- <u>Show All</u>

Microsoft Office Objects

AnswerWizard ^LAnswerWizardFiles Assistant Balloon **•** BalloonCheckboxes BalloonCheckbox BalloonLabels -BalloonLabel **COMAddIns** COMAddIn CommandBarButton CommandBarControl **CommandBarComboBox** CommandBarControl **CommandBarPopup** CommandBarControl **DocumentProperties** DocumentProperty **FileDialog** <u>FileDialogFilters</u> ▶ ^L<u>FileDialogFilter</u> -<u>FileDialogSelectedItems</u> **FileSearch** FileTypes -<u>FoundFiles</u> ⊢<u>PropertyTests</u> ▶ PropertyTest

SearchFolders <u>ScopeFolder</u> ▶ <u>ScopeFolders</u> SearchScopes -<u>SearchScope</u> ▶ -<u>ScopeFolder</u> ▶ <u>ScopeFolders</u> **HTMLProject** HTMLProjectItems -<u>HTMLProjectItem</u> **LanguageSettings MsoEnvelope** CommandBars **NewFile OfficeDataSourceObject** -<u>ODSOColumns</u> » <u>ODSOColumn</u> ODSOFilters <u>ODSOFilter</u> **Scripts** <u>Script</u> **SignatureSet** Signature **WebPageFonts** -<u>WebPageFont</u>

Legend

Object and collection Object only

What's New for Microsoft Office Developers

Extensive changes have been made to the Microsoft Office Visual Basic object model to support new and improved features in shared components. Many objects, properties, methods, and events have been added.

Visit the <u>Office Developer Center</u> at MSDN Online for the latest Microsoft Office development information, including new technical articles, downloads, samples, product news, and more.

New Language Elements

The following topics provide lists of language elements that are new in Office:

New Objects

New Properties (by Object)

New Properties (Alphabetic List)

New Methods

New Events

New Objects

Visit the <u>Office Developer Center</u> at MSDN Online for the latest Microsoft Office development information, including new technical articles, downloads, samples, product news, and more.

Objects that were added to Visual Basic in Microsoft Office are listed in the following table.

FileDialog FileDialogFilter FileDialogFilters FileDialogSelectedItems FileTypes MsoEnvelope NewFile ODSOColumn ODSOColumns ODSOFilter ODSOFilters OfficeDataSourceObject ScopeFolder ScopeFolders SearchFolders SearchScope SearchScopes Signature SignatureSet

Objects

New Events

New events in Microsoft Office are listed in the following table.

	Events
EnvelopeHide	
EnvelopeShow	

New Methods

Methods that have been added to existing objects in Microsoft Office are listed in the following table, sorted by object.

Objects	Methods
<u>Assistant</u>	DoAlert
FileSearch	<u>RefreshScopes</u>
	<u>ApplyFilter</u>
OfficeDataSourceObject	<u>SetSortOrder</u>
ScopeFolder	AddToSearchFolders
<u>SignatureSet</u>	<u>Commit</u>

New Properties (Alphabetic List)

Properties that have been added to existing objects in Microsoft Office are listed in the following table (sorted alphabetically).

Properties AllowMultiSelect AttachCertificate ButtonName Column Columns CommandBars CompareTo Comparison Conjunction ConnectString DataSource DialogType DisableAskAQuestionDropdown DisableCustomize ExpireDate Extensions FileTypes FilterIndex Filters InitialFileName InitialView Introduction IsCertificateExpired IsCertificateRevoked

Issuer IsValid Mask Path Path Picture RowCount ScopeFolders ScopeFolders SearchFolders SearchFolders SelectedItems SignDate Signer

New Properties (by Object)

Properties that have been added to existing objects in Microsoft Office are listed in the following table (sorted by object name).

Objects	Properties
CommandBarButton	<u>Mask</u> <u>Picture</u>
<u>CommandBars</u>	DisableAskAQuestionDropdown
	DisableCustomize
	<u>AllowMultiSelect</u>
FileDialog	ButtonName
	DialogType
	<u>FilterIndex</u>
	<u>Filters</u>
	<u>InitialFileName</u>
	<u>InitialView</u>
	SelectedItems
FileDialogFilter	Extensions
	<u>FileTypes</u>
<u>FileSearch</u>	SearchFolders
	SearchScopes

	CommandBars
<u>MsoEnvelope</u>	Introduction
	<u>Column</u>
	<u>CompareTo</u>
ODSOFilter	<u>Comparison</u>
	Conjunction
	<u>Columns</u>
	ConnectString
	DataSource
OfficeDataSourceObject	Filters
	RowCount
	<u>Table</u>
	Path
<u>ScopeFolder</u>	ScopeFolders
<u>SearchScope</u>	<u>ScopeFolder</u>
	<u>AttachCertificate</u>
	ExpireDate
	IsCertificateExpired
	IsCertificateRevoked
<u>Signature</u>	<u>Issuer</u>
	<u>IsValid</u>

<u>SignDate</u>

<u>Signer</u>

AnswerWizard Object

AnswerWizard (AnswerWizardFiles)

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

Represents the Answer Wizard in a Microsoft Office application. There's only one Answer Wizard per application, and all changes to the **AnswerWizard** or the **AnswerWizardFiles** collection affect the active Office application immediately.

Using the AnswerWizard Object

Use the **ClearFileList** method to remove all entries from the list of files available to the current Answer Wizard. Using this method ensures that the default files available to the Office host application are no longer accessible through the Answer Wizard, such as when you're replacing the Answer Wizard files with custom .AW files. The following example clears the file list for the default Answer Wizard and then adds two files to the custom Answer Wizard.

```
customAnswerWizard.ClearFileList
customAnswerWizard.Files.Add ("c:\awfiles\custom_1.aw")
customAnswerWizard.Files.Add ("c:\awfiles\custom_2.aw")
```

Use the **ResetFileList** method to restore the list of files for the current Answer Wizard to the default list of files for the Office host application. You can also establish a custom default file list in the Windows registry by adding the names of the custom files to the appropriate registry key; the files specified in that registry key will then be restored when **ResetFileList** is called. This example resets the file list for the current Answer Wizard.

```
customAnswerWizard.ResetFileList
```

Use the **Files** property to get the collection of Answer Wizