returns the window containing the specified page.

expression.ActiveFrameWindow

expression Required. An expression that returns a **<u>PageWindowEx</u>** object.

Note Accessing the **ActiveFrameWindow** property is the same as accessing the parent window through the active page window as follows:

ActivePageWindow.Document.parentWindow

The following example retrieves the **ActiveFrameWindow** object.

ActivePageWindow Property

Returns a **<u>PageWindowEx</u>** object that represents the window in which the current page is displayed.

expression.ActivePageWindow

expression Required. An expression that returns one of the objects in the Applies To list.

The following example returns the **PageWindowEx** object for the **Application** object.

```
Private Sub GetActivePageWindow()
    Dim myPageWindow As PageWindowEx
    Set myPageWindow = ActivePageWindow
End Sub
```

The following example returns the **PageWindowEx** object from the **WebWindowEx** object.

```
Private Sub GetActivePageWindow()
Dim myPage As PageWindowEx
Set myPage = ActiveWebWindow.ActivePageWindow
End Sub
```

ActiveWeb Property

Returns a <u>WebEx</u> object that represents the Web site currently open in Microsoft FrontPage.

expression.ActiveWeb

expression Required. An expression that returns an **Application** object.

Remarks

The Microsoft FrontPage Visual Basic for Applications object model provides access to the Web object model through the **ActiveWeb** property. For more information about the FrontPage object models, see <u>Exploring the Object Model</u> in FrontPage.

This example uses the **ActiveWeb** property to locate the Web page index.htm and changes the background color of the page.

Note To run this example, create a form with one command button called cmdActiveWebColorChange and paste the following code in the code window. You must have a Web page called index.htm for this procedure to change the background color.

```
Private Sub cmdActiveWebBKGRDColorChange_Click()
  Dim myPageWin As PageWindowEx
  Set myPageWin = Application.ActiveWeb.LocatePage("index.htm")
  myPageWin.Document.bgColor = "PapayaWhip"
```

End Sub

Note Microsoft Visual Basic provides color constants with a "vb" prefix, such as **vbCyan**. FrontPage builds Web pages in HTML, which uses different color constants than Visual Basic. If you use the color constants provided with Visual Basic in your existing programs, you may need to change these to the equivalent hexadecimal color values when you port your programs to a FrontPage-based Web site, or when you use colors across Microsoft Office applications that include a FrontPage-based Web site.

ActiveWebWindow Property

Returns a **WebWindowEx** object that represents the window in which the currently open Web site is displayed.

expression.ActiveWebWindow

expression Required. An expression that returns one of the objects in the Applies To list.

This example prints the value of the ActiveDocument.nameProp property for the **WebEx** object.

Note To run this example, create a form with a command button called "cmdActiveWebWindowDisplay" and a label called "lblWebWindowDisplay". Paste the following code in the code window. You must have a Microsoft FrontPage-based Web page open for this procedure to return a document name.

```
Private Sub cmdActiveWebWindowDisplay_Click()
On Error Resume Next
Dim myCurrentWebWindow As WebWindowEx
Set myCurrentWebWindow = Application.ActiveWebWindow
With myCurrentWebWindow
lblWebWindowDisplay.Caption = .ActiveDocument.nameProp
End With
End Sub
```

AllFiles Property

Returns a <u>WebFiles</u> collection that represents all files in the specified Web site.

expression.AllFiles

expression Required. An expression that returns one of the objects in the Applies To list.

Remarks

The **WebFiles** collection returns all files in the collection regardless of their position in the Web site hierarchy.

The following example searches through the files in the active Web site for a page with the title "Main Page." If the page is found, it is opened in Microsoft FrontPage.

```
Sub FindFileTitle()
                        'Returns a collection of all files in the cu
    Dim objApp As FrontPage.Application
    Dim objWebFile As WebFile
    Dim objWebFiles As WebFiles
    Set objApp = FrontPage.Application
    'Create a reference to the WebFiles collection.
    Set objWebFiles = objApp.ActiveWeb.AllFiles
    'Check each file in the collection for the title Main Page.
    For Each objWebFile In objWebFiles
        'If the title is found open the page in the editor.
        If objWebFile.Title = "Main Page" Then
            objWebFile.Open
        End If
        'If not found, check next file.
    Next objWebFile
End Sub
```

AllFolders Property

Returns a <u>WebFolders</u> collection that represents all folders in the current Web site.

expression.AllFolders

expression Required. An expression that returns one of the objects in the Applies To list.

Remarks

The **WebFolders** collection returns all folders in the collection regardless of their position in the Web site hierarchy.

The following example searches through the **WebFolders** collection for a folder named "Folder1." If the folder is found, the example searches for a file with the title "Main Page." If the title is found, the file is opened in Microsoft FrontPage.

```
Sub WebFoldersFind()
    Dim objApp As FrontPage.Application
    Dim objWebFolder As WebFolder
    Dim objWebFolders As WebFolders
    Set objApp = FrontPage.Application
    'Create a reference to the WebFolders collection.
    Set objWebFolders = objApp.ActiveWeb.AllFolders
    'Check each folder in the collection for the name "Folder1".
    For Each objWebFolder In objWebFolders
       'If the folder is found then search through each
       'file in the folder for a file with the title
       'Main Page and open the file if it exists.
       If objWebFolder.Name = "Folder1" Then
          For i = 1 To objWebFolder.Files.Count
              If objWebFolder.Files.Item(i).Title = "Main Page" Then
                  objWebFolder.Files.Item(i).Open
                  Exit For
              End If
          Next i
       End If
    'If not found check next file.
    Next objWebFolder
```

End Sub

AllNavigationNodes Property

Returns a **<u>NavigationNodes</u>** collection object that represents all of the navigation nodes in the specified Web site.

Note Navigation nodes are used to display a graphic representation of the current Web site in **Navigation** view.

expression.AllNavigationNodes

expression Required. An expression that returns a <u>WebEx</u> object.

The following example returns a reference to the **NavigationNodes** collection and displays the file name of the first object in the collection and the title of the Web site in which it exists.

```
Sub AllNavigationNodes()
   Dim objApp As FrontPage.Application
   Dim objNavNode As NavigationNode
   Dim objNavNodes As NavigationNodes
   Set objApp = FrontPage.Application
   'Create a reference to the NavigationNodes collection.
   Set objNavNodes = objApp.ActiveWeb.AllNavigationNodes
   'Return a reference to the first node in the collection.
   Set objNavNode = objNavNodes.Item(0)
   'Display the file name and the Web of the first
   'navigation node in the collection
   MsgBox "The URL of this file is " & objNavNode.Url & _
        vbCr & ". It is found in the " _
        & objNavNode.Web.Title & " Web site."
```

End Sub

AllowAttachments Property

Sets or returns a **Boolean** that represents whether a list allows allows attachments.

expression. Allow Attachments

expression Required. An expression that returns one of the objects in the Applies To list.

AllowFillInChoices Property

Sets or returns a **Boolean** that represents whether a choice field allows users to enter a custom choice.

expression.AllowFillInChoices

expression Required. An expression that returns a **ListFieldChoice** object.

AllowModerate Property

Sets or returns a **Boolean** that represents whether to allow using the moderation setting in a list.

expression.AllowModerate

expression Required. An expression that returns one of the objects in the Applies To list.

AllowMultipleResponses Property

Returns or sets a **Boolean** that determines whether users can respond more than once to a given survey. If **False**, a user can only respond once to a survey.

expression. Allow Multiple Responses

expression Required. An expression that returns a **Survey** object.

The following example sets the **AllowMultipleResponses** property of all **Survey** objects in the active Web site to **False** so that users can only respond once to a given survey.

AllowRichHTML Property

Sets or returns a **Boolean** that represents whether to display the Rich Text Format (RTF) edit control in the browser.

expression.AllowRichHTML

expression Required. An expression that returns a **ListFieldMultiLine** object.

AllowsLongFilenames Property

Returns **True** if the operating system on the machine where the Web site resides accepts long file names. Read-only **Boolean**.

expression.AllowsLongFilenames

expression Required. An expression that returns a **WebEx** object.

The following example checks whether the active Web site allows long file names. The example assumes the existence of a procedure named ConvertLongFilenames.

If ActiveWeb.AllowsLongFilenames = True Then
 Exit Sub
Else
 Call ConvertLongFilenames()
End If

AnswerWizard Property

Some of the content in this topic may not be applicable to some languages.

Returns an **AnswerWizard** object that contains the files used by the Help search engine.

expression.AnswerWizard

expression Required. An expression that returns an **Application** object.

This example adds an Answer Wizard file and retrieves the **Creator** and **Count** properties.

Note The **Creator** property for Microsoft FrontPage is different from the **Creator** property used by other Microsoft Office objects, such as the **AnswerWizard** object. The **Creator** property for FrontPage uses a string format, while Office objects use a 32-bit integer to identify the application that created the file. In this example, you'll note that the **Creator** property returns a 32-bit integer rather than the FrontPage string format because the **AnswerWizard** object is an Office shared object. For more information, use the Object Browser for all libraries or Microsoft Visual Basic Help to view more detailed information about the **Creator** property for Microsoft Office objects.

```
Private Sub GetAnswerWizardInfo()
   Dim myAW As AnswerWizard
   Dim myAWFiles As AnswerWizardFiles
   Dim myAWCount As Integer
   Dim myAWCreator As String
   Set myAW = ActiveWeb.Application.AnswerWizard
   Set myAW = myAW.Files
   With myAWFiles
        myAWCreator = .Creator
        .Add("myAWFile")
        myAWCount = .Count
   End With
End Sub
```
Application Property

Returns an <u>Application</u> object that represents the Microsoft FrontPage application.

expression. Application

expression Required. An expression that returns one of the objects in the Applies To list.

The application name can be directly accessed from the objects and collections in the FrontPage object model. This statement shows the FrontPage application accessed from the **Files** collection.

Set currApp = ActiveWeb.RootFolder.Files.Application

Assistant Property

Some of the content in this topic may not be applicable to some languages.

Returns an <u>Assistant</u> object that represents the Microsoft Office Assistant. Readonly.

expression.Assistant

expression Required. An expression that returns an <u>Application</u> object.

The following example returns a reference to the Microsoft Office Assistant and modifies its properties. The subroutine displays the Assistant on the screen, turns animation sounds on, and sets the **MoveWhenInTheWay** property to **True**. The Assistant will now move to another location on the screen if it is in the user's way.

```
Sub fpAsst()
'Creates a reference to the Office Assistant and modifies its proper
Dim objApp As FrontPage.Application
Dim objApst As Assistant
Set objApp = FrontPage.Application
'Return a reference to the Office Assistant
Set objAsst = objApp.Assistant
With objAsst
'Make assistant visible
.Visible = True
'Allow sounds during animations
.Sounds = True
'Move the Assistant out of the way when needed
.MoveWhenInTheWay = True
End With
```



Author Property

Sets or returns a **String** that represents the name of the author who created a Web package.

expression.Author

expression Required. An expression that returns a <u>WebPackage</u> object.

The following example creates a new Web package and adds the page "test.htm" to the package, including all dependencies for the page, and then saves the new Web package.

```
Dim objWeb As WebEx
Dim objPackage As WebPackage
Set objWeb = ActiveWeb
Set objPackage = objWeb.CreatePackage("New Web Package")
With objPackage
    .Author = "John Smith"
    .Company = "Fourth Coffee"
    .Subject = "This is a new Web package for Fourth Coffee."
    .Add objWeb.Url & "/test.htm", fpDepsDefault
    .Save "c:\NewWebPackage.fwp", True
End With
```

Build Property

Returns the build number for the specified object. Read-only **String**.

expression.Build()

expression Required. An expression that returns one of the objects in the Applies To list.

Remarks

If the **Application** object is specified, the **Build** property returns the build number of the application. When the **System** object is specified, this property returns the build number of the operating system. For example, the format used on the **About** dialog box for an application might look like this: "Version: 4.0.0.*xxxx*," where *xxxx* is the build number of the application. When the **Build** property for the **System** object is accessed, it returns the current build of the operating system as you might see used on the **Properties** page of the **System** dialog box.

The following statement returns the build number of the application.

myAppBuild = Application.Build

You can access the system build as shown in the following statement.

mySysbuild = System.Build

Caption Property

Returns a **String** that represents either the caption text in the title bar or the URL of a page.

expression.Caption

expression Required. An expression that returns one of the objects in the Applies To list.

Remarks

The **Caption** property returns different values depending on the object. For example, the **Caption** property for the **PageWindowEx** object returns the file URL of the open page, while the **Caption** property for the **WebWindowEx** object returns the text of the title bar for the Microsoft FrontPage application window.

This statement returns the caption of the active page.

```
myCaption = ActivePageWindow.Caption
```

The following example demonstrates accessing both the active **WebWindowEx** and **PageWindowEx** objects using the **With** and **For** statements.

```
Private Sub GetPageWindowCaption()
   Dim myWebWindow As WebWindowEx
   Dim myPageWindowS As PageWindows
   Dim myPageWindowCaptions As String
   Dim myWebWindowCaption As String
   Set myWebWindow = Application.ActiveWebWindow
   Set myPageWindows = myWebWindow.PageWindows
   With myWebWindow
   myWebWindowCaption = .Caption
   End With
   For Each myPageWindow In myPageWindows
        myPageWindowCaptions = myPageWindowCaptions & myPageWindow.C
   Next
End Sub
```

CheckedoutBy Property

Returns a **String** that represents the logon name of the person to whom the specified file is checked out.

Note You must have a source control project in place before using this method.

expression.CheckedoutBy

expression Required. An expression that returns a **WebFile** object.

This example retrieves the logon alias of the person who checked out the file.

Note You must have a source control project to run this example.

```
Private Sub GetCheckedOutBy()
   Dim myCheckedOutAlias As String
   myCheckedOutAlias = _______
        ActiveWeb.RootFolder.Files(0).CheckedoutBy
End Sub
```

Children Property

Returns a **NavigationNodes** collection that represents the collection of child nodes for a navigation node.

expression.Children

expression Required. An expression that returns a **NavigationNode** object.

The following example returns the number of child nodes that exist directly below the home page navigation node.

```
Private Sub GetChildrenCount()
Dim myCount As Integer
```

myCount = ActiveWeb.HomeNavigationNode.Children.Count
End Sub

Choices Property

Returns or sets a **String** that represents an array of all choices in the current field.

expression.Choices

expression Required. An expression that returns a **ListFieldChoice** object.

The following example displays the text for all choices in the field "NewChoiceField." If the field contains no choices, a message is displayed to the user. The field "NewChoiceField" is an object of type ListFieldChoice.

```
Sub ViewChoices()
'Displays the choices in the current field
    Dim objApp As FrontPage.Application
    Dim objLstFlds As ListFields
    Dim objFldChoice As ListFieldChoice
    Dim VarChoices As Variant
    Dim strChoice As String
    Dim blnFound As Boolean
    Set objApp = FrontPage.Application
    Set objLstFlds = objApp.ActiveWeb.Lists.Item(0).Fields
    'Reference choice field
    Set objFldChoice = objLstFlds.Item("NewChoiceField")
    blnFound = False
    For Each VarChoice In objFldChoice. Choices
        If strChoice = "" Then
            'if first value in string
            strChoice = VarChoice & vbCr
            'The list contains at least one choice
            blnFound = True
        Else
            'add value to string
            strChoice = strChoice & VarChoice & vbCr
        End If
    Next VarChoice
    If blnFound = True Then
        'Display choices
        MsgBox "The current list contains the following choices: " &
        vbCr & strChoice
    Else
        'Display message to user
        MsgBox "The current field contains no choices."
    End If
```

```
End Sub
```

COMAddIns Property

Returns a **COMAddIns** collection that represents all the Component Object Model (COM) add-ins currently loaded in Microsoft FrontPage. These are listed in the **COM Add-Ins** dialog box (**Add-Ins** command on the **Tools** menu).

expression.COMAddIns

expression Required. An expression that returns an **Application** object.

Remarks

All of the FrontPage add-ins that are registered in the Microsoft Windows Registry are located under the following key:

HKEY_CURRENT_USER\Software\Microsoft\Office\version\FrontPage\AddIns

Add-ins designed for administrators are registered under:

 $HKey_Local_Machine\Software\Microsoft\Office\version\FrontPage\Addins$

The following example returns the number of add-ins available to FrontPage.

```
Private Sub GetCOMAddIns()
   Dim myWeb As WebEx
   Dim myAddinCount As Integer
   Set myWeb = ActiveWeb
   myAddinCount = Application.COMAddIns.Count
```

End Sub

CommandBars Property

Returns a <u>**CommandBars</u>** collection that represents the menu and toolbars displayed in Microsoft FrontPage.</u>

expression.CommandBars

expression Required. An expression that returns an **Application** object.

Remarks

You can't access FrontPage-specific pop-up shortcut menus using the **CommandBars** collection. For example, you can't access the shortcut menu that appears when you right-click a page in **Page** view. However, you can create your own pop-up shortcut menus using the **ShowPopup** method for the **CommandBar** object.

The following three procedures set up a new menu item on a toolbar.

Procedure 1

The following procedure connects the **click** event to the custom button and must be added to a class or form module. The module shown is a form module. This procedure adds a new item to the **Tools** menu and connects the events of the custom button by assigning the variable e_NewMenu (used in the WithEvents statement) to the custom button variable newMenu.

Procedure 2

End Sub

To execute FrontPage custom menu items using the **CommandBars** collection, index the menu item and call the **execute** method for that item. The following example inserts a Microsoft Office spreadsheet control at the insertion point.

Procedure 3

```
Sub ExecuteMenu()
   Dim I As String
   Dim C As String
   Dim 0 As String
   I = "Insert"
   C = "C&omponent"
   0 = "Office Sp&readsheet"
   CommandBars(I).Controls(C).Controls(0).Execute
End Sub
```

The following example returns the status of various properties of the command bars in the active Web site.

```
Private Sub GetCommandBars()
    Dim myWeb As WebEx
    Dim myCB As Object
    Dim myCBCount As Integer
    Dim myDisplayFonts As Boolean
    Dim myDisplayKeysInToolTips As Boolean
    Dim myLargeButtons As Boolean
    Dim myMenuAnimationStyle As String
    Set myWeb = ActiveWeb
    Set myCB = Application.CommandBars
    With myCB
            myCBCount = .Count
            myDisplayFonts = .DisplayFonts
            myDisplayKeysInToolTips = .DisplayKeysInTooltips
            myLargeButtons = .LargeButtons
            myMenuAnimationStyle = .MenuAnimationStyle
    End With
End Sub
```

The following example is a tool that iterates through the command bars and returns several properties from each menu.

Note To run this example, create a form that has a text box called txtComBar and a command button called cmdComBar, and copy the following code to the code window.

```
Private Sub cmdComBar Click()
    Dim myWeb As WebEx
    Dim myComBars As Object
    Dim myComBar As Object
    Dim myText As String
    Dim myName As String
    Dim myAdaptMenu As String
    Dim myEnabledMenu As String
    Dim myMenuHeight As String
    Dim myMenuWidth As String
    Set myWeb = ActiveWeb
    Set myComBars = Application.CommandBars
    myName = "Name: "
    myAdaptMenu = "Menu Adaptive? "
   myEnabledMenu = "Menu Enabled? "
   myMenuHeight = "Menu Height: "
   myMenuWidth = "Menu Width: "
    txtComBar.Locked = True
    txtComBar.maxLength = 10000
    txtComBar.MultiLine = True
    txtComBar.ScrollBars = fmScrollBarsVertical
   With myComBars
        For Each myComBar In myComBars
            With myComBar
                myText = myText & myName & .Name & vbCrLf
                myText = myText & myAdaptMenu & .AdaptiveMenu & vbCr
                myText = myText & myEnabledMenu & .Enabled & vbCrLf
                myText = myText & myMenuHeight & .Height & vbCrLf
                myText = myText & myMenuWidth & .Width & vbCrLf
                txtComBar.Text = myText
            End With
        Next
        txtComBar.SetFocus
        txtComBar.CurLine = 0
    End With
End Sub
```

Company Property

Sets or returns a **String** that represents the name of the company that created a Web package.

expression.Company

expression Required. An expression that returns one of the objects in the Applies To list.

The following example creates a new Web package and adds the page "test.htm" to the package, including all dependencies for the page, and then saves the new Web package.

```
Dim objWeb As WebEx
Dim objPackage As WebPackage
Set objWeb = ActiveWeb
Set objPackage = objWeb.CreatePackage("New Web Package")
With objPackage
    .Author = "John Smith"
    .Company = "Fourth Coffee"
    .Subject = "This is a new Web package for Fourth Coffee."
    .Add objWeb.Url & "/test.htm", fpDepsDefault
    .Save "c:\NewWebPackage.fwp", True
End With
```

ComputationFormula Property

Sets or returns a **String** that represents the formula used to get a value for a calculated field.

expression.ComputationFormula

expression Required. An expression that returns a **ListFieldComputed** object.



ConnectionSpeed Property

Returns or sets an **<u>FpConnectionSpeed</u>** value that indicates the user's Internet connection type.

FpConnectionSpeed can be one of these FpConnectionSpeed constants. **fpConnect144** 14400 baud-rate modem **fpConnect288** 28800 baud-rate modem **fpConnect56K** 56600 baud-rate modem **fpConnectISDN** ISDN connection **fpConnectT1** T1 connection **fpConnectT3** T3 connection

expression.ConnectionSpeed

expression Required. An expression that returns an <u>Application</u> object.

Remarks

The **ConnectionSpeed** property is used in conjunction with the <u>SlowPage</u> property to determine the simulated download time of a Web page. The default value is **fpConnect144**.

The following example sets the **ConnectionSpeed** property to **fpConnect56K**, simulating a 56K modem. The amount of time a file takes to download and the criteria used to determine a "slow" page in the **Reports** view will be based on this value.

Sub SpeedOfConnection()
'Modifies the ConnectionSpeed property
Dim objApp As FrontPage.Application
Set objApp = FrontPage.Application
With objApp
 'Set the connection type to 56K modem
 .ConnectionSpeed = fpConnect56K
End With

End Sub

Count Property

Returns the number of items in the specified collection. Read-only **Variant**.

expression.Count

expression Required. An expression that returns one of the objects in the Applies To list.
Remarks

Some collections are 0-based and some are 1-based. The **Count** property for collections begins with 1 even though you access the first item in a collection with a 0.

The following example returns the number of themes and folders in the active Web site.

```
Private Sub GetWebCount()
   Dim myWeb As WebEx
   Dim myCount As Integer
   Set myWeb = ActiveWeb
   With myWeb
        myThemeCount = .Themes.Count
        myFolderCount = .RootFolder.Folders.Count
   End With
End Sub
```

Creator Property

Returns a **String** that represents the name of the application in which this object was created. If the object was created in Microsoft FrontPage, this property returns the **String** "FrontPage.Editor.Document".

Note The **Creator** property for FrontPage is different from the **Creator** property used by other Microsoft Office objects. The **Creator** property for FrontPage uses a string format, while Office objects use a 32-bit integer to identify the application that created the file.

expression.Creator

expression Required. An expression that returns one of the objects in the Applies To list.

This example checks if the creator of a file is FrontPage.

```
Private Sub GetCreator()
   Dim myCreator As String
   myCreator = ActiveWeb.RootFolder.Files(0).Creator
   If myCreator <> "FrontPage.Editor.Document" Then
        MsgBox "This file was not created by FrontPage."
   End If
End Sub
```



Currency Property

Returns or sets an **<u>FpCurrencyFieldFormat</u>** enumerated constant that determines the type of currency to be used in the currency field. Read/write.

FpCurrencyFieldFormat can be one of these **FpCurrencyFieldFormat** constants.

fpCurrencyFieldArgentina fpCurrencyFieldAustralia fpCurrencyFieldAustria fpCurrencyFieldBelgiumBF fpCurrencyFieldBelgiumFB fpCurrencyFieldBolivia fpCurrencyFieldBrazil **fpCurrencyFieldCanada fpCurrencyFieldCanadaF fpCurrencyFieldChile** fpCurrencyFieldColombia **fpCurrencyFieldCostaRica fpCurrencyFieldCzech fpCurrencyFieldDenmark fpCurrencyFieldDominicanRepublic fpCurrencyFieldEcuador fpCurrencyFieldElSalvador fpCurrencyFieldEuro fpCurrencyFieldEuroPostfix fpCurrencyFieldFinland fpCurrencyFieldFrance fpCurrencyFieldGermany fpCurrencyFieldGreece fpCurrencyFieldGuatemala fpCurrencyFieldHonduras**

fpCurrencyFieldHongKong fpCurrencyFieldHungary fpCurrencyFieldIreland fpCurrencyFieldItaly fpCurrencyFieldJapan fpCurrencyFieldKorea fpCurrencyFieldMexico fpCurrencyFieldNetherlands fpCurrencyFieldNewZealand **fpCurrencyFieldNicaragua fpCurrencyFieldNorway fpCurrencyFieldPanama fpCurrencyFieldParaguay** fpCurrencyFieldPeru **fpCurrencyFieldPoland fpCurrencyFieldPortugal fpCurrencyFieldPRChina fpCurrencyFieldRussia fpCurrencyFieldSingapore fpCurrencyFieldSlovakia fpCurrencyFieldSlovenia fpCurrencyFieldSouthAfrica fpCurrencyFieldSpain fpCurrencyFieldSweden fpCurrencyFieldSwitzerland fpCurrencyFieldTaiwan fpCurrencyFieldTurkey fpCurrencyFieldUnitedKingdom fpCurrencyFieldUnitedStates fpCurrencyFieldUruguay fpCurrencyFieldVenezuela**

expression.Currency

expression Required. An expression that returns a **ListFieldCurrency** object.

Remarks

The default value for this field is determined by the user's system settings.

The following example creates a new field of type **fpFieldCurrency** and changes the default currency setting to display Canadian.

```
Sub CreateCurrencyField()
'Add new Currency field
    Dim objApp As FrontPage.Application
    Dim objLstFlds As ListFields
    Dim objFldChoice As ListFieldCurrency
    Dim strName As String
    Set objApp = FrontPage.Application
    Set objLstFlds = objApp.ActiveWeb.Lists.Item(0).Fields
    strName = "NewCurrencyField"
    'Add new Field of type fpFieldCurrency to list
    objLstFlds.Add Name:=strName, Description:="New Choice value Fie
                   Fieldtype:=fpFieldCurrency, Required:=True
    Set objFldChoice = objLstFlds.Item("NewCurrencyField")
    'Change currency type to Canadian
    objFldChoice.Currency = fpCurrencyFieldCanada
    MsgBox "A new field named " & strName & " was added to the list
           objApp.ActiveWeb.Lists.Item(0).Name & "."
```

DefaultText Property

Sets or returns a **String** that represents the default value for a hyperlink field, which is the URL and the text displayed text for the hyperlink.

expression.DefaultText

expression Required. An expression that returns a **ListFieldURL** object.

DefaultValue Property

Returns or sets a **Variant** that defines the default value of the field. Read/write.

expression. Default Value

expression Required. An expression that returns one of the objects in the Applies To list.

Remarks

The default value of all pre-populated fields is **Empty**.

The following example displays the names of all fields in the list and their associated default values. If the active Web site does not contain any lists, a message is displayed to the user.

```
Sub FieldDefaultValue()
'Display the default value of the field
    Dim objApp As FrontPage.Application
    Dim objField As ListField
    Dim strType As String
    Set objApp = FrontPage.Application
    If Not ActiveWeb.Lists Is Nothing Then
        'Display fields in first list of collection
        For Each objField In objApp.ActiveWeb.Lists.Item(0).Fields
            If strType = "" Then
                'if first value in string
                strType = objField.Name & " - " & _
                objField.DefaultValue & vbCr
            Else
                'add value to string
                strType = strType & objField.Name & " - " & _
                objField. DefaultValue & vbCr
            End If
        Next objField
        MsgBox "The names of the fields in this list and their defau
                     " values are: " & vbCr & strType
    Else
        'Otherwise display message to user
        MsgBox "The current Web site contains no lists."
    End If
End Sub
```

DefaultViewPage Property

Returns or sets a **String** that defines the relative URL to the page that is viewed when the list is opened. This property corresponds to the default view page field on the **Supporting Files** tab of the **Properties** dialog box.

expression.DefaultViewPage

expression Required. An expression that returns one of the objects in the Applies To list.

Remarks

The default start page is AllItems.htm.

The following example lists the names of all lists in the document and their corresponding default view page URLs. The subroutine creates a single string containing all list names and default view pages and displays the formatted message to the user.

```
Sub ViewDefaultPage()
'Lets the user view the default view
'page for all lists in the web.
    Dim lstWebList As List
    Dim strURL As String
    If Not ActiveWeb.Lists Is Nothing Then
        'Cycle through lists
        For Each lstWebList In ActiveWeb.Lists
            'add default view pages names to string
            If strURL = "" Then
                strURL = lstWebList.Name & " - " & _
                lstWebList.DefaultViewPage & vbCr
            Else
                strURL = strURL & lstWebList.Name & " - " & _
                lstWebList.DefaultViewPage & vbCr
            End If
        Next
        'Display default view pages of all lists
        MsgBox "The default view pages of all lists in the current w
               & vbCr & vbCr & strURL
    Else
        'Otherwise display message to user
        MsgBox "The current web contains no lists."
    End If
```

Description Property

Returns or sets a **String** that represents the description for the current list. The description appears below the title of the list on the default view page. Read/write.

expression. Description

expression Required. An expression that returns one of the objects in the Applies To list.

The following example prompts the user to enter a description for the first list in the collection, and then changes the **Description** property based on the user's input.



DesignSecurity Property

Returns or sets an **<u>FpListDesignSecurity</u>** constant that defines the security permissions of the List.

FpListDesignSecurity can be one of these FpListDesignSecurity constants. **fpListDesignSecurityCreator** Only the creator of the list has permission to modify it.

fpListDesignSecurityEveryone All users have permission to modify the list.

expression. DesignSecurity

expression Required. An expression that returns one of the objects in the Applies To list.

The following example changes the security mode of each **BasicList** object in the current web. The subroutine changes the **DesignSecurity** property to fpListDesignSecurityEveryone if it isn't already set. Once the property is set, all users can edit the design settings of **BasicList** objects in the current web.

Note Use the ApplyChanges method to save any changes made to the list.

```
Sub SetSecurityType()
'Changes security type of all BasicLists.
    Dim objApp As FrontPage.Application
    Dim objList As List
    Dim objLists As Lists
    Set objApp = FrontPage.Application
    Set objLists = objApp.ActiveWeb.Lists
    'Cycle through each list and check for list type
    For Each objList In objLists
        'If it's a BasicList than change permissions
        If objList.Type = fpListTypeBasicList Then
            If objList.DesignSecurity <> _
               fpListDesignSecurityEveryone Then
                objList.DesignSecurity = _
                fpListDesignSecurityEveryone
            End If
            objList.ApplyChanges
        End If
    Next
```

DisplayForm Property

Returns or sets a **String** that represents the relative URL of the form that contains the user interface associated with the list. Read/write.

expression.DisplayForm

expression Required. An expression that returns one of the objects in the Applies To list.

The following example displays the names of all lists in the active Web site and the relative URLs of their associated Web forms.

```
Sub ViewFormURL()
'Displays the URL of the form
'associated with the list
    Dim lstWebList As List
    Dim strURL As String
    If Not ActiveWeb.Lists Is Nothing Then
        'Cycle through lists and add URLs to string
        For Each lstWebList In ActiveWeb.Lists
            If strURL = "" Then
                strURL = lstWebList.Name & " - " & _
                lstWebList.DisplayForm & vbCr
            Else
                strURL = strURL & lstWebList.Name & " - " & _
                lstWebList.DisplayForm & vbCr
            End If
        Next
        'Display URLs of all forms in Web site
        MsgBox "The relative URLs of the forms are:" _
               & vbCr & vbCr & strURL
    Else
        'Otherwise display message to user
        MsgBox "The current Web site contains no lists."
    End If
```



DisplayFormat Property

DisplayFormat property as it applies to the ListFieldChoice object.

Returns or sets an **<u>FpChoiceFieldFormat</u>** enumerated constant that represents the type of user interface control used by the field. Read/write.

FpChoiceFieldFormat can be one of these FpChoiceFieldFormat constants. fpChoiceFieldDropdown fpChoiceFieldRadioButtons

expression.DisplayFormat

expression Required. An expression that returns a **ListFieldChoice** object.

DisplayFormat property as it applies to the **ListFieldDateTime** object.

Returns or sets an **FpDateTimeFieldFormat** enumerated constant that represents the way in which the date and time values are displayed. Read/write.

FpDateTimeFieldFormat can be one of these FpDateTimeFieldFormat constants.

fpDateTimeFieldDateAndTime fpDateTimeFieldDateOnly fpDateTimeFieldTimeOnly

expression. Display Format

expression Required. An expression that returns a **ListFieldDateTime** object.

DisplayFormat property as it applies to the **ListFieldNumber** object.

Returns or sets an **<u>FpNumberFieldFormat</u>** enumerated constant that determines the way in which numbers are displayed in the List. Read/write.

FpNumberFieldFormat can be one of these FpNumberFieldFormat constants. **fpNumberFieldAuto** The number will be automatically formatted.

fpNumberFieldFiveDecimals The number will be displayed with five decimal places.

fpNumberFieldFourDecimals The number will be displayed with four decimal places.

fpNumberFieldInteger The number will be displayed as an integer with no decimal places.

fpNumberFieldOneDecimal The number will be displayed with one decimal place.

fpNumberFieldThreeDecimals The number will be displayed with three decimal places.

fpNumberFieldTwoDecimals The number will be displayed with two decimal places.

expression.DisplayFormat

expression Required. An expression that returns a **ListFieldNumber** object.

DisplayFormat property as it applies to the ListFieldURL object.

Returns or sets an **FpURLFieldFormat** enumerated constant that represents the image-linking information. Read/write.

FpURLFieldFormat can be one of these FpURLFieldFormat constants. fpURLFieldImage fpURLFieldLink

expression. Display Format

expression Required. An expression that returns a **ListFieldURL** object.

As it applies to the **ListFieldChoice** object.

The following example changes the display type of a field named NewChoiceField in the first list of the active Web site. The choices will now be displayed in a drop-down list.

```
Sub ChangeViewFormat()
'Change the display type of the field
Dim objApp As FrontPage.Application
Dim objListFields As ListFields
Dim objListField As ListFieldChoice
Set objApp = FrontPage.Application
Set objListFields = objApp.ActiveWeb.Lists.Item(0).Fields
Set objListField = objListFields.Item("NewChoiceField")
'Change display format to DropDown list
objListField.DisplayFormat = fpChoiceFieldDropdown
```

Document Property (Web Object Model)

Returns an **FPHTMLDocument** object, providing access to the Page object model in Microsoft FrontPage that is compatible with Microsoft Internet Explorer 4.0 and later. For more information on the Page object model, see <u>Exploring the Object Model in FrontPage</u>.

expression.Document

expression Required. An expression that returns a **PageWindowEx** object.

The following example opens a page and uses the **insertAdjacentText** method to add text to the document.

Note To run this example, you must have a Web site called "C:\My Documents\My Web Sites\Rogue Cellars" that contains a file named Sales.htm. You may substitute an alternative Web site URL or file name.

DownloadTime Property

Returns a **Long** that represents the simulated amount of time (in seconds) a given file will take to download. Read-only.

expression. Download Time

expression Required. An expression that returns a <u>WebFile</u> object.

Remarks

This property is used in conjunction with the <u>ConnectionSpeed</u> property to determine which files will appear in the Slow Pages report.

The following example displays the names of all files in the current Web site with a download time greater than a specified value. The subroutine prompts the user to enter a download time in seconds. It then searches through each file in the **All** collection and displays the names of any file with a download time greater than the specified number of seconds. The **Name** property value is added to a **String** containing the names of all matching files in the collection. The **String**, stored in the variable strName, is then displayed to the user. If no matching files are found in the Web site, a message is displayed to the user.

```
Sub DownloadTime()
'Displays the names of all files with a download time greater than
'a given value
    Dim objApp As FrontPage.Application
    Dim objwebFile As WebFile
    Dim objWebFiels As WebFiles
    Dim strSec As String 'User input value
    Dim strNames As String 'Name of all matching files
    Dim blnFound As Boolean 'Boolean flag
    Set objApp = FrontPage.Application
    Set objWebFiles = objApp.ActiveWeb.AllFiles
    blnFound = False
    'Prompt user to enter input
    strSec = InputBox("Enter the number of seconds download time.")
    'Search through each file in the collection
    For Each objwebFile In objWebFiles
        'If user input is less than download time
        If strSec < objwebFile.DownloadTime Then</pre>
            blnFound = True
            If strName = "" Then
                strName = strName & objwebFile.Name
            Else
                'Otherwise add next file name to string
                strName = strName & ", " & vbCr & objwebFile.Name
            End If
        End If
    Next objwebFile
    If blnFound = True Then
       'Display names of all files that match the criteria
       MsgBox "The files that take longer than " &
              strSec & " seconds to download are: " & vbCr & vbCr &
```

```
strName & "."
Else
'No files, display message
MsgBox "There are no files that match your criteria."
End If
```

DynamicTemplate Property

Returns a **String** that represents returns the path and file name of a Dynamic Web Template. An empty **String** indicates that no Dynamic Web Template is attached.

expression.DynamicTemplate

expression Required. An expression that returns a <u>WebFile</u> object.

Remarks

If an error occurs, the update process will terminate unless the **SkipOnQuery** property is set to **True**.
The following example updates the Dynamic Web Template for each file in the active Web site if a Dynamic Web Template is attached.

EditForm Property

Returns or sets a **String** that represents the relative URL of the form used for editing the current list in Microsoft FrontPage. The edit form allows you to modify the columns in the current list. Read/write.

expression.EditForm

expression Required. An expression that returns one of the objects in the Applies To list.

Remarks

The default filename is EditForm.htm.

The following example displays the names of all lists in the active Web site and the relative URL of their associated edit forms.

```
Sub ViewEditFormURL()
'Displays the URL of the form
'associated with editing the list
    Dim lstWebList As List
    Dim strURL As String
    If Not ActiveWeb.Lists Is Nothing Then
        'Cycle through lists and add URLs to string
        For Each lstWebList In ActiveWeb.Lists
            If strURL = "" Then
                strURL = lstWebList.Name & " - " & _
                lstWebList.EditForm & vbCr
            Else
                strURL = strURL & lstWebList.Name & " - " & _
                lstWebList.EditForm & vbCr
            End If
        Next
        'Display URLs of all editing forms in Web site
        MsgBox "The relative URLs of the editing forms are:" _
               & vbCr & vbCr & strURL
    Else
        'Otherwise display message to user
        MsgBox "The current Web site contains no lists."
    End If
```

End Sub



EditSecurity Property

Returns or sets an **<u>FpListEditSecurity</u>** constant that determines which users can edit the current list.

FpListEditSecurity can be one of these FpListEditSecurity constants. fpListEditSecurityAll All users can edit the list. fpListEditSecurityNone No users can edit the list. fpListEditSecurityOnlyOwn Users can only edit their own lists.

expression.EditSecurity

expression Required. An expression that returns one of the objects in the Applies To list.

The following example changes the edit permissions of all lists of type **fpListTypeBasicList** to **fpListEditSecurityOnlyOwn**. Once the property is set, users can edit only lists that they have created.

Note Use the ApplyChanges method to save any changes made to the list.

```
Sub ChangeEditPermissions()
'Changes the permissions of all BasicLists in the web
    Dim objApp As FrontPage.Application
    Dim objList As Object
    Dim objLists As Lists
    Set objApp = FrontPage.Application
    Set objLists = objApp.ActiveWeb.Lists
    'Cycle through each list and check for list type
    For Each objList In objLists
        'If it's a BasicList then change permissions
        If objList.Type = fpListTypeBasicList Then
            If objList.EditSecurity <> fpListEditSecurityOnlyOwn The
                objList.EditSecurity = fpListEditSecurityOnlyOwn
                objList.ApplyChanges
            End If
        End If
    Next
```

End Sub

EndNumber Property

Sets or returns a **Long** that represents the ending number for the number scale in a rating scale field.

expression.EndNumber

expression Required. An expression that returns a **ListFieldRatingScale** object.

Extension Property

Returns a **String** that represents the extension for the specified file. Read-only.

expression. Extension

expression Required. An expression that returns a **WebFile** object.

This statement returns the extension of the first page in the root folder of the active Web site.

myHomePageExt = ActiveWeb.RootFolder.File(0).Extension

Fields Property

Returns a **ListFields** collection that represents all fields in the current list. The **ListFields** collection contains **ListField** objects that correspond to the column properties of a list member document.

expression.Fields

expression Required. An expression that returns one of the objects in the Applies To list.

The following example displays the names of all fields in the first list of the **Lists** collection.

```
Sub DisplayFields()
'Returns the fields collection
    Dim objApp As FrontPage.Application
    Dim lstWebList As List
    Dim lstFields As ListFields
    Dim lstField As ListField
    Dim StrName As String
    Set objApp = FrontPage.Application
    Set lstWebList = objApp.ActiveWeb.Lists.Item(0)
    Set lstFields = lstWebList.Fields
    If Not ActiveWeb.Lists Is Nothing Then
        For Each lstField In lstFields
                'add URLs to string
                If StrName = "" Then
                    'If empty string
                    StrName = lstField.Name & vbCr
                Else
                    'add names to string
                    StrName = StrName & lstField.Name & vbCr
                End If
        Next
        'Display formatted string
        MsgBox "The list " & lstWebList.Name &
               "contains the following fields" & vbCr & vbCr & _
               StrName
    Else
      'Otherwise display message to user
        MsgBox "The current web contains no lists."
    End If
```

End Sub

File Property

Returns a <u>WebFile</u> object that represents the Web page associated with the specified object.

expression.File

expression Required. An expression that returns one of the objects in the Applies To list.

This example returns the name of the Web page associated with a navigation node in the active Web site.

```
Private Sub GetFileFromNavNode()
   Dim myNavNode As NavigationNode
   Dim myNavNodes As NavigationNodes
   Dim myNavFiles As String
   Dim myNavFile As String
   On Error Resume Next
   Set myNavNodes = ActiveWeb.HomeNavigationNode.Children
   For Each myNavNode In myNavNodes
        myNavFile = myNavNode.File.Name
        If ERR <> 0 Then Exit Sub
        myNavFiles = myNavFiles & myNavFile & vbCrLf
   Next
```

End Sub



FileDialog Property

Returns a **FileDialog** object that represents a single instance of a file dialog box.

expression.FileDialog(DialogType)

expression Required. An expression that returns an **Application** object.

DialogType Required <u>MsoFileDialogType</u>. The type of dialog box to open.

MsoFileDialogType can be one of these MsoFileDialogType constants. msoFileDialogFilePicker msoFileDialogFolderPicker msoFileDialogOpen msoFileDialogSaveAs

The following example displays the **Save As** dialog box.

```
Sub ShowSaveAsDialog()
'Display the Save As dialog box
Dim dlgSaveAs As FileDialog
'Set the dialog type
Set dlgSaveAs = Application.FileDialog(msoFileDialogSaveAs)
'Display the dialog
dlgSaveAs.Show
End Sub
```

The following example displays the **Open** dialog box, and allows the user to open multiple files at the same time.

The following example displays the **Open** dialog box, and allows the user to open multiple files at the same time. If the documents are HTML files, they are opened in Microsoft FrontPage.

```
Sub ShowOpenDialog()
'Display the Open dialog box
Dim dlgOpen As FileDialog
'Set the dialog box type to Open
Dim i as Integer
Set dlgOpen = Application.FileDialog(msoFileDialogOpen)
'Display the dialog box
With dlgOpen
    .AllowMultiSelect = True
    .Show
```

FileDialogViewPage Property

Returns or sets a **String** that represents the relative URL of the page associated with the **DocumentLibrary** object. Read/write.

expression.FileDialogViewPage

expression Required. An expression that returns a **<u>DocumentLibrary</u>** object.

The following example creates a new document library called NewLibrary and displays the relative URL of the **File** dialog page.

```
Sub NewLibrary()
'Add a new list to the current web
    Dim objApp As FrontPage.Application
    Dim objLists As Lists
    Dim objLibrary As DocumentLibrary
    Set objApp = FrontPage.Application
    Set objLists = objApp.ActiveWeb.Lists
    'Add new list
    objLists.Add Name:="NewLibrary", _
                 ListType:=fpListTypeDocumentLibrary, _
                 Description:="List of Shared files"
    Set objLibrary = objLists.Item("NewLibrary")
    'Display message to user
    MsgBox "A new list was added to the Lists collection." & _
           "The page associated with the file dialog is " &
            objLibrary.FileDialogViewPage & "."
```

End Sub

Files Property

Some of the content in this topic may not be applicable to some languages.

Returns a **WebFiles** collection that represents the items in the specified Web folder. Read-only object.

expression.Files

expression Required. An expression that returns a **WebFolder** object.

Remarks

The **Files** property is an accessor property used to access the **WebFiles** collection. To access the collection, declare a variable of type **WebFiles** as shown in the statement Dim myFiles As **WebFiles**, and then set the variable myFiles to Web.RootFolder.**Files**.

The following example retrieves all of the properties of a file and concatenates the META tags into a string with a pipe ("|") delimiter separating the data.

Note The **PropertyKeys** shown in this example apply to a Web site created with the One Page Web Site template. Other templates may use other **PropertyKeys**. For more information about using **PropertyKeys**, see the **Properties** collection.

```
Private Sub GetFileProperties()
    Dim myFiles As WebFiles
    Dim myFile As WebFile
    Dim myAuthor As String
    Dim myModifiedBy As String
    Dim myTimeCreated As String
    Dim myTimeLastModified As String
    Dim myFileSize As String
    Dim myTitle As String
    Dim myMetaTags As Variant
    Dim myMetaTag As Variant
    Dim myProgID As Variant
    Dim myGenerator As String
    Dim myTimeLastWritten As String
    Dim myProperties As Properties
    Dim myMetaTagList As String
    Set myFiles = ActiveWeb.RootFolder.Files
    For Each myFile In myFiles
        Set myProperties = myFile.Properties
        myAuthor = myAuthor & myProperties("vti_author")
        myModifiedBy = myModifiedBy & _
            myProperties("vti_modifiedby) & "|"
        myTimeCreated = myTimeCreated &
            myProperties("vti_timecreated") & "|"
        myTimeLastModified = myTimeLastModified &
            myProperties("vti_timelastmodified") & "|"
        myFileSize = myFileSize & _
            myProperties("vti_FileSize) & "|"
        myTitle = myTitle & myProperties("vti_title") & "|"
        myProgID = myProgID & myProperties("vti_title") & "|"
        myGenerator = myGenerator &
        myProperties("vti_generator") & "|"
myTimeLastWritten = myTimeLastWritten & _
            myProperties("vti_timelastwritten") & "|"
```

```
myMetaTags = myProperties("vti_metatags")
For Each myMetaTag In myMetaTags
    myMetaTagList = myMetaTagList & myMetaTag & "|"
    Next
Next
End Sub
```

FileSaveForm Property

Returns or sets a **String** that represents the relative URL of the form page that is displayed when files are saved. Read/write.

expression.FileSaveForm

expression Required. An expression that returns a **<u>DocumentLibrary</u>** object.

The following example creates a new document library called "NewLibrary" and displays the relative URL of the page associated with saving a file to the library.

```
Sub NewLibrary()
'Adds a new list to the current web
    Dim objApp As FrontPage.Application
    Dim objLists As Lists
    Dim objLibrary As DocumentLibrary
    Set objApp = FrontPage.Application
    Set objLists = objApp.ActiveWeb.Lists
    'Add new list
    objLists.Add Name:="NewLibrary", _
                 ListType:=fpListTypeDocumentLibrary, _
                 Description:="List of Shared files"
    Set objLibrary = objLists.Item("NewLibrary")
    'Display message to user
    MsgBox "A new list was added to the Lists collection." & _
           "The URL of the page associated with the Save dialog is "
            objLibrary.FileSaveForm & "."
```

End Sub

FileSearch Property

Returns a **<u>FileSearch</u>** object that provides access to file search capabilities within a Web site.

expression.FileSearch

expression Required. An expression that returns an **Application** object.

Remarks

The **FileSearch** object is a Microsoft Office shared object and does not accept a URL as a file name or folder name value.

The following example searches the Adventure Works Web site and its subdirectories and returns the number of Index.htm files found.

Note To run this example, you must have a Web site called "C:\My Documents\My Web Sites\Adventure Works". As an alternative, you can change the value for the **LookIn** property to a Web site that is currently available to you. You may also need to set a reference to the Microsoft Office Object Library (**References** option on the **Tools** menu).

```
Private Sub WebFileSearch()
   Dim myFileSearch As FileSearch
   Dim myFileCount As Integer
   Set myFileSearch = Application.FileSearch
   With myFileSearch
        .FileName = "index.htm"
        .LookIn = "C:\My Web Sites\Adventure Works"
        .SearchSubFolders = True
        .Execute
        myFileCount = .FoundFiles.Count
   End With
End Sub
```

Folder Property

Returns a <u>WebFolder</u> object that represents the folder associated with the list. The Web folder hierarchy provides the link to folders and files on a Web server directory. The navigation structure provides the underlying structure for the **Web** objects within individual Microsoft FrontPage Web sites.

expression.Folder

expression Required. An expression that returns one of the objects in the Applies To list.

The following example displays the name of the Web folder associated with the current list. If the current Web site contains no lists, a message is displayed to the user.

```
End Sub
```

Folders Property

Some of the content in this topic may not be applicable to some languages.

Returns a <u>WebFolders</u> collection that represents the child folders contained the specified folder. Read-only.

expression.Folders

expression Required. An expression that returns a **WebFolder** object.

Remarks

The **Folders** property returns the **WebFolders** collection for the specified Web site. To access the collection, you declare a variable of type **WebFolders** as in the statement Dim myFolders As **WebFolders**, and then set the variable to Web.RootFolder.Folders.

The following example retrieves two of the properties of a folder and concatenates the data into a string with a pipe ("|") delimiter separating the data.

Note The **PropertyKeys** shown in this example apply to a Web site created with the One Page Web Site template. Other templates may use other **PropertyKeys**. For more information about using **PropertyKeys**, see the **Properties** collection.

```
Private Sub GetFolderProperties()
   Dim myFolders As WebFolders
   Dim myFolder As WebFolder
   Dim myHasSubDirs As String
   Dim myProperties As Properties
   Set myFolders = ActiveWeb.RootFolder.Folders
   For Each myFolder In myFolders
        Set myProperties = myFolder.Properties
        myHasSubDirs = myHasSubDirs & ________
        myProperties("vti__ hassubdirs") & "|"
        myIsScriptable = myIsScriptable & ________
        myProperties("vti__ isscriptable") & "|"
        Next
End Sub
```

Format Property

Returns a **String** that represents the format of the specified **Theme** object. Read-only.

expression.Format

expression Required. An expression that returns a **Theme** object.
Remarks

The theme format reflects the difference between the formats for the different versions of Microsoft FrontPage. For example, the format number for FrontPage 98 can be either 0.0 or 1.0, while the format number for FrontPage 2000 is 2.0, and so on.

The following example retrieves the format of the applied theme for the active Web site.

```
Private Sub GetThemeFormat()
   Dim myFormat As String
   myFormat = ActiveWeb.Themes(0).Format
End Sub
```

FrameWindow Property

Returns an **FPHTMLWindow2** object that represents the page window and accesses the Internet Explorer Window objects.

Note The **Document** property of the **FPHTMLWindow2** object points to the **FPHTMLDocument** object of the frames page.

expression. FrameWindow

expression Required. An expression that returns a **PageWindowEx** object.

This example retrieves the **IHTMLLocation** property of the **FrameWindow** object.

Home Property

Returns a **<u>HomeNavigationNode</u>** object that represents the navigation node for a specified page.

expression.Home

expression Required. An expression that returns a **NavigationNode** object.

The following example searches for a navigation node with the label "Sale", and then updates it.

```
Private Sub ChangeNavLabel()
   Dim myFiles As WebFiles
   Dim myFile As WebFile
   Set myFiles = ActiveWeb.RootFolder.Files
   For Each myFile In myFiles
        If myFile.NavigationNode.Home.Label = "Sale" Then
            myFile.NavigationNode.Home.Label = "Sales Items"
        End If
        Next
End Sub
```

HomeNavigationNode Property

Returns a **NavigationNode** object for the home page.

expression.HomeNavigationNode

expression Required. An expression that returns a **WebEx** object.

Remarks

The home navigation node is the starting point for all navigation addressing within the navigation structure. Whenever you want to add, move, or delete a node, change any of the attributes for a node, or just access the current navigation structure, you use the **HomeNavigationNode** object as a starting point. The only time you would use the **RootNavigationNode** object to access the navigation structure is when you want to add or access a navigation node at the same level as the **HomeNavigationNode** object.

The following example accesses the **HomeNavigationNode** object and retrieves the URL for the home page.

```
Private Sub GetHomeNavigationNode()
   Dim myWeb As WebEx
   Dim myHomeNode As NavigationNode
   Dim myHomeUrl As String
   Set myWeb = ActiveWeb
   myHomeNode = myWeb.HomeNavigationNode
   myHomeUrl = myHomeNode.Url
End Sub
```

HorizontalResolution Property

Returns the horizontal resolution of the screen in pixels. Read-only **Long**.

expression.HorizontalResolution()

expression Required. An expression that returns a **System** object.

The following example returns the horizontal resolution of the screen.

myHoriz = System.HorizontalResolution

InNavBars Property

True to specify that the current page will be visible in the Web site's link bars. Read/write **Boolean**.

Note A link bar is a set of hyperlinks used for navigating a Web site.

expression.InNavBars

expression Required. An expression that returns a **<u>NavigationNode</u>** object.

Remarks

All pages with the **InNavBars** property set to **False** will appear grayed out in Navigation view.

The following example prompts the user to select which navigation nodes will appear in the link bar. If the user selects Yes, the current page will appear in the link bar. If the user selects No, the current page will not appear in the link bar and will appear grayed out in Navigation view. The user is prompted for each navigation node in the active Web site.

```
Sub AllNavigationNodes()
'Return a collection of all navigation nodes used in the current web
'Allows you to select which pages will appear in the link bar
    Dim objApp As FrontPage.Application
    Dim objNavNode As NavigationNode
    Dim objNavNodes As NavigationNodes
    Dim strAns As String
    Set objApp = FrontPage.Application
    'Create a reference to the NavigationNodes collection
    Set objNavNodes = objApp.ActiveWeb.AllNavigationNodes
    'For each node in the collection
    For Each objNavNode In objNavNodes
        'Prompt the user
        strAns = MsgBox("Do you want the page " & objNavNode.Label &
               " to appear in the link bar?", vbYesNo)
        'If user answers yes, set to True
        If strAns = vbYes Then
            objNavNode.InNavBars = True
        Else
            'If no, set to False
            objNavNode.InNavBars = False
        End If
    'Go to next node
    Next objNavNode
End Sub
```

IsDirty Property

True if the page displayed in the specified page window has changed since the last time the user saved the page. Read-write **Boolean**.

expression.IsDirty

expression Required. An expression that returns a **PageWindowEx** object.

The following example checks if the active page has changed, and then executes the **Save** method if the page has been modified.

IsExecutable Property

Returns or sets a **Boolean** that represents the setting for execute permission for a **WebFolder** object.

expression.IsExecutable

expression Required. An expression that returns one of the objects in the Applies To list.

This example retrieves the setting of the **IsExecutable** property for a **WebFolder** object.

Private Sub CheckExecutable()
 Dim myFolder As WebFolder
 Dim myExeStatus As Boolean
 Set myFolder = ActiveWeb.RootFolder.Folders("images")
 myExeStatus = myFolder.IsExecutable

End Sub

IsHidden Property

Sets or returns a **Boolean** that represents whether a list is displayed in the browser.

expression.IsHidden

expression Required. An expression that returns one of the objects in the Applies To list.

IsHiddenFoldersEnabled Property

True to display hidden folders in the specified Web site. Read/write **Boolean**.

$expression. \\ Is Hidden Folders Enabled$

expression Required. An expression that returns a <u>WebEx</u> object.

The following example prompts the user to display hidden folders in the current Web site. The **IsHiddenFoldersEnabled** property is set based on the user's response.

```
Sub ViewAllFolders()
'Prompts the user to view hidden folders
Dim objApp As FrontPage.Application
Dim objWeb As WebEx
Dim strAns As String
Set objApp = FrontPage.Application
Set objWeb = objApp.ActiveWeb
'prompt user
strAns = MsgBox("Do you want to view hidden folders?", vbYesNo)
'Set value of property to match user's response
If strAns = vbYes Then
        objWeb.IsHiddenFoldersEnabled = True
Else
        objWeb.IsHiddenFoldersEnabled = False
End If
```

End Sub

IsLinkBar Property

True indicates that the navigation node is a link bar. Read-only **Boolean**.

Note Link bars provide hypertext links that allow you to navigate through the pages in the current Web site.

expression.IsLinkBar

expression Required. An expression that returns a **<u>NavigationNode</u>** object.

The following example traverses the navigation node hierarchy and displays the names of any link bars in the Web site. If no link bars are found a message is displayed to the user.

```
Sub DisplayLinkBar()
'Return a collection of all navigation nodes used in the current Web
'Searches through the collection and displays the names of all link
    Dim objApp As FrontPage.Application
    Dim objNavNode As NavigationNode
    Dim objNavNodes As NavigationNodes
    Dim strAns As String
    Dim blnFound As Boolean
    blnFound = False
    Set objApp = FrontPage.Application
    'Create a reference to the NavigationNodes collection
    Set objNavNodes = objApp.ActiveWeb.AllNavigationNodes
    'For each node in the collection
    For Each objNavNode In objNavNodes
       'If set to True, this is a link bar
       If objNavNode.IsLinkBar = True Then
           MsgBox objNavNode.Label & " is a link bar."
           blnFound = True
       End If
    'Go to next node
    Next objNavNode
    'If no link bars are found, display a message
    If blnFound = False Then
        MsgBox "There are no link bars in the current Web site."
    End If
End Sub
```

IsModified Property

Returns a **Boolean** that represents whether a list has been changed since the last time the list was updated.

expression.IsModified

expression Required. An expression that returns one of the objects in the Applies To list.

IsOpen Property

True if the specified Web page is displayed in the page window. Read-only **Boolean**.

expression.IsOpen

expression Required. An expression that returns a **WebFile** object.

The following example uses the **IsOpen** property to check if a file named "index.htm" is open, and opens it if it isn't.

```
Private Sub CheckForOpenFile()
    Dim myWeb As WebEx
    Dim myFiles As WebFiles
    Dim myFile As WebFile
    Set myWeb = ActiveWeb
    Set myFiles = myWeb.RootFolder.Files
   With myWeb
        For Each myFile In myFiles
            If myFile.Name = "index.htm" Then
                If myFile.IsOpen = True Then
                    MsgBox "This file is open, try again later."
                    Exit Sub
                Else
                    myFile.Open
                    Exit Sub
                End If
            End If
        Next
    End With
End Sub
```

IsOrphan Property

True indicates that the file cannot be reached by hyperlink from any page in the Web site. Read-only.

expression.IsOrphan

expression Required. An expression that returns a **WebFile** object.

The following example searches through the current Web site and displays the names of all orphan files. An orphan file is denoted by its **IsOrphan** property. Once a file is found with an **IsOrphan** property that equals **True**, the **Label** property value is added to a **String** containing the names of all orphan nodes in the Web site. The names of the orphan files, stored in the **String** variable strName, are then displayed to the user. If no orphan files are found in the Web site, a message is displayed to the user.

```
Sub ListOrphans()
'Displays the names of orphan files.
    Dim objApp As FrontPage.Application
    Dim objWebFile As WebFile
    Dim strName As String
    Set objApp = FrontPage.Application
    'For each file in the Web site site
    For Each objWebFile In ActiveWeb.AllFiles
        'Check if the file is an orpahn
        If objWebFile. IsOrphan Then
            strName = strName & objWebFile.Name & " | "
        End If
    Next
    If strName <> "" Then
       'Display names of all orphan pages
       MsgBox "The orphan pages in the current Web site are: " & vbC
              strName & "."
    Else
        'No orphans, display message
        MsgBox "There are no orphan pages in the Web site."
    End If
End Sub
```

IsReadable Property

True indicates that a folder is has read permission. Read/write **Boolean**.

expression.IsReadable

expression Required. An expression that returns a **WebFolder** object.

The following example retrieves the setting of the **IsReadable** property for a folder called "images" in the active Web site.

```
Private Sub CheckReadable()
   Dim myFolder As WebFolder
   Dim myReadStatus As Boolean
   Set myFolder = ActiveWeb.RootFolder.Folders("images")
   myReadStatus = myFolder.IsReadable
```

End Sub

IsRoot Property

True if the specified Web folder is the root folder in the Web site. Read-only **Boolean**.

expression.IsRoot

expression Required. An expression that returns a **WebFolder** object.

The following example retrieves the setting of the **IsRoot** property for a **WebFolder** object.

Private Sub CheckExecutable()
 Dim myFolder As WebFolder
 Dim myIsRoot As Boolean
 Set myFolder = ActiveWeb.RootFolder.Folders("images")
 myIsRoot = myFolder.IsRoot

End Sub

IsUnderRevisionControl Property

True if source control is used on files in the specified Web site. Read-only **Boolean**.

expression.IsUnderRevisionControl

expression Required. An expression that returns a **WebEx** object.

Remarks

You must have a source control project created through Microsoft Visual SourceSafe or Microsoft Office in order to use revision control. For information about source control projects, refer to <u>Managing Source Control Projects</u>.

The following example creates a source control project and then, in the second procedure, checks the state of the **IsUnderRevisionControl** property.

```
Private Sub SourceControlProject()
    Dim myWeb As WebEx
    Set myWeb = ActiveWeb
    If Not (myWeb.IsUnderRevisionControl) Then
        myWeb.RevisionControlProject =
            "<FrontPage-based Locking>"
    End If
End Sub
Private Sub GetRevisionState()
    Dim myWeb As WebEx
    Dim myRevCtrlProj As String
    Dim myIsRevCtrl As Boolean
    Set myWeb = ActiveWeb
   With myWeb
        myRevCtrlProj = .RevisionControlProject
        myIsUnderRevCtrl = .IsUnderRevisionControl
    End With
End Sub
```
IsWeb Property

True if the specified Web folder is the root folder for a Web site, which may be the root folder for the active Web site or a subsite off of the active Web site. Read-only.

expression.IsWeb

expression Required. An expression that returns a **WebFolder** object.

This example retrieves the setting of the **IsWeb** property for a **WebFolder** object.

Private Sub CheckExecutable()
 Dim myFolder As WebFolder
 Dim myWebStatus As Boolean
 Set myFolder = ActiveWeb.RootFolder.Folders("images")
 myWebStatus = myFolder.IsWeb

End Sub

IsWritable Property

True if a folder has write permissions. Read-only **Boolean**.

expression.IsWritable

expression Required. An expression that returns a **WebFolder** object.

The following example retrieves the setting of the **IsWritable** property for a **WebFolder** object.

```
Private Sub CheckExecutable()
   Dim myFolder As WebFolder
   Dim myWritableStatus As Boolean
   Set myFolder = ActiveWeb.RootFolder.Folders("images")
   myWritableStatus = myFolder.IsWritable
```

End Sub



Item Property

Item property as it applies to the **MetaTags** object.

Returns a **Variant** representing a property key/value pair.

expression.Item(PropertyKey)

expression Required. An expression that returns one of the above objects.

PropertyKey Required **String**. A string that contains an index number of the collection. The index starts at zero.

Item property as it applies to the **Properties** object.

Returns or sets a **Variant** that represents a property. Read/write.

expression.Item(PropertyKey)

expression Required. An expression that returns one of the above objects.

PropertyKey Required **String**. A string that contains an index number of the collection. The index starts at zero.

Item property as it applies to all other objects in the Applies To list.

Returns a an individual object in a collection.

expression.Item(Index)

expression Required. An expression that returns one of the objects in the Applies To list.

Index Required **Variant**. The name or ordinal value of the object within the collection. Index starts at zero.

As it applies to the **NavigationNodes** collection.

The following example demonstrates returning a value by indexing an item in the collection. This example returns the label for the first navigation node in the navigation structure of the active Web.

Note You access the **NavigationNodes** collection through the **Children** property of the **RootNavigationNode** property of the active Web.

```
Private Sub GetNavigationNode()
   Dim myWeb As WebEx
   Dim myNavNodes As NavigationNodes
   Dim myNavNodeLabel As String
   Set myWeb = ActiveWeb
   myNavNodeLabel = myWeb.RootNavigationNode _
    .Children.Item(0).Label End Sub
```

As it applies to the MetaTags object.

The following statement returns the contents of a META tag that exists on a Web page in the active Web, and demonstrates the *PropertyKey* argument.

```
myMetaTagContents = ActiveWeb.RootFolder.Files _
.Item(0).MetaTags.Item("generator")
```

It isn't always necessary to specify the index or property name of the **Item** property when returning values from a collection. The following example returns a list of file names of each Web page that contains a META tag name equivalent to "generator" in the active Web, without specifying the **Item** property. FindGeneratorTags retrieves a list of the files that contain the "generator" META tag and adds *value* of the **Item** property to the variable myMetaTag, because in this case the value of the **Item** property is the same as the file name. This is different from the previous example, which returned the *contents* of the "generator" META tag.

Function FindGeneratorTags() As String

```
Dim myWeb As WebEx
    Dim myMetaTags As MetaTags
    Dim myMetaTag As Variant
    Dim myFiles As WebFiles
    Dim myFile As WebFile
    Dim myMetaTagName As String
    Dim myReturnFileName As String
    Set myWeb = ActiveWeb
    Set myFiles = myWeb.RootFolder.Files
   With myWeb
        For Each myFile In myFiles
            Set myMetaTags = myFile.MetaTags
                For Each myMetaTag In myMetaTags
                    myMetaTagName = myMetaTag
                    If myMetaTagName = "generator" Then
                        myReturnFileName = myReturnFileName & myFile
                    End If
                Next
        Next
    End With
    FindGeneratorTags = myReturnFileName
End Function
```



Label Property

As it applies to the **NavigationNode** object.

Sets or returns a **String** that represents the label associated with the active navigation node. This label is used as a reference for individual navigation nodes in Navigation view. Read/write.

expression.Label()

expression Required. An expression that returns a **NavigationNode** object.

Remarks

Use the text in the **Label** property for the text in a banner or button that links to another navigation node in the navigation structure.

As it applies to the **Theme** object.

Returns a **String** that represents the name of the specified theme.

expression.Label()

expression Required. An expression that returns a **Theme** object.

As it applies to the **NavigationNode** object.

The following example sets the text for the label of the first child node. The label shows the placement of the node in Navigation view.

```
Private Sub AddLabelToNavigationNode()

Dim myNode As NavigationNode

Set myNode = ActiveWeb.HomeNavigationNode.Children(0)

myNewNode.Label = "Finance Page"

End Sub
```

As it applies to the **Theme** object.

The following example retrieves the name of a theme.

Note To run this procedure, you must have an open Web site with a page that has previously had a theme applied to it.

```
Private Sub GetThemeName()
Dim myTheme As String
myTheme = ActiveWeb.Themes(0).Label
End Sub
```

LanguageDesignation Property

Returns the abbreviated name of the designated language of the system software. Read-only **String**.

```
expression.LanguageDesignation()
```

expression Required. An expression that returns a **System** object.

Remarks

Using the two-letter language abbreviation from the ISO Standard 639 and adding a third letter, such as "u" for the United States, creates the three-letter abbreviation.

You can also get the two-letter language abbreviation with **GetLocaleInfo()** by specifying **LOCALE_SABBREVLANGNAME** as the **LCType**. The abbreviated name "enu" is returned for the English (U.S.) language and is the same abbreviation that is returned with the **LanguageDesignation** property.

This example displays the abbreviated name of the designated language of the system software.

myLang = System.LanguageDesignation

LanguageSettings Property

Returns the <u>LanguageSettings</u> object for the Microsoft FrontPage application. Read-only LanguageSettings.

expression.LanguageSettings()

expression Required. An expression that returns an <u>Application</u> object.

The following example returns the **LanguageID** property for the user interface and sets the **LanguagePreferredForEditing** properties.

The LanguageSettings property has a required constant called MsoAppLanguageID. The enumerated values for this constant are msoLanguageIDHelp, msoLanguageIDInstall, msoLanguageIDUI, and msoLanguageIDUIPrevious. The LanguagePreferredForEditing property also has enumerated constants for the available languages. For more detailed information, see the LanguageSettings object.

List Property

Returns a **List** object that represents the list associated with the specified folder.

expression.List

expression Required. An expression that returns a <u>WebFolder</u> object.

The following example returns the list object associated with the second folder in the active Web site and displays the names of all fields in the list.

```
Sub ReturnList()
    'Returns the list associated with a folder
    Dim objApp As FrontPage.Application
    Dim objFolder As WebFolder
    Dim objListField As ListField
    Dim objList As List
    Dim strName As String
    Set objApp = FrontPage.Application
    For Each objFolder In objApp.ActiveWeb.AllFolders
        If Not objFolder.List Is Nothing Then
            'Return the List using the List property
            Set objList = objFolder.List
            For Each objListField In objList.Fields
                'Add list names to string
                If strName = "" Then
                    strName = objListField.Name & vbCr
                Else
                    strName = strName & objListField.Name & vbCr
                End If
            Next
            MsgBox "The field names within the" & objList.Name & " 1
            strName
        End If
    Next
End Sub
```

Lists Property

Returns a **Lists** collection object that represents a collection of all lists in a Web site. A list can be a **DocumentLibrary** object, a **BasicList** object, or a **Survey** object.

expression.Lists

expression Required. An expression that returns a **WebEx** object.

The following example returns a reference to the **Lists** collection using the **Lists** property for the active Web site. The example displays the name of each list in the collection.

End Sub

LookupField Property

Returns a **ListField** object that defines the field on which to perform a search.

expression.LookupField

expression Required. An expression that returns a **ListFieldLookup** object.

The following example creates a new field of type **fpFieldLookup** and displays the name of the new field and the name of the field that is being searched.

```
Sub CreateLookup()
'Adds new Lookup field
    Dim objApp As FrontPage.Application
    Dim objLstFlds As listFields
    Dim objFldLookup As ListFieldLookup
    Dim ObjField As ListField
    Dim strName As String
    Set objApp = FrontPage.Application
    Set objLstFlds = objApp.ActiveWeb.Lists.Item(0).Fields
    strName = "NewFileLookupField"
    'Add new Field of type fpFieldLookup to list
    objLstFlds.Add Name:=strName, Description:="New Lookup Field", _
                   Fieldtype:=fpFieldLookup
    Set objFldLookup = objLstFlds.Item("NewFileLookupField")
    MsgBox "A new field named " & strName & " was added to the list
           objApp.ActiveWeb.Lists.Item(0).Name & ". The field used f
           "lookup is " & objFldLookup.LookupField.Name & "."
```

End Sub

MaximumValue Property

Returns or sets a **Variant** that specifies the maximum value allowed for this field. This property is the equivalent of setting the **Maximum value allowed** field in the **Modify Field** dialog box of the user interface. Read/write.

expression.MaximumValue

expression Required. An expression that returns one of the objects in the Applies To list.

Remarks

An error message will be displayed to the user if invalid data is entered into this field. Use the **MinimumValue** property value to set the minimum value for the field.

The following example displays the names and maximum values of all fields of type **fpFieldNumber** and **fpFieldCurrency** in the first list of the active Web site. If the list contains no fields of this type, a message is displayed to the user.

```
Sub DisplayMaximum()
'Displays the maximum value of all ListFieldNumber
'and ListFieldCurrency fields in the list
    Dim objApp As FrontPage.Application
    Dim objLstFlds As ListFields
    Dim strName As String
    Dim objLstFld As Object
    Dim strValues As String
    Set objApp = FrontPage.Application
    Set objLstFlds = objApp.ActiveWeb.Lists.Item(0).Fields
    'Cycle through lists and add value to string
    For Each objLstFld In objLstFlds
        If (objLstFld.Type = fpFieldNumber) Or (objLstFld.Type = fpF
            strValues = strValues & objLstFld.Name & vbTab &
                        objLstFld.MaximumValue & vbCr
        End If
    Next objLstFld
    If strValues <> "" Then
        MsgBox "The fields and their maximum values are:" & vbCr & _
                vbCr & strValues
    Else
        MsgBox "There are no ListFieldNumber or ListFieldCurrency fi
    End If
```

End Sub

The following example changes the maximum value of all fields of type **ListFieldNumber** in the first list of the active Web site to a constant with the value 200.

Note Use the **<u>ApplyChanges</u>** method to apply any changes made to the list.

```
Sub ChangeMaximum()
'Changes maximum value for all fields of type
'ListFieldNumber
```

```
Dim objApp As FrontPage.Application
    Dim objLstFlds As ListFields
    Dim strName As String
    Dim objLstFld As Object
    Const varMax As Variant = 200
    Set objApp = FrontPage.Application
    If objApp.ActiveWeb.Lists.Count > 0 Then
        Set objLstFlds = objApp.ActiveWeb.Lists.Item(0).Fields
        'Cycle through lists and change values
        For Each objLstFld In objLstFlds
            If objLstFld.Type = fpFieldNumber Then
                objLstFld.MaximumValue = varMax
            End If
        Next objLstFld
        objApp.ActiveWeb.Lists(0).ApplyChanges
    Else
        MsqBox "The active Web site contains no lists."
    End If
End Sub
```

MaxLength Property (Web Object Model)

Returns or sets a **Long** that represents the maximum length (in characters) of the specified field. Read/write.

expression.MaxLength

expression Required. An expression that returns a **ListFieldSingleLine** object.

The following example adds a new field of type **fpFieldSingleLine** to the **ListFields** collection of the first list in the active Web site and displays the name of the new field, the name of the list to which it was added, and the maximum length in characters of the new field.

```
Sub CreateSingleLineField()
'Add new SingleLineField
    Dim objApp As FrontPage.Application
    Dim objLstFlds As ListFields
    Dim objListField As ListFieldSingleLine
    Dim strName As String
    Set objApp = FrontPage.Application
    Set objLstFlds = objApp.ActiveWeb.Lists.Item(0).Fields
    strName = "AlternativeName"
    'Add new field of type fpFieldSingleLine to list
    objLstFlds.Add Name:=strName, Description:="Numeric Total Field"
                   Fieldtype:=fpFieldSingleLine
    Set objListField = objLstFlds.Item("AlternativeName")
    MsgBox "A new field named " & strName & _
           " was added to the list " &
           objApp.ActiveWeb.Lists.Item(0).Name &
            _ ". The maximum length of the " & _
           "field is " & objListField.MaxLength & " characters."
```

End Sub

MetaTags Property

Returns the **MetaTags** collection for the specified **WebFile** object.

expression.MetaTags

expression Required. An expression that returns a **WebFile** object.

The following example iterates through the META tags collection in each file in the active Web site and concatenates the file names and META tag names into a string called myReturnInfo.

```
Private Sub GetMetaTagInfo_Click()
    Dim myWeb As WebEx
    Dim myFiles As WebFiles
    Dim myFile As WebFile
    Dim myMetaTags As MetaTags
    Dim myMetaTag As Variant
    Dim myFileName As String
    Dim myMetaTagName As String
    Dim myReturnInfo As String
    Set myWeb = ActiveWeb
    Set myFiles = myWeb.RootFolder.Files
   With myWeb
         For Each myFile In myFiles
             Set myMetaTags = myFile.MetaTags
             For Each myMetaTag In myMetaTags
                     myFileName = myFile.Name
                     myMetaTagName = myMetaTag
                     myReturnInfo = myFileName & ": " _
                         & myMetaTagName
             Next
         Next
    End With
End Sub
```

MinimumValue Property

Returns or sets a **Variant** that specifies the minimum allowed value for the field. This property is the equivalent of setting the **Minimum value allowed** field in the **Modify Field** dialog box of the user interface. Read/write.

expression.MinimumValue

expression Required. An expression that returns one of the objects in the Applies To list.

Remarks

An error message will be displayed to the user if invalid information is entered into this field. Use the **MaximumValue** property to set the maximum allowed value of the field.

The following example displays the names and minimum values for all fields of type **fpFieldNumber** and **fpFieldCurrency** in the first list of the active Web site.

```
Sub DisplayMinimum()
'Displays the minimum value of all ListFieldNumber
'and ListFieldCurrency fields in the list
    Dim objApp As FrontPage.Application
    Dim objLstFlds As ListFields
    Dim strName As String
    Dim objLstFld As Object
    Dim strValues As String
    Set objApp = FrontPage.Application
    Set objLstFlds = objApp.ActiveWeb.Lists.Item(0).Fields
    'Cycle through lists and add value to string
    For Each objLstFld In objLstFlds
        If (objLstFld.Type = fpFieldNumber) Or (objLstFld.Type = fpF
            strValues = strValues & objLstFld.Name & vbTab &
                        objLstFld.MinimumValue & vbCr
        End If
    Next objLstFld
    If strValues <> "" Then
        MsgBox "The fields and their minimum values are:" & vbCr & _
                vbCr & strValues
    Else
        MsgBox "There are no ListFieldNumber or ListFieldCurrency fi
    End If
```

End Sub

The following example changes the minimum value of all fields of type **fpListFieldNumber** in the first list in the active Web site to a constant with the value 200.

Note Use the **<u>ApplyChanges</u>** method to apply any changes made to the list.

```
Sub ChangeMinimum()
'Changes minimum value for all fields of type
'ListFieldNumber
```

```
Dim objApp As FrontPage.Application
    Dim objLstFlds As ListFields
    Dim strName As String
    Dim objLstFld As Object
    Const varMin As Variant = 1
    Set objApp = FrontPage.Application
    If objApp.ActiveWeb.Lists.Count > 0 Then
        Set objLstFlds = objApp.ActiveWeb.Lists.Item(0).Fields
        'Cycle through lists and change values
        For Each objLstFld In objLstFlds
            If objLstFld.Type = fpFieldNumber Then
                objLstFld.MinimumValue = varMin
            End If
        Next objLstFld
        objApp.ActiveWeb.Lists(0).ApplyChanges
    Else
        MsqBox "The active Web site contains no lists."
    End If
End Sub
```

MonthsShown Property

Returns or sets a **Long** that determines how many months will be displayed in the Microsoft FrontPage Reports view. Read/write.

expression.MonthsShown

expression Required. An expression that returns an <u>Application</u> object.
The following example prompts the user to enter the number of months he or she wants to view in the report, and then sets the **MonthsShown** property to that value. The subroutine "SetMonths" prompts the user for input, performs a validation on the input data, converts it to the correct type and sets the **MonthsShown** property to the new value. If the value is of an incorrect type, an error message is displayed to the user.

```
Sub SetMonthsShown()
'Modifies the MonthsShown property
   Dim objApp As FrontPage.Application
  Set objApp = FrontPage.Application
  Call SetMonths(objApp)
End Sub
Sub SetMonths(ByRef objApp As Application)
'Sets the number of months to view in Reports view
    Dim varNum As Variant
    Dim lngNum As Long
    'Prompt the user to enter a value
    varNum = InputBox("Enter the number of months you wish to view i
    'Check to see that the value is of the correct type
    If IsNumeric(varNum) Then
       'If it's numeric, convert it to Long
       lngNum = CLng(varNum)
       'Set the MonthsShown value to the new value
       objApp.MonthsShown = lngNum
       'Display the new setting information to the user
       MsgBox "The MonthsShown value was set correctly." & _
           " The number of months that will be shown is " _
           & lngNum & "."
    Else
       'Otherwise, display an error message to the user
       MsqBox "The input value was incorrect", vbCritical
    End If
```

End Sub

Name Property

Returns a **String** that represents the name assigned to the specified object. Readonly **String**.

expression.Name

expression Required. An expression that returns one of the objects in the Applies To list.

The name of a file or folder is usually the last part of the URL. For example, if you have a URL of "C:\My Web Sites\Adventure Works\index.htm", the value of the **Name** property for the file is "index.htm". Similarly, "Images" is the value of the **Name** property for the folder in the URL "C:\My Web Sites\Adventure Works\Images".

The following statement returns the application name with AppName as the string variable.

AppName = Application.Name

NavigationNode Property

Returns a **NavigationNode** object that represents the current node in the navigation structure.

expression.NavigationNode

expression Required. An expression that returns a **WebFile** object.

If a **NavigationNode** is not found within the navigation structure for the specified Web site, the **NavigationNode** property returns **Null**.

The following example uses the **NavigationNode** property to return the file name associated with the navigation node.

NewForm Property

Returns or sets a **String** that represents the form used for adding new content to the current list in Microsoft FrontPage. Read/write.

expression.NewForm

expression Required. An expression that returns one of the objects in the Applies To list.

The default file name for the **<u>BasicList</u>** and <u>**Survey</u>** objects is NewForm.htm. The default file name for the <u>**DocumentLibrary**</u> object is Upload.htm.</u>

The following example displays the name of each list in the active Web site and the relative URLs of their associated New form pages. If the active Web site contains no lists, a message is displayed to the user.

```
Sub ViewNewFormURL()
'Display the URL of the form
'associated with adding new content
    Dim lstWebList As List
    Dim strURL As String
    If Not ActiveWeb.Lists Is Nothing Then
        'Cycle through lists and add URLs to string
        For Each lstWebList In ActiveWeb.Lists
            If strURL = "" Then
                strURL = lstWebList.Name & " - " &
                lstWebList.NewForm & vbCr
            Else
                strURL = strURL & lstWebList.Name & " - " & _
                lstWebList.NewForm & vbCr
            End If
        Next
        'Display URLs of all New forms in Web site
        MsgBox "The relative URLs of the New forms are:" _
               & vbCr & vbCr & strURL
    Else
        'Otherwise display message to user
        MsqBox "The current Web site contains no lists."
    End If
```

```
End Sub
```

NewPageorWeb Property

Returns a **<u>NewFile</u>** object that represents a page or Web site listed on the **New** task pane.

expression.NewPageorWeb

expression Required. An expression that returns an <u>Application</u> object.

The following example creates a new instance of the **NewFile** object and adds a file named "template.htm" to the **New** task pane. The new file listing will show up under the **Other files** section listed at the bottom of the task pane.

```
Sub NewPage()
'Creates a new page using the NewFile object
Dim objApp As FrontPage.Application
Dim objNewFile As NewFile
Set objApp = FrontPage.Application
'Create a reference to an instance of the NewFile object
Set objNewFile = objApp.NewPageorWeb
objNewFile.Add "template.htm"
```

End Sub

Next Property

Returns a **NavigationNode** object that represents the next navigation node in the navigation sequence. Read-only Object.

expression.Next

expression Required. An expression that returns a **NavigationNode** object.

Although the **Next** property is a member of the **NavigationNode** class, this property navigates within the **Children** collection of the specified **NavigationNode** object.

Note The **Children** collection does not wrap, so that code such as Children(Children.Count – 1).Next returns an "Object variable or With block variable not set" error.

The following example moves the navigation pointer to the next node, unless the current node is the last node of the level in the navigation structure.

```
Private Sub MoveNext()
   Dim theNode As NavigationNode
   Dim theNextNode As NavigationNode
   On Error Resume Next
   Set theNode = ActiveWeb.HomeNavigationNode.Children(1)
   Set theNextNode = theNode.Next
   If Err <> 0 then
        MsgBox "End of the current navigation row"
   End If
End Sub
```

NumberOfLines Property

Returns or sets a **Long** that represents the number of lines that will appear in the field. Read/write.

expression.NumberOfLines

expression Required. An expression that returns a **ListFieldMultiLine** object.

The following example adds a new **ListFieldMultiLine** field named "Description" to the **ListFields** collection. The subroutine displays the name of the new field as well as the number of lines it will contain.

```
Sub CreateMultiLine()
'Add new MultiLine Field
    Dim objApp As FrontPage.Application
    Dim objLstFlds As ListFields
    Dim ObjField As ListField
    Dim objLstFldMulti As ListFieldMultiLine
    Dim strName As String
    Set objApp = FrontPage.Application
    Set objLstFlds = objApp.ActiveWeb.Lists.Item(0).Fields
    strName = "Description"
    'Add new Field of type fpFieldMultiLine to list
    objLstFlds.Add Name:=strName, Description:="Description Field",
                   Fieldtype:=fpFieldMultiLine
    Set objLstFldMulti = objLstFlds.Item(strName)
    objLstFldMulti.NumberOfLines = 5
    MsgBox "A new field named " & strName & " was added to the list
           objApp.ActiveWeb.Lists.Item(0).Name & ". It contains " &
           objLstFldMulti.NumberOfLines & " lines."
```

End Sub

OlderFile Property

Returns or sets a **Long** that determines the number of days that a file must exist in a Web site (without being modified) before it is classified as older. Once a file is classified as older, it appears in the Older Files view in the Reports view. Read/write.

expression.OlderFile

expression Required. An expression that returns an <u>Application</u> object.

Use the **RecentFile** property to return or set the number of days that a new or recently modified file shows up in the Recently Added Files list in the Reports view.

The following example prompts the user to enter a value that specifies the number of days a file must exist before it is classified as older. The subroutine SetOldVal prompts the user for input, performs a validation on the input data, converts it to the correct type, and sets the **OlderFile** property to the new value. If the value is of an incorrect type, an error message is displayed to the user.

```
Sub FPOldFile()
'Sets a value that determines how old a file is
   Dim objApp As FrontPage.Application
  Set objApp = FrontPage.Application
  Call SetOldVal(objApp)
End Sub
Sub SetOldVal(ByRef objApp As Application)
'Sets the value that determines how old a file is
    Dim varNum As Variant
    Dim lngNum As Long
    'Prompt the user to enter a value
    varNum = InputBox("Enter the number of days a file can exist " &
                        "before it is classified as old.")
    'Check to see that the value is of the correct type
    If IsNumeric(varNum) Then
       'If it's numeric, convert it to Long
       lngNum = CLng(varNum)
       'Set the OlderFIle value to the new value
       objApp.OlderFile = lnqNum
       'Display the new setting information to the user
       MsgBox "The OlderFile value was set correctly." & vbCr & _
              "The number of days after which a file becomes old is
    Else
       'Otherwise, display an error message to the user
       MsgBox "The input value was incorrect.", vbCritical
    End If
```

End Sub

OperatingSystem Property

Returns a **String** that represents the name of the current operating system, for example, "Windows" or "Windows NT".

expression.OperatingSystem()

expression Required. An expression that returns a **System** object.

The following example displays system information in a label on a form.

lblSystemInfo.Caption = System.OperatingSystem

This example prints the name of the current operating system in the **Immediate** window.

myOpSys = System.OperatingSystem



OptimizeHTMLFlags Property

Returns or sets an **<u>FpOptimizeHTMLFlags</u>** constant that represents how the HyperText Markup Language (HTML) is optimized in a Web page.

FpOptimizeHTMLFlags can be one or more of the following **FpOptimizeHTMLFlags** constants.

fpHtmlOptAdjacentTags	Combines adjacent elements of the same type.
fpHtmlOptAuthorComponents	Removes Author-Time FrontPage Web component comments.
fpHtmlOptBrowseComponents	Removes Browse-Time FrontPage Web component comments.
fpHtmlOptCellFormattingAttr	Removes cell formatting sttributes.
fpHtmlOptDwtCmnts	Removes Dynamic Web Template comments.
fpHtmlOptEmpty	Removes empty tags.
fpHtmlOptGenerator	Removes META elements that contain Generator and Programatic Identifier information.
fpHtmlOptHTMLAllWhitespace	Removes all white spaces that don't affect rendering.
fpHtmlOptHTMLCmnts	Removes all other HTML comments.
fpHtmlOptHTMLLeadWhitespace	Removes leading white spaces from each line.
fpHtmlOptHTMLMisnest	Removes incorrectly nested tags.
fpHtmlOptOn	Enables optimization.
fpHtmlOptScriptCmnts	Removes script comments.
fpHtmlOptThemes	Removes theme comments.
fpHtmlOptTrcImageAttr	Removes image tracing attributes from the BODY element.
fpHtmlOptUnusedStyles	Removes unused styles.
	Removes Vector Markup Language

fpHtmlOptVMLGraphics

(VML) from a page that uses Office drawings and WordArt. Removes Word-specific HTML markup.

fpHtmlOptWordHTML

expression.OptimizeHTMLFlags

expression Required. An expression that returns an **Application** object.

Use **fpHTMLOptOn** to turn optimization on. Specify additional optimization settings by separating each with an ampersand (&).

The following example turns optimization on for the current page and specifies that all Microsoft FrontPage Web component comments are to be removed from the page.



OptimizeHTMLPublishFlags Property

Sets or returns one or more **FpOptimizeHTMLFlags** constants that represents how the HyperText Markup Language (HTML) in the pages of a Web site is optimized during publishing.

FpOptimizeHTMLFlags can be one or more of the following **FpOptimizeHTMLFlags** constants.

fpHtmlOptAuthorComponents	Removes Author-Time FrontPage Web
fpHtmlOptBrowseComponents	Removes Browse-Time FrontPage Web component comments.
fpHtmlOptCellFormattingAttr	Removes cell formatting attributes.
fpHtmlOptDwtCmnts	Removes Dynamic Web Template comments.
fpHtmlOptGenerator	Removes META elements that contain Generator and Programatic Identifier information.
fpHtmlOptHTMLAllWhitespace	Removes all white spaces that don't affect rendering.
fpHtmlOptHTMLCmnts	Removes all HTML comments.
fpHtmlOptHTMLLeadWhitespace	Removes leading white spaces from each line.
fpHtmlOptOn	Enables optimization.
fpHtmlOptScriptCmnts	Removes script comments.
fpHtmlOptThemes	Removes theme comments.
fpHtmlOptTrcImageAttr	Removes image tracing attributes from the BODY element.
fpHtmlOptVMLGraphics	Removes Vector Markup Language (VML) from a page that uses Office drawings and WordArt.
fpHtmlOptWordHTML	Removes Word-specific HTML markup.

expression.OptimizeHTMLPublishFlags

expression Required. An expression that returns one a <u>WebEx</u> object.

Use **fpHTMLOptOn** to turn optimization on. Specify additional optimization settings by separating each with an ampersand (&).

The following example turns optimization on for the active Web site and specifies that all Microsoft FrontPage Web component comments are to be removed when publishing the site.

OrganizationName Property

Returns the name of the organization for the application. Read-only **String**.

Note The name of the organization is usually set during the installation of an application or operating system.

Instead of creating a company name variable for your Web page, you can use the organization name as shown in the following statement.

myCompanyName = Application.OrganizationName
PageWindows Property

Returns the specified **<u>PageWindowEx</u>** object.

expression.PageWindows

expression Required. An expression that returns a **WebWindowEx** object.

The following example retrieves a page window that contains the page Spain.htm and sets the view mode to Preview.

Note To run this procedure, you must have an open Web site that contains an open page called Spain.htm, or substitute a file of your choice.

```
Private Sub SetPagePreview()
   Dim myPage As PageWindowEx
   Set myPage = ActiveWebWindow.PageWindows("Spain.htm")
   myPage.ViewMode = fpPageViewPreview
End Sub
```

Parent Property

Returns an Object that represents the **Parent** object for the specified object.

expression.Parent

Remarks

Specifying the **Parent** property of a **WebFile** object returns the **WebFolder** object. When an object is contained in a Web site, the **Parent** property returns the **WebEx** object. For example, if the **Theme** object is contained in a **WebEx** object called "Adventure Works Web", the **Parent** property returns "Adventure Works Web"; otherwise, for a theme referenced outside of a **WebEx** object, the **Parent** property returns the name of the **Application** object— in this case, "Microsoft FrontPage". However, when the **System** object is specified from a client computer, the **Parent** property returns the **Application** object of the host, not the client.

The following table describes the values returned for the different object types.

Object	Description	
MetaTags	Returns the WebFile object for the META tags.	
NavigationNode	Returns the parent NavigationNode object, except in the case of the RootNavigationNode object, whose parent is the WebEx object.	
NavigationNodes	Returns the parent NavigationNode object for the collection of navigation nodes based on the navigation structure.	
PageWindowEx	Returns the WebWindowEx or Application object in which the page resides.	
PageWindows	Returns the Application object that contains the collection.	
Properties	Returns the WebEx , WebFile , or WebFolder object from the META tag information.	
System	Returns the Application object.	
Theme	Returns the parent WebEx or WebFile object for the theme.	
Themes	Returns the WebEx object that contains the collection.	
WebEx	Returns the Application object.	
WebFile	Returns the WebFolder object.	
WebFiles	Returns the parent WebFolder object.	
WebFolder	Returns either the parent WebFolder object (if it's a subfolder) or the WebEx object for the root folder.	

WebFolders	Returns the WebFolder object that contains the collection.
Webs	Returns the Application object.
WebWindowEx	Returns the Application object that contains the specified object.
WebWindows	Returns the Application object that contains the collection.

In the following example, myParent returns the file type and build of the parent application for the active Web site.

```
Private Sub GetParentInfo()
   Dim myWeb As WebEx
   Dim myParent As String
   Dim myParentBuild As String
   Set myWeb = Application.ActiveWeb
   With myWeb
      myParent = .Parent.FileSearch.FileType
      myParentBuild = .Parent.Build
   End With
End Sub
```

Prev Property

Returns a **NavigationNode** object that represents the previous navigation node in the navigation sequence. Read-only.

expression.Prev

expression Required. An expression that returns a **NavigationNode** object.

The following example moves the navigation pointer to the previous node, unless the current node is the first node of the level in the navigation structure.

```
Private Sub MovePrev()
   Dim theNode as NavigationNode
   Dim thePrevNode as NavigationNode
   On Error Resume Next
   Set theNode = ActiveWeb.HomeNavigationNode.Children(1)
   Set thePrevNode = theNode.Prev
   If Err <> 0 Then
        MsgBox "The current navigation level starts here."
   End If
End Sub
```

PreviewDocument Property

Returns an **IHTMLDocument2** object that represents the document in the preview mode of the **Page** view.

expression.**PreviewDocument**

expression Required. An expression that returns a **<u>PageWindowEx</u>** object.

Remarks

The **PreviewDocument** property returns nothing if the active page window is not currently in preview mode.

The following example displays the title of the document currently in preview mode. If there is no document in preview mode, a message is displayed to the user.

```
End Sub
```

ProductCode Property

Returns the globally unique identifier (GUID) for Microsoft FrontPage. Readonly **String**.

expression.ProductCode()

expression Required. An expression that returns an <u>Application</u> object.

This statement returns the Product ID for FrontPage.

mySystem = Application.ProductCode

ProfileString Property

Returns or sets a **String** that represents an entry in the Microsoft Windows registry under the following subkey:

HKEY_CURRENT_USER\Software\Microsoft\FrontPage\

expression.ProfileString(RegistrySection, RegistryKey)

expression Required. An expression that returns a **System** object.

RegistrySection Required String. A subkey below the "HKEY_CURRENT_USER\Software\Microsoft\FrontPage\" subkey in the Windows registry.

RegistryKey Required **String**. The name of the entry in the subkey specified by *szSection*. For example, Software or Network in the Registry Editor are subkeys.

The following example returns the value of the subkey, 0 (zero), in the Recently Used URLs entry.

Note To run this example, you must have recently opened a page that exists in one of your Web sites.

```
Private Sub GetRegRecentlyUsedInfo()
   Dim mySecKey As String
   Dim myRegKey As String
   Dim myProfile As String
   mySecKey = "HKEY_CURRENT_USER\Software"
   mySecKey = mySecKey & _______"\Microsoft\FrontPage\Editor\Recently Used URLs"
   myRegKey = "0"
   myProfile = System.ProfileString(mySecKey, myRegKey)
End Sub
```

Properties Property

Returns a **Properties** collection that represents the properties for the specified object.

expression.**Properties**

The following example adds a new property and displays it on the active Web page.

Note To run this example, you must have a Web site called "C:\My Documents\My Web Sites\Rogue Cellars" and a file called "Zinfandel.htm". Or, you may substitute an alternative Web site URL and file name.

```
Private Sub CopyrightAdd()
   Dim myWeb As WebEx
   Dim myCopyright As String
   myCopyright = "Copyright 1999 by Rogue Cellars"
   Set myWeb = Webs.Open("C:\My Documents\My Web Sites\Rogue Cellar
   myWeb.Activate
   ActiveWeb.Properties.Add "Copyright", myCopyright
   ActiveWeb.RootFolder.Files("Zinfandel.htm").Open
   ActiveWeb.RootFolder.Files("Copyright")
   ActiveWeb.Properties("Copyright")
   ActiveWeb.Close
End Sub
```

ReadOnly Property

Returns a **Boolean** that determines if a specified field has read-only permissions. If **True**, the field cannot be modified by the user. Read-only.

expression.ReadOnly

The following example stores the names and default values of all fields with read-only permissions in the first list of the active Web site. If the active Web site contains no lists, a message is displayed to the user.

```
Sub FieldPermissions()
'Displays read/write permissions of all
'fields in the list.
    Dim objApp As FrontPage.Application
    Dim objField As ListField
    Dim objFields As listFields
    Dim strPerms As String
    Set objApp = FrontPage.Application
    Set objFields = objApp.ActiveWeb.Lists.Item(0).Fields
    If Not ActiveWeb.Lists Is Nothing Then
        For Each objField In objFields
             'If field is read-only, add to list
             If objField.ReadOnly = True Then
                If strPerms = ""<sup>Then</sup>
                    'if first value in string
                    strPerms = objField.Name & " - " & _
                    objField.DefaultValue & vbCr
                Else
                    'add value to string
                    strType = strPerms & objField.Name & " - " &
                    objField.DefaultValue & vbCr
                End If
             End If
        Next objField
    Else
        'display message to user
        MsqBox "The active Web site contains no lists."
    End If
End Sub
```



ReadSecurity Property

Returns or sets an **<u>FpListReadSecurity</u>** constant that represents which users can read the information in a specified list. Read/write.

FpListReadSecurity can be one of these FpListReadSecurity constants. **fpListReadSecurityAll** All users can read the list. **fpListReadSecurityOnlyOwn** Only the creator of the list can read it.

expression.ReadSecurity

The following example changes the read permissions of all lists of type **fpListBasicList** in the active Web site to **fpListReadSecurityAll**. All users can now read all lists of type **fpListTypeBasicList**.

Note Use the ApplyChanges method to save any changes to the list.

```
Sub ChangePermissions()
'Changes permission of all BasicLists in the current Web site
    Dim objApp As FrontPage.Application
    Dim objList As Object
    Dim objLists As Lists
    Set objApp = FrontPage.Application
    Set objLists = objApp.ActiveWeb.Lists
    'Cycle through each list and check for list type
    For Each objList In objLists
        'If it's a BasicList then change permissions
        If objList.Type = fpListTypeBasicList Then
            If objList.ReadSecurity <> fpListReadSecurityAll Then
                objList.ReadSecurity = fpListReadSecurityAll
            objList.ApplyChanges
            End If
        End If
    Next
```

End Sub

RecentFile Property

Returns or sets a **Long** that represents the number of days that a new or recently modified file shows up in the Recently Added Files list in Reports view. For example, if the **RecentFile** property is set to 20, a new file or a file that has been modified will be classified as recent for the first 20 days of its existence. Read/write.

expression.RecentFile

Remarks

Use the **OlderFile** property to set the number of days a file exists in a Web site without being modified before it shows up in the Older Files list in Reports view.

End Sub

The following example prompts the user to enter the number of days a file can exist with the classification recent, and then sets the **RecentFile** property to that value. The subroutine SetRecent prompts the user for input, performs a validation on the input data, converts it to the correct type, and sets the **RecentFile** property to the new value. If the value is of an incorrect type, an error message is displayed to the user.

```
Sub FPRecentFile()
'Sets a value that determines how long a file can be classified rece
   Dim objApp As FrontPage.Application
  Set objApp = FrontPage.Application
  Call SetRecent(objApp)
End Sub
Sub SetRecent(ByRef objApp As Application)
'Sets the value that determines how long a file will be classified a
    Dim varNum As Variant
    Dim lngNum As Long
    'Prompt the user to enter a value
    varNum = InputBox("Enter the number of days a file " & _
                      "can exist before it is classified as old.")
    'Check to see that the value is of the correct type
    If IsNumeric(varNum) Then
       'If it's numeric, convert it to Long
       lngNum = CLng(varNum)
       'Set the RecentFile value to the new value
       objApp.RecentFile = lngNum
       'Display the new setting information to the user
       MsgBox "The RecentFile value was set correctly." & vbCr & _
              "The number of days a new or modified file will be cla
               & lngNum & "."
    Else
       'Otherwise, display an error message to the user
       MsgBox "The input value was incorrect.", vbCritical
    End If
```

Required Property

Returns or sets a **Boolean** that determines if the field is required. If the field is required, it cannot be removed from the current list. Read/write.

expression.Required

The following example displays the names and default values of all required fields in the current list. If no required fields exist or if the active Web site contains no lists, a message is displayed to the user.

```
Sub DisplayRequiredFields()
'Displays the names and default values
'of all required fields in the first list of
'the web.
    Dim objApp As FrontPage.Application
    Dim objField As ListField
    Dim objFields As ListFields
    Dim strReg As String
    Dim BlnFound As Boolean
    Set objApp = FrontPage.Application
    Set objFields = objApp.ActiveWeb.Lists.Item(0).Fields
    'set found flag to false
    BlnFound = False
    If Not ActiveWeb.Lists Is Nothing Then
        For Each objField In objFields
             'If field is required, add to list
            If objField.Required = True Then
                If strReg = "" Then
                    'if first value in string
                    strReq = objField.Name & " - " & _
                    objField.DefaultValue & vbCr
                    'The list contains at least 1 required field
                    BlnFound = True
                Else
                    'add value to string
                    strReq = strReq & objField.Name & " - " & _
                    objField.DefaultValue & vbCr
                End If
            End If
        Next objField
    Else
        'display message to user
        MsgBox "The active web contains no lists."
    End If
    If BlnFound = True Then
        MsgBox "The current list contains the following required fie
               vbCr & strReq
```

Else MsgBox "The current list contains no required field(s)." End If End Sub



ReturnType Property

Returns an **<u>FpFieldType</u>** that represents the type of field.

FpFieldType can be one of the following **FpFieldType** constants.

fpFieldAttachments Returns a ListFieldAttachments object.			
fpFieldChoice	Returns a ListFieldChoice object.		
fpFieldComputed	Returns a ListFieldComputed object.		
fpFieldCounter	Returns a ListFieldCounter object.		
fpFieldCurrency	Returns a ListFieldCurrency object.		
fpFieldDateTime	Returns a ListFieldDateTime object.		
fpFieldFile	Returns a ListFieldFile object.		
fpFieldInteger	Returns a ListFieldInteger object.		
fpFieldLookup	Returns a ListFieldLookup object.		
fpFieldMultiLine	Returns a ListFieldMultiline object.		
fpFieldNumber	Returns a ListFieldNumber object.		
fpFieldRatingScale	Returns a ListFieldRatingScale object.		
fpFieldSingleLine	Returns a ListFieldSingleLine object.		
fpFieldTrueFalse	Returns a ListFieldTrueFalse object.		
fpFieldURL	Returns a ListFieldURL object.		

expression.ReturnType

RevisionControlProject Property

Returns or sets the **RevisionControlProject** property. Read/write **String**.

expression.RevisionControlProject()

expression Required. An expression that returns a **WebEx** object.

Remarks

A revision control project can either be a Microsoft Visual SourceSafe project, or a FrontPage-based locking project. For a Visual SourceSafe control project, you must start the **RevisionControlProject** property with the string "\$/"; for a FrontPage-based locking control project, you must set the **RevisionControlProject** property to "<FrontPage-based Locking>".

To remove a source control project, set the **RevisionControlProject** property to an empty string.

The following example sets the **RevisionControlProject** property in a Visual SourceSafe project.

RootFolder Property

Returns a **WebFolder** object that represents the active **WebEx** object's root.

expression.RootFolder()

expression Required. An expression that returns a **WebEx** object.
The following statement returns the name of the root folder.

ActiveWeb.RootFolder.Name

RootNavigationNode Property

Returns a **NavigationNode** object that represents the top-level navigation node.

expression.RootNavigationNode()

expression Required. An expression that returns a **WebEx** object.

Remarks

You can use the **RootNavigationNode** property to determine the root navigation node. The **RootNavigationNode** property returns the **NavigationNode** object from which you can access all other navigation nodes in a Web site. The **RootNavigationNode** object is created by default when you create a Web site and provides the basis for the navigation structure, which is accessed through the <u>Children</u> collection. The first child node of the navigation structure is the home page of the Web site.

The following example adds a global navigation node to the right of the home navigation node.

```
Private Sub AddNode()
   Dim myRNode As NavigationNode
   Dim myPage As String
   Set myRNode = ActiveWeb.RootNavigationNode
   myPage = "http://myServer/myWeb/search.htm"
   Call myRNode.Children.Add(myPage, "Search", _
        fpStructRightmostChild)
   ActiveWeb.ApplyNavigationStructure
End Sub
```



SelectedFiles Property

Returns an array of **<u>WebFile</u>** objects representing the selected files.

expression.SelectedFiles

expression Required. An expression that returns a **WebWindowEx** object.

Remarks

You must have Folders view open in Microsoft FrontPage in order to select multiple files and you must use the right pane to select the files. From Folders view you can select multiple files in a single Web site or in multiple Web sites. If you must use Page view, you can only select one file per Web site from the left pane.

Г	ïp
---	----

You can use the **fpWebViewFolders** constant of the <u>ViewMode</u> property to set the view to the Folders view.

The following example concatenates the names of the selected files.

Note The delimiter used to separate the file names in the variable mySelName is a space.

```
Private Sub GetSelectedFileNames()
    Dim myWebWindows As WebWindowEx
    Dim mySelFiles As Variant
    Dim mySelFile As WebFile
    Dim mySelName As String
    Dim myCount As Integer
    Set myWebWindows = WebWindows
    mySelFiles = ActiveWebWindow.SelectedFiles
    For myCount = 0 To UBound(mySelFiles)
        Set mySelFile = mySelFiles(myCount)
        mySelName = mySelName & " " & mySelFile.Name
    Next
End Sub
```

SelectedFolders Property

Returns an array of **<u>WebFolder</u>** objects representing the selected folders.

expression.SelectedFolders

expression Required. An expression that returns a <u>WebWindowEx</u> object.

Remarks

You must have Folders view open in Microsoft FrontPage in order to select multiple folders and you must use the right pane to select the folders. From Folders view, you can select multiple folders in a single Web site or in multiple Web sites. If you must use Page view, you can only select one folder per Web site from the folders in the left pane.

Note If you're programmatically selecting folders, you can use the **fpWebViewFolders** constant of the **ViewMode** property to set the view to Folders view.

The following example concatenates the names of the selected folders.

Note The delimiter used to separate the folder names in the variable mySelName is a space.

```
Private Sub GetSelectedFolderNames()
    Dim myWebWindows As WebWindows
    Dim myWebWindow As WebWindowEx
    Dim mySelFolders As Variant
    Dim mySelFolder As WebFolder
    Dim mySelName As String
    Dim myCount As Integer
    Set myWebWindows = WebWindows
    mySelFolders = ActiveWebwindow.SelectedFolders
    For myCount = 0 To UBound(mySelFolders)
        Set mySelFolder = mySelFolders(myCount)
        mySelName = mySelName & " " & mySelFolder.Name
        Next
End Sub
```



SharedBorders Property

True if any shared borders are in use for the **WebEx** or **WebFile** object. Read/write **Variant**.

expression.SharedBorders(BorderIndex)

expression Required. An expression that returns a **WebEx or WebFile** object.

BorderIndex Optional **FpSharedBorders**. The border index can be one of the **FpSharedBorders** constants. The default constant is **fpBorderTop**.

FpSharedBorders	s Value	Description
fpBorderTop	1	Sets a border for the top of a page.
fpBorderLeft	2	Sets a border for the left side of a page.
fpBorderRight	4	Sets a border for the right side of a page.
fpBorderBottom	8	Sets a border for the bottom of a page.
fpBorderAll	255 or &HF	F Sets borders on all sides of a page.

Remarks

Shared borders, such as the constant **fpBorderLeft**, can be used to set individual border values.

Note The default shared border is used if a shared border is not specified.

The following example sets the shared border for the active Web site.

Private Sub SetSharedBorders() Dim myPage As PageWindowEx

ActiveWeb.SharedBorders(fpBorderLeft) = True

End Sub

ShowAsPercentage Property

Returns or sets a **Boolean** that determines if the value in the field will be displayed as a percentage. Read/write.

expression.ShowAsPercentage

expression Required. An expression that returns one of the objects in the Applies To list.

The following example sets the **ShowAsPercentage** property of all fields of type **fpFieldNumber** to **True**. The values in the fields will now appear as percentages.

Note Use the **<u>ApplyChanges</u>** method to apply any changes made to the list.

```
Sub DisplayAsPercentage()
'Displays all fields of type fpFieldNumber as
'a percentage
    Dim objApp As FrontPage.Application
    Dim objLstFlds As ListFields
    Dim strName As String
    Dim objLstFld As Object
    Set objApp = FrontPage.Application
    Set objLstFlds = objApp.ActiveWeb.Lists.Item(0).Fields
    'Cycle through lists and displays as a percentage
    For Each objLstFld In objLstFlds
        If objLstFld.Type = fpFieldNumber Then
            objLstFld.ShowAsPercentage = True
        End If
    Next objLstFld
    objApp.ActiveWeb.Lists.Item(0).ApplyChanges
```

End Sub

ShowStartupDialog Property

Returns or sets a **Boolean** that determines if the **New** task pane will be displayed when Microsoft FrontPage is started.

expression.ShowStartupDialog

expression Required. An expression that returns an <u>Application</u> object.

Remarks

The **ShowStartup** property is global, meaning that any changes will not take affect until FrontPage is restarted.

The following example sets the **ShowStartupDialog** property to **True**. The next time FrontPage is started, the **New** pane will appear.

```
Sub TaskPaneStartup()
'Modifes the Startup task pane property
Dim objApp As FrontPage.Application
Set objApp = FrontPage.Application
With objApp
    'Set to True, the task pane will appear on startup
    .ShowStartupDialog = True
End With
```

End Sub

ShowUserNamesInResults Property

Returns or sets a **Boolean** that determines if the names of users who have completed the survey will be visible. Read/write.

expression.ShowUserNamesInResults

expression Required. An expression that returns a <u>Survey</u> object.

The following example sets the **ShowUserNamesInResults** property for each **Survey** object in the active Web site to **True**, displaying the names of all users who completed the survey.

Note Use the ApplyChanges method to save any changes made to the list.

```
Sub ChangePermissions()
'Changes permission of all BasicLists in the current Web site
    Dim objApp As FrontPage.Application
    Dim objList As Object
    Dim objLists As Lists
    Set objApp = FrontPage.Application
    Set objLists = objApp.ActiveWeb.Lists
    'Cycle through each list and check for list type
    For Each objList In objLists
        'If it's a Survey then change permissions
        If objList.Type = fpListTypeSurvey Then
            If objList.ShowUserNamesInResults <> True Then
                objList.ShowUserNamesInResults = True
                objList.ApplyChanges
            End If
        End If
    Next
```

End Sub

SlowPage Property

Returns or sets a **Long** that specifies the number of seconds a page can take to download before it is classified as slow. Pages that are classified as slow appear in the Slow Pages list in the Microsoft FrontPage Reports view. Read/write.

expression.SlowPage

expression Required. An expression that returns an <u>Application</u> object.

Remarks

The **SlowPage** property is used in conjunction with the <u>ConnectionSpeed</u> property to determine the simulated download time of a Web page.

The following example sets the **SlowPage** property to 10 seconds, indicating that all pages that take over 10 seconds to download (in this case, using a 56K modem) will be classified as slow in the FrontPage Reports view.

```
Sub SetSlowPage()
'Modifies the SlowPage property
Dim objApp As FrontPage.Application
Set objApp = FrontPage.Application
With objApp
        'Set value to 10, slow pages take at least 10 seconds
        .SlowPage = 10
        'Set user's connection speed to 56K modem
        .ConnectionSpeed = fpConnect56K
End With
```

End Sub

StartNumber Property

Returns a **Long** that represents the starting number for the number scale in a rating scale field.

expression.StartNumber

expression Required. An expression that returns a **ListFieldRatingScale** object.

Subject Property

Sets or returns a **String** that represents the subject of a Web package.

expression.Subject

expression Required. An expression that returns one of the objects in the Applies To list.

The following example creates a new Web package and adds the page "test.htm" to the package, including all dependencies for the page, and then saves the new Web package.

```
Dim objWeb As WebEx
Dim objPackage As WebPackage
Set objWeb = ActiveWeb
Set objPackage = objWeb.CreatePackage("New Web Package")
With objPackage
    .Author = "John Smith"
    .Company = "Fourth Coffee"
    .Subject = "This is a new Web package for Fourth Coffee."
    .Add objWeb.Url & "/test.htm", fpDepsDefault
    .Save "c:\NewWebPackage.fwp", True
End With
```

SubTree Property

Returns a **NavigationNodes** collection object that represents a collection of all the nodes in the subtree of the current navigation node. A subtree of a node is defined as all nodes which are adjacent to or attached to the parent node and appear below the parent node in the tree hierarchy.

expression.SubTree

expression Required. An expression that returns a **NavigationNode** object.

The following example prompts the user to enter the name of a navigation node in the current Web site and returns the subtree for that particular node. If the node is found, its **label** property value is added to a **String** containing the names of all nodes in the parent node's subtree. The **String** stored in the variable strSubNodes is displayed to the user. If the node is not found in the Web site, a message is displayed to the user.

```
Sub DisplaySubTree()
'Returns the subtree of a given node
    Dim objApp As FrontPage.Application
    Dim objNavNode As NavigationNode
    Dim objNavNodes As NavigationNodes
    Dim objSubTree As NavigationNodes
    Dim objSubNode As NavigationNode
    Dim strAns As String
                              'User input
    Dim blnFound As Boolean 'Boolean found flag
    Dim intCount As Integer 'Integer counter
    Dim strSubNodes As String 'Names of all sub nodes
    blnFound = False
    intCount = 0
    Set objApp = FrontPage.Application
    'Create a reference to the NavigationNodes collection
    Set objNavNodes = objApp.ActiveWeb.AllNavigationNodes
    'Prompt the user to enter the name of the node
    strAns = InputBox("Enter the name of the node for which " & _
                       " you want to view the subtree.")
    'While the node is not found and there are more nodes in the tre
    Do While (Not blnFound = True) And (intCount <= objNavNodes.Coun
        'Compare user input with node label
        If Trim(strAns) = objNavNodes.Item(intCount).Label Then
            'If found, return node
            Set objNavNode = objNavNodes.Item(intCount)
            'Set found flag to true
            blnFound = True
        Else
            'Otherwise increase counter, keep checking
            intCount = intCount + 1
        End If
    Loop
    If blnFound = True Then
       Set objSubTree = objNavNode.SubTree
```

```
For Each objSubNode In objSubTree
            'If the string is empty or has not yet been set
           If strSubNodes = "" Then
                strSubNodes = strSubNodes & objSubNode.Label
           Else
                'otherwise add next node lable to string
                strSubNodes = strSubNodes & ", " & vbCr & objSubNode.
           End If
       Next objSubNode
       'Display names of all nodes in subtree
       MsgBox "The nodes found in the subtree of " & objNavNode.Labe
& vbCr & vbCr & strSubNodes & "."
    Else
       'If not found, display message to user
       MsgBox "The navigation node " & strAns & " was not found."
    End If
End Sub
```



SubViewMode Property

Returns or sets an **<u>FpWebSubView</u>** constant that determines the view type in the current sub window. Read/write.

FpWebSubView can be one of these FpWebSubView constants.

fpWebSubViewFolders Change the current sub window view to Folders view. **fpWebSubViewNavigation** Change the current sub window to Navigation view.

fpWebSubViewNone Close the current sub window.

expression.SubViewMode

expression Required. An expression that returns a **WebWindowEx** object.

The following example prompts the user to open the sub window in Folders view if the sub window is not currently visible. If the sub window is currently open, the user is not prompted and the program ends.

```
Sub SubModeType()
'Modifies the sub window view mode of the current web window
    Dim objApp As FrontPage.Application
    Dim objWebwdw As WebWindowEx
    Dim strAns As String
    Set objApp = FrontPage.Application
    Set objWebwdw = objApp.ActiveWebWindow
    'Check if the sub window is open or closed
    If objWebwdw.SubViewMode = fpWebSubViewNone Then
        strAns = MsgBox("The subwindow is not visible." & _
                        "Would you like to view the subwindow?", vbY
        'Prompt the user to open the subwindow
        If strAns = vbYes Then
            'Change the sub window to Folder view
            objWebwdw.SubViewMode = fpWebSubViewFolders
        End If
    End If
```

End Sub

System Property

Returns the **System** object.

expression.System()

expression Required. An expression that returns an **Application** object.

This statement returns the current operating system.

mySystem = Application.System.OperatingSystem
Template Property

Returns or sets a **String** that represents the file path of the template applied to the document library. Read/write.

expression.Template

expression Required. An expression that returns a **<u>DocumentLibrary</u>** object.

The following example creates a reference to a document library named NewLibrary and displays the file path of the template associated with the library.

```
Sub TemplatePath()
'Displays the file path of the template
Dim objApp As FrontPage.Application
Dim objLists As Lists
Dim objLibrary As DocumentLibrary
Set objApp = FrontPage.Application
Set objLists = objApp.ActiveWeb.Lists
Set objLibrary = objLists.Item("NewLibrary")
'Display message to user
MsgBox "The file path of the template associated with the " & _____"
"document library is " & objLibrary.Template & "."
```

End Sub



ThemeProperties Property

Returns the **<u>Theme</u>** object for the specified object.

expression.ThemeProperties(PropertyIndex)

expression An expression that returns a **PageWindowEx**, **WebEx**, or **WebFile** object.

PropertyIndex Optional <u>FpThemeProperties</u>. Returns or sets the theme properties.

For more detailed information on the individual Type constants shown in the following table, see the tables and lists in the **<u>ApplyTheme</u>** method.

FpThemeProperties	Value	Description
fpThemeActiveGraphics	16 or &H10	Returns the active graphics constant.
fpThemeBackgroundImage	1	Returns a background image.
fpThemeCSS	4096 or &H1000	Returns the cascading style sheet.
fpThemeDefaultSettings	16777216 or &H1000000	Returns the theme applied to the web.
fpThemeName	33554432 or &H2000000	Returns the ThemeName constant.
fpThemeNoBackgroundImage	0	Returns a background without an image.
fpThemeNoCSS	0	Returns this property if a cascading style sheet has not been set or is not wanted.
fpThemeNormalColors	0	Returns the color mode for normal color.
		Returns the graphics mode for

fpThemeNormalGraphics	0	normal graphics.
fpThemePropertiesAll	4369 or &H1111	Returns all of the theme properties. After a theme is applied to an object, the fpThemePropertiesAll property combines all the properties applied to the object.
fpThemePropertiesNone	0	Returns none for the theme properties.
fpThemeVivdColors	256	Returns the vivid colors property.

The following example checks the theme properties for active graphics. If active graphics have been applied, then vivid colors are applied in addition to the theme properties that are already applied to the active page window. If active graphics aren't applied, then active graphics and vivid colors are both applied to the active page window.

The following example adds a background picture to the specified file.

```
Private Sub AddBackgroundImage()
    Dim myFile As WebFile
    Set myFile = Webs(0).Rootfolder.Files("index.htm")
    If myFile.ThemeProperties(fpThemeBackgroundImage) = 0 Then
        myFile.ApplyTheme myFile.ThemeProperties(fpThemeName), _
        myfile.ThemeProperties(fpThemePropertiesAll) + _
        fpThemeBackgroundImage
    End If
End Sub
```

Themes Property

Returns a **Themes** collection that represents the themes available for the specified object.

expression.Themes

expression Required. An expression that returns one of the objects in the Applies to list.

Remarks

The **Themes** property as applied to the **Application** object returns the collection of themes available to be applied. When the **Themes** property is applied to the **WebEx** object, it returns the collection of themes that have been applied to the Web site. This is the same as the list that appears in the Web's _theme directory. If a theme is applied to a **WebEx** object, there will be one theme in the collection. However, if a page in a Web site has its own theme, separate from the theme that was applied to the Web site, then the **Themes** collection for the **WebEx** object will have two themes in it— the theme that was originally applied to the Web site, and the theme that was applied specifically to the page.

The following example searches for a specific theme among the themes that are available locally on the client, as well as the themes applied to the active Web site.

```
Private Sub SearchAllThemes()
    Dim myTheme As Theme
    Dim myThemeToFind As String
    Dim myIsFound As Boolean
    myThemeToFind = "blends"
    myIsFound = False
    For Each myTheme In Application. Themes
         If myTheme.Label = myThemeToFind Then
             myIsFound = True
             Exit For
         End If
    Next
    For Each myTheme In ActiveWeb.Themes
         If myTheme.Label = myThemeToFind Then
             myIsFound = True
             Exit For
         End If
    Next
End Sub
```

Title Property

Returns the title of the specified object. Read-only or read/write **String** depending on the specified object.

expression.Title

expression Required. An expression that returns one of the objects in the Applies to list.

Remarks

For the **WebEx** and **WebFile** objects the **Title** property returns the title of the active **WebEx** object as a read/write **String**.

At the same time as you're traversing the navigation nodes, you can also return the title (file name) of the file that's associated with a particular node. The following example returns the title of the file associated with a navigation node in the active Web site.

The following example shows how you can set the title of the first file in the Web site.

```
Private Sub SetTitle()
   Dim myWeb As WebEx
   Dim myNewTitle As String
   Dim myFile As WebFile
   MyNewTitle = "Inventory.htm"
   Set myWeb = ActiveWeb
   Set myFile = myWeb.RootFolder.Files(0)
   MyFile.Title = myNewTitle
End Sub
```



Type Property

Type property as it applies to the **BasicList**, **DocumentLibrary**, **List**, and **Survey** objects.

Returns an **FpListType** constant that represents the type of the current list. Read-only.

FpListType can be one of these FpListType constants. fpListTypeBasicList fpListTypeDocumentLibrary fpListTypeSurvey fpListTypeDiscussion

expression.Type

expression Required. An expression that returns one of the above objects.

Type property as it applies to all other objects in the Applies To list.

Returns an **FpFieldType** constant that represents the type of the current field. Read-only.

FpFieldType can be one of these FpFieldType constants.

fpFieldAttachments fpFieldChoice fpFieldComputed fpFieldCounter fpFieldCurrency fpFieldDateTime fpFieldFile fpFieldInteger fpFieldInteger fpFieldLookup fpFieldMultiLine fpFieldNumber fpFieldRatingScale fpFieldSingleLine fpFieldTrueFalse fpFieldURL

expression.**Type**

expression Required. An expression that returns one of the objects as mentioned above.

As it applies to the **BasicList**, **DocumentLibrary**, **List**, and **Survey** objects.

The following example displays the names of all lists in the active Web site and their associated type names. If the active Web site contains no lists, a message is displayed to the user.

```
Sub ViewListTypes()
'Displays the name of the list and
'its associated type
    Dim lstWebList As List
    Dim strType As String
    If Not ActiveWeb.Lists Is Nothing Then
    'Cycle through lists
        For Each lstWebList In ActiveWeb.Lists
            'add types to string
            If strType = "" Then
                strType = lstWebList.Name & " - " &
                lstWebList.Type & vbCr
            Else
                strType = strType & lstWebList.Name & " - " & _
                lstWebList.Type & vbCr
            End If
        Next
            'Display types of all lists in the web
            MsgBox "The list types in the current web are:" _
            & vbCr & strType
       Else
           'Otherwise display message to user
           MsgBox "The current web contains no lists."
     End If
```

End Sub

As it applies to the **ListField** object.

The following example displays the names of all fields in the first list of the **Lists** collection and their associated type names. If the active Web site contains no lists, a message is displayed to the user.

```
Sub Fieldtype()
'Displays the field types of the current list
    Dim objApp As FrontPage.Application
    Dim objField As ListField
    Dim strType As String
    Set objApp = FrontPage.Application
    If Not ActiveWeb.Lists Is Nothing Then
        For Each objField In objApp.ActiveWeb.Lists.Item(0).Fields
            If strType = "" Then
                strType = objField.Name & " - " & _
                objField.Type & vbCr
            Else
                strType = strType & objField.Name & " - " & _
                objField.Type & vbCr
            End If
        Next objField
            MsgBox "The names of the fields in this list and their t
            vbCr & strType
    Else
         'Otherwise display message to user
         MsgBox "The current Web site contains no lists."
         End If
End Sub
```

Url Property

Returns the URL for the specified object. Read-only **String**.

expression.Url

expression Required. An expression that returns one of the objects in the Applies to list.

Remarks

URLs can be absolute or relative. An absolute URL contains the exact path to the specified object while a relative URL may contain characters such as "../..", "?", or ";" depending on the URL access scheme used to parse the relative URL. The Microsoft FrontPage object model always uses absolute URLs. For more information on URLs, see <u>Understanding Absolute and Relative URLs in FrontPage</u>.

The following example returns the URL of the first file in the root folder of the active Web site.

```
Private Sub GetFileUrl()
   Dim myFile As WebFile
   Dim myURL As String
   Set myFile = ActiveWeb.RootFolder.Files(0)
   myURL = myFile.Url
```

End Sub

UserName Property

Returns the logon name of the user that is currently logged on to the network or operating system. Read/write **String**.

expression.UserName

expression Required. An expression that returns an **Application** object.

Remarks

The **UserName** property is the default parameter whenever a user name is required for a parameter in the Web Object Model. For example, the **Webs.Open** and **Webs.Publish** methods both require a **UserName** parameter. When the parameter isn't specified, the default parameter is the user name of the user currently logged on.

The following example returns the current user's logon name.

myLogonName = Application.UserName

VBE Property

Returns a **VBE** object that represents the Microsoft Visual Basic Editor.

expression.VBE

expression Required. An expression that returns an <u>Application</u> object.

The following example creates a reference to the Visual Basic Editor and displays the name of the active project to the user.

Version Property

Returns the version of the specified object with the format: "*x*.*x*.*x*", where *x* represents a number in the version. Read-only **String**.

```
expression.Version()
```

expression Required. An expression that returns one of the objects in the Applies To list.

Remarks

If the **Application** object is specified, the **Version** property returns the version of the application. When the **Version** property for the **System** object is accessed, it returns the current version of the operating system as you might see used on the **Properties** page of the **System** dialog box.

The following example returns the version of the application.

```
Public Sub GetAppVersion()
Dim myAppVersion As String
```

```
myAppVersion = Application.Version
End Sub
```

VerticalResolution Property

Returns the vertical resolution of the screen in pixels. Read-only **Long**.

expression.VerticalResolution()

expression Required. An expression that returns a **System** object.

This example returns the vertical resolution of the screen.

myVerticalRes = System.VerticalResolution



ViewMode Property (Web Object Model)

ViewMode property as it applies to the **PageWindowEx** object.

Returns or sets an **<u>FpPageViewMode</u>** constant that represents the view mode of the active page window. Read/write.

FpPageViewMode can be one of these FpPageViewMode constants.

fpPageViewNoFrames fpPageViewNormal fpPageViewNoWindow fpPageViewPreview fpPageViewDefault fpPageViewHtml

expression.ViewMode

expression Required. An expression that returns a **<u>PageWindowEx</u>** object.

ViewMode property as it applies to the **WebWindowEx** object.

Returns or sets an **<u>FpWebViewMode</u>** constant that defines the view mode of the current window. Read/write.

FpWebViewMode can be one of these FpWebViewMode constants. fpWebViewAllFiles fpWebViewBrokenLinks fpWebViewFolders fpWebViewLinks fpWebViewPage fpWebViewSiteSummary fpWebViewStructure

fpWebViewTodo

expression.ViewMode

expression Required. An expression that returns a **WebWindowEx** object.

ViewMode property as it applies to the **FPHTMLDocument** and **IFPDocument** objects.

Returns a **Long** that represents the view mode of the document. Read-only.

expression.ViewMode(ViewMode)

expression Required. An expression that returns an **FPHTMLDocument** or **IFPDocument** object.

ViewMode Required Long. The new viewing mode.

As it applies to the **PageWindowEx** object.

The following example changes the view mode of the active window to the value **fpPageViewNormal** (**Design** view) if it is not already in the default view mode.

```
Sub ChangeViewMode()
'Changes the view mode of the active window
Dim fpApp As FrontPage.Application
Dim objPage As PageWindowEx
Set fpApp = FrontPage.Application
Set objPage = fpApp.ActivePageWindow
If objPage.ViewMode <> fpPageViewDefault Then
objPage.ViewMode = fpPageViewNormal
MsgBox "The current page window has been restored " & _
"to normal view."
End If
```

End Sub

As it applies to the **WebWindowEx** object.

The following example changes the view mode of the active Web site window to **fpWebBrokenLinks** (Broken Links view) if the window is not already in the view.

```
Sub ChangeViewMode()
'Changes the view mode of the active window
Dim fpApp As FrontPage.Application
Dim objWebWindow As WebWindowEx
Set fpApp = FrontPage.Application
Set objWebWindow = fpApp.ActiveWebWindow
If objWebWindow.ViewMode <> fpWebViewBrokenLinks Then
objWebWindow.ViewMode = fpWebViewBrokenLinks
MsgBox "The current page window has been restored " &
```

"to Broken Links view."

End Sub

End If

As it applies to the **FPHTMLDocument** object.

The following example displays the view mode of the active document.

End Sub


ViewModeEx Property

Returns or sets an **<u>FpWebViewModeEx</u>** enumerated constant that represents the current view mode of the Web site window. Read/write.

FpWebViewModeEx can be one of these FpWebViewModeEx constants. **fpWebViewExAccessibility fpWebViewExAllFiles fpWebViewExAssignedTo** fpWebViewExBrokenLinks fpWebViewExBrowserTypes **fpWebViewExCategories** fpWebViewExCheckoutStatus fpWebViewExComponentErrors **fpWebViewExCSSLinks** fpWebViewExDailyPageHits fpWebViewExDailySummary **fpWebViewExFolders fpWebViewExLinks** fpWebViewExMasterPages fpWebViewExMonthlyPageHits fpWebViewExMonthlySummary **fpWebViewExNavigation fpWebViewExOlderFiles fpWebViewExOsTypes fpWebViewExPage** fpWebViewExPublishStatus fpWebViewExRecentlyAddedFiles fpWebViewExRecentlyChangedFiles fpWebViewExReferringDomains fpWebViewExReferringURLs **fpWebViewExRemoteSite**

fpWebViewExReviewStatus fpWebViewExSearchStrings fpWebViewExSharedBorders fpWebViewExSiteSummary fpWebViewExSlowPages fpWebViewExThemes fpWebViewExThemes fpWebViewExUnlinkedFiles fpWebViewExUsageSummary fpWebViewExVisitingUsers fpWebViewExWeeklyPageHits fpWebViewExWeeklySummary

expression. View Mode Ex

expression Required. An expression that returns a **WebWindowEx** object.

The following example changes the view mode to display the daily page hits.

Sub ViewPageHits()
'Changes the current view mode to view page hits

ActiveWeb.ActiveWebWindow.ViewModeEx = fpWebViewExDailyPageHits

ViewPages Property

Rreturns a **String** array that represents a list of pages that contain a view for the list.

expression.ViewPages

expression Required. An expression that returns one of the objects in the Applies To list.

Visible Property

Returns or sets the visible state of the **PageWindowEx** or **WebWindowEx** object. The visible state for the **PageWindowEx** object is read-only **Boolean**. The visible state for the **WebWindowEx** object is read/write **Boolean**.

expression.Visible

expression Required. An expression that returns one of the objects in the Applies To list.

In the following example, the GetVisibleState procedure returns the visible state of the first **PageWindowEx** object from the CheckIfVisible function.

```
Private Sub GetVisibleState()
    Dim myIsVisible As Boolean
    myIsVisible = CheckIfVisible
End Sub
Function CheckIfVisible() As Boolean
    Dim myPage As PageWindowEx
    Dim myVisibleState As Boolean
    Set myPage = Application.Webs(0).WebWindows(0).PageWindows(0)
    myVisibleState = myPage.Visible
    CheckIfVisible = myVisibleState
End Function
```

The following example sets the visible state of a **WebWindowEx** object to a windowless state.

```
Private Sub OpenInNoWindow()
   Dim myWebWindow As WebWindowEx
   Set myWebWindow = Webs(0).WebWindows(0)
   myWebWindow.Visible = False
End Sub
```

Web Property

Returns a **WebEx** object that represents a Microsoft FrontPage Web site.

expression.Web

expression Required. An expression that returns one of the above objects.

The following example returns the title of the active Web site.

```
Private Sub GetWebTitle()
   Dim myWeb As WebEx
   Dim myWebName As String
   Set myWeb = ActiveWeb
   myWebName = myWeb.RootFolder.Web.Title
```

Webs Property

Returns the collection of open Web sites for the specified object.

expression.Webs

expression Required. An expression that returns an **Application** object.

End Sub

The following example creates an array that contains all of the URLs for all the open subsites in your Web site.

Note You must open any Web sites for which you wish to retrieve the URLs.

WebWindows Property

Returns the collection of open **WebWindows** objects in the specified object.

expression.WebWindows

expression Required. An expression that returns one of the objects in the Applies To list.

The following example checks if any of the Web windows are set to Page view, and changes those that are in Page view to Folders view.

```
Private Sub GetViewModes()
   Dim myWebWindows As WebWindowEx
   Dim myWebWindow As WebWindowEx
   Dim myView As FpWebViewMode
   Set myWebs = Webs
   Set myWebWindows = myWebs.WebWindows
   For Each myWebWindow In myWebWindows
        myView = myWebWindow.ViewMode
        If myView = fpWebViewPage Then
            myWebWindow.ViewMode = fpWebViewFolders
        End If
   Next
End Sub
```

OnActivate Event

Occurs when the page window or Web window obtains the focus and becomes the current window.

Private Sub expression_OnActivate()

expression A variable which references an object of type **PageWindowEx** or **WebWindowEx** declared with events in a class module.

The following example uses the <u>IsDirty</u> property to determine if the page shown in the specified window has changed since the last refresh or save and saves the document that has changed.

```
Private Sub PageWindowEx_OnActivate()
'Displays a message when the window obtains focus
    If PageWindowEx.IsDirty Then
        PageWindowEx.Save
    End If
End Sub
```

OnAfterPageSave Event

Occurs after a page is saved.

Private Sub Application_OnAfterPageSave(ByVal *pPage* As PageWindowEx, *Success* As Boolean)

pPage Required **PageWindowEx** object.

Success Required **Boolean**. **True** if the **PageWindowEx** object was successfully saved.

Remarks

The **OnAfterPageSave** event is associated with the **Application** object. After the user saves a page or closes Microsoft FrontPage, the **OnAfterPageSave** event fires and executes the code that you specified within the event procedure.

The following example displays a message box after the page has been saved and displays the file name of the page.

Note To run this example, you must have at least one open Web site and one open page within that Web site.

Create a form called frmLaunchEvents.frm and add two buttons, a button called cmdSave and a button called cmdCance1. Add the following code to the form code window.

```
Option Explicit
Private WithEvents eFPApplication As Application
Private Sub UserForm Initialize()
    Set eFPApplication = New Application
End Sub
Private Sub cmdSave_Click()
    Dim myPageWindow As PageWindowEx
    Set myPageWindow = ActiveWeb.ActiveWebWindow.ActivePageWindow
   myPageWindow.Save
End Sub
Private Sub cmdCancel_Click()
    'Hide the form.
    frmLaunchEvents.Hide
    Exit Sub
End Sub
Private Sub eFPApplication_OnAfterPageSave(ByVal pPage As _
        PageWindow, Success As Boolean)
    If Success = True Then
        MsgBox "The following page was saved: " & pPage.File.Name
    Else
         MsgBox "There was a problem with saving your page: " & _
        pPage.File.Name
    End If
End Sub
```

OnAfterPageWindowViewChange Event

Occurs when a page window has switched view types.

Private Sub *expression_***OnAfterPageWindowViewChange**(**ByVal** *pPage* **As PageWindowEx**)

expression An object of type **Application** declared using the **WithEvents** keyword in a class module.

pPage The <u>PageWindowEx</u> object in which the view has changed.

The following example prompts the user to refresh the window after it has changed view modes.

Private Sub objApp_**OnAfterPageWindowViewChange**(ByVal pPage As PageWi 'Occurs when a view changes

OnAfterPublish Event

Occurs after a Web site is published.

Private Sub expression_OnAfterPublish(Success As Boolean)

expression A variable name which references an object of type <u>WebEx</u> declared with events in a class module.

Success A **Boolean** that determines if the publish operation was successful. If **True**, then the Web site was successfully published.

The following example displays a message to the user based on the result of the **Publish** method.

```
Private Sub objWeb_OnAfterPublish(Success As Boolean)
'Occurs after a web is published

If Success = True Then
    MsgBox "The web was published successfully."
Else
    MsgBox "An error occurred, the web was not published.", vbEx
End If
```

OnAfterSave Event

Occurs after the active document has been saved by the user.

Private Sub expression_OnAfterSave(Success As Boolean)

expression A variable name which references an object of type **<u>PageWindowEx</u>** declared using the **WithEvents** keyword in a class module.

Success A **Boolean** that indicates if the operation was successful. If **True**, the file was saved.

The event in the following example occurs after the active document is saved in an extended page window. A message is displayed to the user based on the result of the operation.

```
Private Sub PageWindowEx_OnAfterSave(Success As Boolean)
'Displays message based on value of Success

If Success = True Then
    MsgBox "The file " & PageWindowEx.ActiveDocument.Title & " w
Else
    MsgBox "The file " & PageWindowEx.ActiveDocument.Title & " w
End If
End If
```

OnAfterSubViewChange Event

Occurs after the Web window subview changes.

Private Subexpression_OnAfterSubViewChange()

expression A variable name which references an object of type <u>WebWindowEx</u> declared using the **WithEvents** keyword in a class module.

The following example displays a message to the user whenever the subview of the active Web site window changes or after the subview closes.

```
Private Sub objWebWindow_OnAfterSubViewChange()
   'Display a message indicating which view the subwindow is currently
   Select Case objwebWindow.SubViewMode
      Case fpWebSubViewFolders
           MsgBox "The view in the subwindow has changed to Folder
      Case fpWebSubViewNavigation
           MsgBox "The view in the subwindow has changed to Navigat
      Case fpWebSubViewNone
           MsgBox "The sub window has closed."
   End Select
```

OnAfterViewChange Event

Occurs after the view has changed in the active page window or Web site window.

Private Sub expression_OnAfterViewChange()

expression A variable name which references an object in the Applies To list declared using the **WithEvents** keyword in a class module.

The following example displays the name of the new page window view mode.

Private Sub PageWindowEx_OnAfterViewChange()
'Occurs when the view changes in the active window

MsgBox "The page has changed to " & PageWindowEx.ViewMode & "."

OnAfterWebPublish Event

Occurs after a Web site is published.

Private Sub *expression_***OnAfterWebPublish(ByVal** *pWeb* **As WebEx,** *Success* **As Boolean**)

expression A variable name which references an object in the Applies To list declared using the **WithEvents** keyword in a class module.

pWeb Required **WebEx**. The specified **WebEx** object.

Success Required **Boolean**. **True** if the specified Web was successfully published.

Remarks

The **OnAfterWebPublish** event is associated with the **Application** object. After the user publishes a Web site in Microsoft FrontPage, the **OnAfterWebPublish** event fires and executes the code that you specified within the event procedure.

The following example creates a property called "Published" with the value of **True** after a Web site has been published.

Note To run this example you must have one Web site open. This example uses a Web site called Rogue Cellars. You can create a Web site called Rogue Cellars, or you can substitute a Web site of your choice in the following code sample.

Create a form called frmLaunchEvents.frm and add two command buttons, a button called cmdPublishWeb, and a button called cmdCancel. Then add the following code to the form code window.

```
Option Explicit
Private WithEvents eFPApplication As Application
Private pPage As PageWindowEx
Private Sub UserForm Initialize()
    Set eFPApplication = New Application
End Sub
Private Sub cmdPublishWeb Click()
    ActiveWeb.Publish "C:\My Documents\My Web Sites\Rogue Cellars"
End Sub
Private Sub cmdCancel Click()
    'Hide the form.
    frmLaunchEvents.Hide
    Exit Sub
End Sub
Private Sub eFPApplication_OnAfterWebPublish(ByVal pWeb As WebEx, Su
    If Success = True Then
        pWeb.Properties.Add "Published", True
        pWeb.Properties.ApplyChanges
    Else
        MsgBox "There was a problem publishing your " & pWeb & " web
```

End If End Sub

OnAfterWebWindowSubViewChange Event

Occurs after the Folder List in the Web view sub window has changed from Folders view to Navigation view.

Private Sub *application*_OnAfterWebWindowSubViewChange(ByVal *pWebWindow* As WebWindowEx)

application An object of type **Application** declared with events in a class module.

pWebWindow The <u>WebWindowEx</u> object in which the view has changed.

The following example displays the name of the view type in the sub window, unless the sub window is closed by the user.

Private Sub objApp_**OnAfterWebWindowSubViewChange(ByVal** *pWebWindow* **As** 'Occurs when a sub view in the Web window changes

```
'Display a message indicating what view the sub window is curren
Select Case pWebWindow.SubViewMode
Case fpWebSubViewFolders
MsgBox "The view in the sub window has changed to Folder
Case fpWebSubViewNavigation
MsgBox "The view in the sub window has changed to Naviga
Case fpWebSubViewNone
MsgBox "The sub window has closed."
End Select
```

OnAfterWebWindowViewChange Event

Occurs after the Web site window view has changed.

Private Sub *expression_***OnAfterWebWindowViewChange(ByVal** *pWebWindow* **As WebWindowEx**)

expression An object of type **Application** declared using the **WithEvents** keyword in a class module.

pWebWindow The <u>WebWindowEx</u> object in which the view has changed.

The following example displays the name of the new view type to the user every time the view is changed.

Private Sub objApp_**OnAfterWebWindowViewChange**(ByVal pWebWindow As We 'Occurs after the Web site window view changes.

'Display message to user MsgBox "The view has changed to " & pWebWindow.ViewModeEx & " mc
OnBeforePageSave Event

Occurs before a page is saved.

Private Sub *expression_***OnBeforePageSave(ByVal** *pPage* **As PageWindowEx,** *SaveAsUI* **As Boolean,** *Cancel* **As Boolean**

expression A variable name which references an object of type **<u>PageWindowEx</u>** declared using the **WithEvents** keyword in a class module.

pPage Required **PageWindowEx**. The specified **PageWindowEx** object.

SaveAsUI Required **Boolean**. **True** when the **Save As** dialog box is used to save a page. This can be the first time the page is saved or when the page is saved as a new page. **False** when when an existing page is saved.

Cancel Required **Boolean**. Causes Microsoft FrontPage to abort the save when set to **True**. When *Cancel* is programmatically set to **True**, the user can abort the save process by clicking the Cancel button on the form. Default is **False**.

Remarks

The **OnBeforePageSave** event is associated with the **Application** object. When the user saves a page or closes FrontPage, the **OnBeforePageSave** event fires and executes the code that you specified within the event procedure.

Note If you set *Cancel* to **True**, the page won't be saved.

The following example displays a message box before the page has been saved and displays the document title of the file for the page.

Note To run this example, you must have at least one open Web site and one open page within that site.

Create a form called frmLaunchEvents.frm and add two buttons, a button called cmdSave, and a button called cmdCance1. Add the following code to the form code window.

```
Option Explicit
Private WithEvents eFPApplication As Application
Private Sub UserForm Initialize()
    Set eFPApplication = New Application
End Sub
Private Sub cmdSave Click()
    Dim myPageWindow As PageWindowEx
    Set myPageWindow = ActiveWeb.ActiveWebWindow.ActivePageWindow
    myPageWindow.Save
End Sub
Private Sub cmdCancel Click()
    'Hide the form.
    frmLaunchEvents.Hide
    Exit Sub
End Sub
Private Sub eFPApplication_OnBeforePageSave(ByVal pPage As _
        PageWindowEx, SaveAsUI As Boolean, Cancel As Boolean)
    MsgBox "The following page will be saved: " & pPage.File.Name _
        & "will be saved with the title: " & pPage.Document.Title
```

OnBeforePageWindowViewChange Event

Occurs before the current Page window view is changed.

Private Sub *application*_OnBeforePageWindowViewChange(ByVal *pPage* As PageWindowEx, ByVal *TargetView* As FpPageViewMode, *Cancel* As Boolean)

application An object of type **Application** declared using the **WithEvents** keyword in a class module.

pPage The <u>PageWindowEx</u> object in which the view has changed.

TargetView The specified target window type.

Cancel A **Boolean** that determines if the event will be cancelled. If **False**, the event will not be cancelled. If **True**, the event will be cancelled.

Remarks

This event can be cancelled.

The following example prompts the user before the current view is changed. The **Cancel** argument is modified based on the user's response.

```
Private Sub objApp_OnBeforePageWindowViewChange(ByVal pPage As PageW
ByVal TargetView As FpPageViewMode, Cancel As Boolean)
'Prompts the user before changing the view type
Dim strAns As String
'Prompt user to change view
strAns = MsgBox("Are you sure you want to change the current vie
vbYesNo)
If strAns = vbYes Then
'Don't cancel event
Cancel = False
Else
'Cancel event
Cancel = True
End Sub
```

OnBeforePublish Event

Occurs before a Web site is published.

Private Sub *expression_***OnBeforePublish**(*Destination* As String, *Cancel* As Boolean)

expression A variable name which references an object of type <u>WebEx</u> declared using the WithEvents keyword in a class module.

Destination A **String** that specifies the URL of the published Web site.

Cancel A **Boolean** that specifies whether to cancel the publish operation. **True** cancels publishing the Web site.

Remarks

This event can be cancelled.

The following example displays a message to the user before a Web site is published. It also allows the user to cancel the event before it occurs.

Private Sub expression_**OnBeforePublish**(Destination As String, Cancel 'Occurs before a web is published.

```
Dim blnAns As Boolean
blnAns = MsgBox _
        ("Are you sure you want to publish to the following destinat
        Destination)
If blnAns = False Then
        Cancel = True
Else
        Cancel = False
End If
```

OnBeforeSave Event

Occurs before a page in an active extended page window is saved.

Private Sub *expression_***OnBeforeSave**(*SaveAsUI* As Boolean, *Cancel* As Boolean)

expression A variable name that references an object of type **<u>PageWindowEx</u>** declared using the **WithEvents** keyword in a class module.

SaveAsUI A Boolean that determines if the user interface will be displayed. If **True**, the **Save As** dialog box will be displayed.

Cancel A **Boolean** that determines if the save operation will be cancelled. If **True**, the document will not be saved.

The following example prompts the user before saving the document. If the user clicks **No**, the document will not be saved. If the user clicks **Yes**, the document will be saved.

```
Private Sub PageWindowEx_OnBeforeSave(SaveAsUI As Boolean, Cancel As 'Prompt user before saving the document
```

```
Dim strAns As String
strAns = MsgBox("Do you really want to save the document?", vbYe
'Change cancel value based on user input
If strAns = VbNo Then
        Cancel = True
End If
```

OnBeforeSubViewChange Event

Occurs before the sub view of the Web site window changes.

Private Sub *expression_***OnBeforeSubViewChange**(**ByVal** *TargetSubView* **As FpWebSubView**, *Cancel* **As Boolean**)

expression A variable name which references an object of type <u>WebWindowEx</u> declared using the **WithEvents** keyword in a class module.

TargetSubView An **FpWebSubView** enumerated constant that represents the new sub view type.

Cancel A **Boolean** that determines if the operation will be cancelled. If **False**, the sub view will change view types.

The following example prompts the user before changing the sub view of the Web site window. If the user clicks **No**, the sub view will not change. If the user clicks **Yes**, the sub view will change to a new view type.



OnBeforeViewChange Event

OnBeforeViewChange event as it applies to the **PageWindowEx** object.

Occurs before the view mode of the page window changes.

Private Sub *expression_***OnBeforeViewChange(ByVal** *TargetView* **As FpPageViewMode**, *Cancel* **As Boolean**)

expression A variable name which references an object of type **<u>PageWindowEx</u>** declared using the **WithEvents** keyword in a class module.

TargetView An **FpPageViewMode** enumerated constant that represents the new view type.

Cancel A **Boolean** that determines if the operation will be cancelled. If **True**, the view will not be changed.

OnBeforeViewChange event as it applies to the **WebWindowEx** object.

Occurs before the view mode of the Web site window changes.

Private Sub *expression_***OnBeforeViewChange**(**ByVal** *TargetView* **As FpWebViewModeEx**, *Cancel* **As Boolean**)

expression A variable name which references an object of type <u>WebWindowEx</u> declared using the **WithEvents** keyword in a class module.

TargetView An **FpWebViewModeEx** enumerated constant that represents the new view type.

Cancel A **Boolean** that determines if the operation will be cancelled. If **True**, the view will not be changed.

The following example prompts the user before changing the view of the page window. If the user clicks **Yes**, the view is changed.

```
Private Sub PageWindowEx_OnBeforeViewChange(ByVal TargetView As FpPa
Cancel As Boolea
'Prompts user before changing views
Dim blnAns As Boolean
strAns = MsgBox("Are you sure you want to change the current vie
vbYesNo)
If strAns = vbYes Then
Cancel = False
Else
Cancel = True
End If
```

OnBeforeWebPublish Event

Occurs before a Web site is published.

Private Sub *expression_***OnBeforeWebPublish(ByVal** *pWeb* **As WebEx,** *Destination* **As String,** *Cancel* **As Boolean)**

expression The variable name of an object of type <u>Application</u> declared using the **WithEvents** keyword in a class module.

pWeb Required **WebEx**. The specified **WebEx** object.

Destination Required **String**. The URL of the target location.

Cancel Required **Boolean**. Causes Microsoft FrontPage to abort the publish when set to **True**. When *Cancel* is programmatically set to **True**, the user can abort the save process by clicking the Cancel button on the form. Default is **False**.

Remarks

The **OnBeforeWebPublish** event is associated with the **Application** object. When the user publishes a Web site in FrontPage, the **OnBeforeWebPublish** event fires and executes the code within the event procedure.

The following example adds a copyright string to the index page of the specified Web site.

Note To run this example, you must have at least one open Web site. This example uses a Web site called Rogue Cellars. You can create a Web site called Rogue Cellars or you can substitute a Web site of your choice in the following code sample.

Create a form called frmLaunchEvents.frm and add two buttons, a button called cmdPublishWeb, and a button called cmdCancel. Add the following code to the form code window.

```
Option Explicit
Private WithEvents eFPApplication As Application
Private pPage As PageWindowEx
Private Sub UserForm_Initialize()
    Set eFPApplication = New Application
End Sub
Private Sub cmdPublishWeb Click()
    ActiveWeb.Publish "C:\My Documents\My Web Sites\Rogue Cellars"
End Sub
Private Sub cmdCancel Click()
    'Hide the form.
    frmLaunchEvents.Hide
    Exit Sub
End Sub
Private Sub eFPApplication_OnBeforeWebPublish(ByVal pWeb As WebEx, _
        Destination As String, Cancel As Boolean)
    Dim myCopyright As String
    Dim myIndexFile As WebFile
```

```
myCopyright = "Copyright 1999 by Rogue Cellars"
Set myIndexFile = pWeb.RootFolder.Files("index.htm")
myIndexFile.Open
If myIndexFile.Application.ActiveDocument.body.outerText <> _
myCopyright Then
myIndexFile.Application.ActiveDocument.body.insertAdjacentTe
"BeforeEnd", myCopyright
End If
ActivePageWindow.Close
End Sub
```

OnBeforeWebWindowSubViewChang Event

Occurs before the sub window of the current Web window is changed by the user.

Private Sub *expression_***OnBeforeWebWindowSubViewChange(ByVal** *pwebwindow* **As WebWindowEx**, **ByVal** *TargetSubView* **As FpWebSubView**, *Cancel* **As Boolean**)

expression An object of type **Application** declared using the **WithEvents** keyword in a class module.

pWebWindow The <u>WebWindowEx</u> object that contains the sub window.

TargetSubView The sub window view type.

Cancel A **Boolean** that determines if the event will be cancelled. If **False**, the event will not be cancelled. If **True**, the event will be cancelled.

The following example prompts the user before changing the current sub window view. The **Cancel** argument is modified based on the users' response.

```
Private Sub objApp_OnBeforeWebWindowSubViewChange(ByVal pwebwindow A
ByVal TargetSubView As FpWebSubView, Cancel As Boolean)
'Occurs before the web window sub view is changed. Prompts the user
Dim strAns As String
'Prompt the user before changing views
strAns = MsgBox("Are you sure you want to change the sub window
vbYesNo)
If strAns = vbYes Then
'Yes, don't cancel the event
Cancel = False
Else
'No, cancel the event
Cancel = True
End If
```

OnBeforeWebWindowViewChange Event

Occurs before the Web site window view changes.

Private Sub *expression*_OnBeforeWebWindowViewChange(ByVal *pWebWindow* As WebWindowEx, ByVal *TargetView* As FpWebViewModeEx, *Cancel* As Boolean)

expression An object of type **Application** declared using the **WithEvents** keyword in a class module.

pWebWindow The <u>WebWindowEx</u> object that contains the view.

TargetView The **FPWebViewModeEx** window view type.

Cancel A **Boolean** that determines if the event will be cancelled. If **False**, the event will not be cancelled. If **True**, the event will be cancelled.

The following example prompts the user before changing the current view. The **Cancel** argument is modified based on the users' response.

```
Private Sub objApp_OnBeforeWebWindowViewChange(ByVal pWebWindow As w
ByVal TargetView As FpWebViewModeEx, Cancel As Boolean)
'Occurs before the view is changed in the web window. Prompts the us
Dim strAns As String
'Prompt the user before changing views
strAns = MsgBox("Are you sure you want to change the view mode?"
vbYesNo)
If strAns = vbYes Then
'Yes, don't cancel the event
Cancel = False
Else
'No, cancel the event
Cancel = True
End If
End Sub
```



OnClose Event

OnClose event as it applies to the **PageWindowEx** object.

Occurs when the active page window is closed by the user.

Private Sub expression_OnClose(Cancel As Boolean)

expression A variable name which references an object of type **PageWindowEx** declared using the **WithEvents** keyword in a class module.

Cancel A **Boolean** that determines if the operation will be cancelled. **True** cancels closing the active page window.

OnClose event as it applies to the **WebEx** object.

Occurs when the active Web site window is closed by the user.

Private Sub expression_OnClose(pCancel As Boolean)

expression A variable name which references an object of type <u>WebEx</u> declared using the **WithEvents** keyword a class module.

Cancel A **Boolean** that determines if the operation will be cancelled. **True** cancels closing the active Web site.

The following example prompts the user before closing the active page window. If the user clicks **No**, the window will not close.

OnDeactivate Event

Occurs when a user switches to another application window and the active Web site window loses the focus.

Private Sub expression_OnDeactivate()

expression A variable name which references an object of type <u>WebWindowEx</u> declared using the **WithEvents** keyword in a class module.

The following example displays a message to the user when the window is deactivated.

Private Sub objWebWindow_OnDeactivate()
'Occurs when the current web window is deactivated

MsgBox "The window has been deactivated."

OnPageClose Event

Occurs when a page is closed.

Private Sub *expression_***OnPageClose(ByVal** *pPage* **As PageWindowEx,** *Cancel* **As Boolean)**

expression The variable name of an object of type <u>Application</u> declared using the **WithEvents** keyword in a class module.

pPage Required **PageWindowEx**. The specified **PageWindowEx** object.

Cancel Required **Boolean**. Causes Microsoft FrontPage to abort the close when set to **True**. When *Cancel* is programmatically set to **True**, the user can abort the save process by clicking the Cancel button on the form. Default is **False**.

Remarks

The **OnPageClose** event is associated with the **Application** object. When the user closes a **PageWindowEx** object, the **OnPageClose** event fires and executes the code within the event procedure.

The following example uses the **IsDirty** property to check if a page has been modified, and if it has saves the page before closing it.

Note To run this example, you must have at least one open Web site and one open page within that Web site.

Create a form called frmLaunchEvents.frm and add two buttons, a button called cmdClosePage, and a button called cmdCance1. Add the following code to the form code window.

```
Option Explicit
Private WithEvents eFPApplication As Application
Private pPage As PageWindowEx
Private Sub UserForm Initialize()
    Set eFPApplication = New Application
End Sub
Private Sub cmdClosePage Click()
    ActivePageWindow.Close
End Sub
Private Sub cmdCancel_Click()
    'Hide the form.
    frmLaunchEvents.Hide
    Exit Sub
End Sub
Private Sub eFPApplication_OnPageClose(ByVal pPage As _
        PageWindowEx, Cancel As Boolean)
    If pPage.IsDirty = True Then pPage.Save
End Sub
```

OnPageNew Event

Occurs when a new page is created.

Private Sub expression_OnPageNew(ByVal pPage As PageWindowEx)

expression The variable name of an object of type <u>Application</u> declared using the **WithEvents** keyword in a class module.

pPage Required **PageWindowEx**. A **PageWindowEx** object.

Remarks

When the user creates a new page within a frameset, the **OnPageNew** event is only fired once— when the page containing the frameset tags is opened. Then Microsoft FrontPage executes the code within the event procedure.

Note The **OnPageNew** event only fires for the default frameset, even if there are more frames on the page. This event only fires if FrontPage is in Page view. If FrontPage is in any other view, the **OnPageNew** event won't fire.
The following example applies a theme to a new page.

Note To run this example, you must have at least one open Web site. This example uses Rogue Cellars as the specified Web site and Zinfandel.htm as the specified page. You can create a Web site and page using these names or you can substitute a Web site and page of your choice.

Create a form called frmLaunchEvents.frm and add two buttons, a button called cmdAddPage, and a button called cmdCance1. Add the following code to the form code window.

```
Option Explicit
Private WithEvents eFPApplication As Application
Private pPage As PageWindowEx
Private Sub UserForm Initialize()
    Set eFPApplication = New Application
End Sub
Private Sub cmdAddPage_Click()
    Dim myPageWindows As PageWindows
    Dim myFile As String
    Set myPageWindows = ActiveWeb.ActiveWebWindow.PageWindows
    myFile =
        "C:/My Documents/My Web Sites/Rogue Cellars/Zinfandel.htm"
   myPageWindows.Add (myFile)
End Sub
Private Sub cmdCancel Click()
    'Hide the form.
    frmLaunchEvents.Hide
    Exit Sub
End Sub
```

OnPageOpen Event

Occurs when a page is opened.

Private Sub *expression_***OnPageOpen(ByVal** *pPage* **As PageWindowEx)**

expression The variable name of an object of type <u>Application</u> declared using the **WithEvents** keyword in a class module.

pPage Required **PageWindowEx**. A **PageWindowEx** object.

Remarks

The **OnPageOpen** event is associated with the **Application** object. When the user opens a page, Microsoft FrontPage opens the frameset for the page and fires the **OnPageOpen** event for the default frameset. Then FrontPage executes the code that you specified within the event procedure.

Note The **OnPageOpen** event only fires for the default frameset, even if there are more frames on the page. The **OnPageOpen** event only fires if the page is not open.

The following example changes the title of the **FPHTMLDocument** object, when the document is opened in a **PageWindowEx** object.

Note To run this example, you must have at least one open Web site. This example uses Rogue Cellars as the specified Web site and Zinfandel.htm as the specified page. You can create a Web site and page using these names or you can substitute a Web site and page of your choice.

Create a form called frmLaunchEvents.frm and add two buttons, a button called cmdAddPage, and a button called cmdCance1. Add the following code to the form code window.

```
Option Explicit
Private WithEvents eFPApplication As Application
Private Sub UserForm_Initialize()
    Set eFPApplication = New Application
End Sub
Private Sub cmdAddPage_Click()
    Dim myPageWindows As PageWindows
    Dim myFile As String
    Set myPageWindows = ActiveWeb.ActiveWebWindow.PageWindows
    m∨File =
        "C:/My Documents/My Web Sites/Rogue Cellars/Zinfandel.htm"
   myPageWindows.Add (myFile)
End Sub
Private Sub cmdCancel Click()
    'Hide the form.
    frmLaunchEvents.Hide
    Exit Sub
End Sub
```

Private Sub eFPApplication_OnPageOpen(ByVal pPage As _

```
PageWindowEx)
Dim myDoc As FPHTMLDocument
```

```
Set myDoc = pPage.ActiveDocument
```

myDoc.Title = "Rogue Cellars Home Page"
End Sub

OnPageWindowActivate Event

Occurs when the page in the current window obtains the focus.

Private Sub *expression_***OnPageWindowActivate(ByVal** *pPage* **As PageWindowEx**)

expression The variable name of an object of type <u>Application</u> declared using the **WithEvents** keyword in a class module.

pPage The <u>PageWindowEx</u> object that contains the view.

The following example prompts the user to refresh the page each time the page window obtains the focus.

Private Sub objApp_**OnPageWindowActivate**(ByVal pPage As PageWindowEx) 'Occurs when current page in the main window obtains focus. 'Prompts the user to refresh the current page.

End Sub

OnRecalculateHyperlinks Event

Occurs before the hyperlink structure in Hyperlinks view is recalculated to view any changes made to the Web site.

Private Sub *expression_***OnRecalculateHyperlinks**(**ByVal** *pWeb* **As WebEx**, *Cancel* **As Boolean**)

expression An object of type <u>Application</u> declared using the **WithEvents** keyword in a class module.

pWeb The <u>WebEx</u> object that contains the view.

Cancel A **Boolean** that determines if the event will be cancelled. If **False**, the event will not be cancelled. If **True**, the event will be cancelled.

The following example prompts the user before recalculating the hyperlink structure. If the user accepts, the event will continue and the hyperlinks will be recalculated.

End Sub

OnWebClose Event

Occurs when a Web site is closed.

Private Sub *expression_***OnWebClose(ByVal** *pWeb* **As WebEx,** *Cancel* **As Boolean)**

expression An object of type <u>Application</u> declared using the **WithEvents** keyword in a class module.

pWeb Required **WebEx**. A **WebEx** object.

Cancel Required **Boolean**. **True** if the closing process was cancelled through the user interface, or if *Cancel* was set to **True**. Default is **False**.

Remarks

The **OnWebClose** event is associated with the **Application** object. When you close a Web site, the **OnWebClose** event fires and executes the code that you specified within the event procedure.

The following example iterates through the open pages and, if necessary, saves them before the Web site is closed.

Note To run this example, you must have at least one open Web site and one open page within that Web site. This example uses Rogue Cellars as the specified Web site. You can create a Web site called Rogue Cellars or substitute a Web site of your choice.

Create a form called frmLaunchEvents.frm and add two buttons, a button called cmdCloseWeb, and a button called cmdCancel. Add the following code to the form code window.

```
Option Explicit
Private WithEvents eFPApplication As Application
Private pPage As PageWindowEx
Private Sub UserForm_Initialize()
    Set eFPApplication = New Application
End Sub
Private Sub cmdCloseWeb Click()
   Webs("C:/My Documents/My Web Sites/Rogue Cellars").Close
End Sub
Private Sub cmdCancel Click()
    'Hide the form.
    frmLaunchEvents.Hide
    Exit Sub
End Sub
Private Sub eFPApplication_OnWebClose(ByVal pWeb As WebEx, _
        Cancel As Boolean)
    Dim myPageWindows As PageWindows
    Dim myPageWindow As PageWindowEx
```

```
Set myPageWindows = pWeb.ActiveWebWindow.PageWindows
For Each myPageWindow In myPageWindows
    If myPageWindow.IsDirty = True Then myPageWindow.Save
    Next
End Sub
```



OnWebFileCheckOut Event

Occurs when a file in the current Web site is checked out.

Private Sub *expression_***OnWebFileCheckOut(ByVal** *pWeb* **As Web, ByVal** *pFile* **As WebFile**, *CheckedOut* **As Boolean**, *pCheckOutOption* **As FpCheckOutOption**)

expression The variable name that references an object of type <u>Application</u> declared using the **WithEvents** keyword in a class module.

pWeb The current <u>WebEx</u> object that contains the file.

pFile The <u>WebFile</u> being checked out.

CheckedOut A **Boolean** that indicates the status of the file. If **True**, the file is checked out. If **False**, the file has not been checked out.

pCheckOutoption <u>FpCheckOutOption</u>. Specifies the checkout file option.

FpCheckOutOption can be one of these FpCheckOutOption constants.
FPCheckOut Default. Check the file out.
FPCheckOutCancel Cancel the file check out.
FPCheckOutReadOnly Checks out a read-only version of the file.
FPCheckOutPromptUser Prompts the user before checking out the file.

The following example prompts the user before the file is checked out.

'Prompt the user before checking out the file
pCheckOutOption = fpCheckOutPromptUser

End Sub

OnWebNew Event

Occurs when a new Web site is created.

Private Sub expression_OnWebNew(ByVal pWeb As Web)

expression The variable name that references an object of type <u>Application</u> declared using the **WithEvents** keyword in a class module.

pWeb Required **WebEx**. A **WebEx** object.

Remarks

The **OnWebNew** event is associated with the **Application** object. When the user creates a new Web site in Microsoft FrontPage, the **OnWebNew** event fires and executes the code within the event procedure.

The following example creates a temporary Web site and adds a new file.

Create a form called frmLaunchEvents.frm and add two buttons, a button called cmdCreateWeb, and a button called cmdCance1. Add the following code to the form code window.

```
Option Explicit
Private WithEvents eFPApplication As Application
Private pPage As PageWindowEx
Private Sub UserForm_Initialize()
    Set eFPApplication = New Application
End Sub
Private Sub cmdCreateWeb_Click()
   Webs.Add ("C:/My Documents/My Web Sites/TempWeb")
End Sub
Private Sub cmdCancel_Click()
    'Hide the form.
    frmLaunchEvents.Hide
    Exit Sub
End Sub
Private Sub eFPApplication_OnWebNew(ByVal pWeb As Web)
    Dim myFile As WebFile
    Set myFile = pWeb.RootFolder.Files.Add("index.htm")
   myFile.Open
End Sub
```

OnWebOpen Event

Occurs when a Web site is opened.

Private Sub expression_OnWebOpen(ByVal pWeb As Web)

expression The variable name that references an object of type <u>Application</u> declared using the **WithEvents** keyword in a class module.

pWeb Required **WebEx**. A **WebEx** object.

Remarks

The **OnWebOpen** event is associated with the **Application** object. When the user opens a Web site in Microsoft FrontPage, the **OnWebOpen** event fires and executes the code that you specified within the event procedure.

The following example opens the Index.htm file when a Web site is opened.

Note This example uses Rogue Cellars as the specified Web site to be opened. You can create a Web site called Rogue Cellars or you can substitute a Web site of your choice.

Create a form called frmLaunchEvents.frm and add two buttons, a button called cmdOpenWeb, and a button called cmdCancel. Add the following code to the form code window.

```
Option Explicit
Private WithEvents eFPApplication As Application
Private pPage As PageWindowEx
Private Sub UserForm Initialize()
    Set eFPApplication = New Application
End Sub
Private Sub cmdOpenWeb Click()
   Webs.Open ("C:/My Documents/My Web Sites/Rogue Cellars")
End Sub
Private Sub cmdCancel_Click()
    'Hide the form.
    frmLaunchEvents.Hide
    Exit Sub
End Sub
Private Sub eFPApplication_OnWebOpen(ByVal pWeb As Web)
    Dim myFile As WebFile
    Set myFile = pWeb.RootFolder.Files.Add("index.htm")
    myFile.Open
End Sub
```

WindowActivate Event

Occurs when a window is activated.

Private Sub *expression*_**WindowActivate**(**ByVal** *pWebWindow* **As WebWindowEx**)

expression The variable name of an object of type <u>Application</u> declared using the **WithEvents** keyword in a class module.

pWebWindow The <u>WebWindowEx</u> object.

Remarks

When an instance of Microsoft FrontPage obtains the focus, a **WindowActivate** event will fire for each open window.

The following example prompts the user to close the window when the FrontPage window obtains the focus.

Private Sub expression_WindowActivate(ByVal pWebWindow As WebWindowE 'Occurs when a FrontPage window obtains focus.

End Sub

WindowDeactivate Event

Occurs when a Web window is deactivated.

Private Sub *expression_***WindowDeactivate**(**ByVal** *pWebWindow* **As WebWindow**Ex)

expression The variable name of an object of type <u>Application</u> declared with events in a class module.

pWebWindow The <u>WebWindowEx</u> object.

The following example prompts the user to close the window when the window loses the focus.

Private Sub expression_WindowDeactivate(ByVal pWebWindow As WebWindo 'Occurs when a Microsoft FrontPage window loses focus.

End Sub



FrontPage Web Object Model Enumerated Constants

This topic provides a list of all enumerated constants in the Microsoft FrontPage Web Object Model.

FpCheckOutOption

Constant	Value
fpCheckOut	1
fpCheckOutCancel	3
fpCheckOutPromptUser	0
fpCheckOutReadOnly	2

FpChoiceFieldFormat

Constant	Value
fpChoiceFieldCheckBoxes	2
fpChoiceFieldDropdown	0
fpChoiceFieldRadioButtons	1

FpConnectionSpeed

Constant	Value
fpConnect1000K	9
fpConnect128K	3
fpConnect144	0
fpConnect1500K	4
fpConnect256K	6
fpConnect288	1
fpConnect384K	7
fpConnect512K	8

 fpConnect56K
 2

 fpConnectISDN
 3

 fpConnectT1
 4

 fpConnectT3
 5

FpCurrencyFieldFormat

Constant	Value
fpCurrencyFieldAlgerian	5121
fpCurrencyFieldArgentina	11274
fpCurrencyFieldAustralia	3081
fpCurrencyFieldAustria	3079
fpCurrencyFieldBahraini	15361
fpCurrencyFieldBelgiumBF	2067
fpCurrencyFieldBelgiumFB	2060
fpCurrencyFieldBolivia	16394
fpCurrencyFieldBrazil	1046
fpCurrencyFieldCanada	4105
fpCurrencyFieldCanadaF	3084
fpCurrencyFieldChile	13322
fpCurrencyFieldColombia	9226
fpCurrencyFieldCostaRica	5130
fpCurrencyFieldCzech	1029
fpCurrencyFieldDenmark	1030
fpCurrencyFieldDominicanRepublic	7178
fpCurrencyFieldEcuador	12298
fpCurrencyFieldEgyptian	3073
fpCurrencyFieldElSalvador	17418
fpCurrencyFieldEuro	-1
fpCurrencyFieldEuroPostfix	-2
fpCurrencyFieldFinland	1035
fpCurrencyFieldFrance	1036
fpCurrencyFieldGermany	1031

fpCurrencyFieldGreece	1032
fpCurrencyFieldGuatemala	4106
fpCurrencyFieldHonduras	18442
fpCurrencyFieldHongKong	3076
fpCurrencyFieldHungary	1038
fpCurrencyFieldIndian	1081
fpCurrencyFieldIranian	1065
fpCurrencyFieldIraqi	2049
fpCurrencyFieldIreland	6153
fpCurrencyFieldIsraeli	1037
fpCurrencyFieldItaly	1040
fpCurrencyFieldJapan	1041
fpCurrencyFieldJordanian	11265
fpCurrencyFieldKorea	1042
fpCurrencyFieldKuwaiti	13313
fpCurrencyFieldLebanese	12289
fpCurrencyFieldLibyan	4097
fpCurrencyFieldMexico	2058
fpCurrencyFieldMoroccan	6145
fpCurrencyFieldNetherlands	1043
fpCurrencyFieldNewZealand	5129
fpCurrencyFieldNicaragua	19466
fpCurrencyFieldNorway	1044
fpCurrencyFieldOmani	8193
fpCurrencyFieldPakistani	1056
fpCurrencyFieldPanama	6154
fpCurrencyFieldParaguay	15370
fpCurrencyFieldPeru	10250
fpCurrencyFieldPoland	1045
fpCurrencyFieldPortugal	2070
fpCurrencyFieldPRChina	2052
fpCurrencyFieldQatari	16385
fpCurrencyFieldRussia	1049

fpCurrencyFieldSaudiArabian	1025
fpCurrencyFieldSingapore	4100
fpCurrencyFieldSlovakia	1051
fpCurrencyFieldSlovenia	1060
fpCurrencyFieldSouthAfrica	7177
fpCurrencyFieldSpain	1034
fpCurrencyFieldSweden	1053
fpCurrencyFieldSwitzerland	2055
fpCurrencyFieldSyrian	10241
fpCurrencyFieldTaiwan	1028
fpCurrencyFieldThai	1054
fpCurrencyFieldTunisian	7169
fpCurrencyFieldTurkey	1055
fpCurrencyFieldUAE	14337
fpCurrencyFieldUnitedKingdom	2057
fpCurrencyFieldUnitedStates	1033
fpCurrencyFieldUruguay	14346
fpCurrencyFieldVenezuela	8202
fpCurrencyFieldVietnamese	1066
fpCurrencyFieldYemeni	9217

FpDateTimeFieldFormat

Constant	Value
fpDateTimeFieldDateAndTime	0
fpDateTimeFieldDateOnly	1

FpDependencyFlags

Constant	Value
fpDepsDefault	255
fpDepsImages	2
fpDepsLinkbars	16
fpDepsLinks	1

fpDepsLists	4
fpDepsNavbars	32
fpDepsNone	0
fpDepsRecurse	256
fpDepsSharedBorders	128
fpDepsThemes	8
fpDepsWebParts	64

<u>FpFieldType</u>

Constant	Value
fpFieldAttachments	13
fpFieldChoice	5
fpFieldComputed	11
fpFieldCounter	9
fpFieldCurrency	3
fpFieldDateTime	4
fpFieldFile	12
fpFieldInteger	10
fpFieldLookup	6
fpFieldMultiLine	1
fpFieldNumber	2
fpFieldRatingScale	14
fpFieldSingleLine	0
fpFieldTrueFalse	7
fpFieldURL	8

FpFolderType

ConstantValuefpFolderDHTMLSharedLib1

FpListDesignSecurity

Constant Value

fpListDesignSecurityCreator 1 fpListDesignSecurityEveryone 0

FpListEditSecurity

Constant	Value
fpListEditSecurityAll	0
fpListEditSecurityNone	2
fpListEditSecurityOnlyOwn	1

FpListReadSecurity

Constant	Value
fpListReadSecurityAll	0
fpListReadSecurityOnlyOwn	1

FpListType

Constant	Value
fpListTypeBasicList	0
fpListTypeDiscussion	3
fpListTypeDocumentLibrary	2
fpListTypeSurvey	1

FpNumberFieldFormat

Constant	Value
fpNumberFieldAuto	-1
fpNumberFieldFiveDecimals	5
fpNumberFieldFourDecimals	4
fpNumberFieldInteger	0
fpNumberFieldOneDecimal	1
fpNumberFieldThreeDecimals	3
fpNumberFieldTwoDecimals 2

FpOptimizeHTMLFlags

Constant	Value
fpHtmlOptAdjacentTags	64
fpHtmlOptAuthorComponents	8
fpHtmlOptBots	8
fpHtmlOptBrowseComponents	131072
fpHtmlOptCellFormattingAttr	256
fpHtmlOptDreamWeaver	2
fpHtmlOptDwtCmnts	1024
fpHtmlOptEmpty	32
fpHtmlOptGenerator	16
fpHtmlOptHTMLAllWhitespace	16384
fpHtmlOptHTMLCmnts	4096
fpHtmlOptHTMLLeadWhitespace	8192
fpHtmlOptHTMLMisnest	32768
fpHtmlOptOn	-2147483648
fpHtmlOptScriptCmnts	2048
fpHtmlOptThemes	4
fpHtmlOptTrcImageAttr	512
fpHtmlOptUnusedStyles	65536
fpHtmlOptVMLGraphics	128
fpHtmlOptWordHTML	1

FpPageViewMode

Constant	Value
fpPageViewDefault	0
fpPageViewHtml	2
fpPageViewInBetween	512
fpPageViewInvalid	-1
fpPageViewNoFrames	4

fpPageViewNormal	1
fpPageViewNoWindow	128
fpPageViewPreview	8
fpPageViewPrintPreview	256
fpPageViewSplit	16
fpPageViewText	32
fpPageViewXml	64

FpPkgImportConflictOpts

Constant	Value
fpPkgFileConflictMask	15
fpPkgListConflictMask	240
fpPkgOnConflictSkip	17
fpPkgOnConflictStop	0
fpPkgOnFileConflictOverwrite	2
fpPkgOnFileConflictSkip	1
fpPkgOnFileConflictStop	0
fpPkgOnListConflictMergeOrRename	64
fpPkgOnListConflictMergeOrSkip	48
fpPkgOnListConflictMergeOrStop	32
fpPkgOnListConflictRename	80
fpPkgOnListConflictSkip	16
fpPkgOnListConflictStop	0

FpPkgImportResult

Constant	Value
fpPkgImportCancelled	2
fpPkgImportComplete	0
fpPkgImportErrorInPackage	3
fpPkgImportFailed	1
fpPkgImportNotTrusted	5
fpPkgImportServerNotSupported	6

fpPkgImportStopped

FpPkgTrustLevel

4

Constant	Value
fpPkgTrustAll	1
fpPkgTrustCertificateStore	2

FpSharedBorders

Constant	Value
_fpBorderNone	0
fpBorderAll	255
fpBorderBottom	8
fpBorderLeft	2
fpBorderRight	4
fpBorderTop	1

FpStructModType

 Constant
 Value

 fpStructBaseOnSibling
 0

 fpStructLeftmostChild
 1

 fpStructRightmostChild
 2

FpThemeProperties

Constant	Value
fpThemeActiveGraphics	16
fpThemeBackgroundImage	1
fpThemeCSS	4096
fpThemeDefaultSettings	16777216
fpThemeName	33554432
fpThemeNoBackgroundImage	0

fpThemeNoCSS	0
fpThemeNormalColors	0
fpThemeNormalGraphics	0
fpThemePropertiesAll	4369
fpThemePropertiesNone	0
fpThemeVividColors	256

FpURLComponent

Constant	Value
fpURLComponentBookmark	256
fpURLComponentFileExtension	128
fpURLComponentFileName	64
fpURLComponentPassword	4
fpURLComponentPath	32
fpURLComponentPort	16
fpURLComponentQuery	512
fpURLComponentScheme	1
fpURLComponentServer	8
fpURLComponentUserName	2

FpURLFieldFormat

ConstantValuefpURLFieldImage1fpURLFieldLink0

FpWebDeleteFlags

ConstantValuefpDeleteEntireWeb0fpDeleteFrontPageInfoFromWeb1

FpWebOpenFlags

Constant Value fpOpenInWindow 0 fpOpenNoWindow 2

FpWebPublishFlags

Value
2
64
4
1
8
16
0
128
256
32
512
1024

FpWebSubView

Constant	Value
fpWebSubViewFolders	1
fpWebSubViewNavigation	2
fpWebSubViewNone	0

FpWebViewMode

Constant	Value
fpWebViewAllFiles	4
fpWebViewBrokenLinks	6
fpWebViewFolders	1
fpWebViewLinks	0

 fpWebViewPage
 3

 fpWebViewRemoteSite
 8

 fpWebViewSiteSummary
 7

 fpWebViewStructure
 2

 fpWebViewTodo
 5

FpWebViewModeEx

Constant	Value
fpWebViewExAccessibility	37
fpWebViewExAllFiles	4
fpWebViewExAssignedTo	13
fpWebViewExBrokenLinks	6
fpWebViewExBrowserTypes	29
fpWebViewExCategories	17
fpWebViewExCheckoutStatus	16
fpWebViewExComponentErrors	18
fpWebViewExCSSLinks	33
fpWebViewExDailyPageHits	19
fpWebViewExDailySummary	22
fpWebViewExFolders	1
fpWebViewExLinks	0
fpWebViewExMasterPages	32
fpWebViewExMonthlyPageHits	21
fpWebViewExMonthlySummary	24
fpWebViewExNavigation	2
fpWebViewExOlderFiles	8
fpWebViewExOsTypes	28
fpWebViewExPage	3
fpWebViewExPublishStatus	15
fpWebViewExRecentlyAddedFiles	9
fpWebViewExRecentlyChangedFiles	10
fpWebViewExReferringDomains	25

fpWebViewExReferringURLs	26
fpWebViewExRemoteSite	34
fpWebViewExReviewStatus	14
fpWebViewExSearchStrings	27
fpWebViewExSharedBorders	36
fpWebViewExSiteSummary	7
fpWebViewExSlowPages	11
fpWebViewExThemes	35
fpWebViewExTodo	5
fpWebViewExUnlinkedFiles	12
fpWebViewExUsageSummary	31
fpWebViewExVisitingUsers	30
fpWebViewExWeeklyPageHits	20
fpWebViewExWeeklySummary	23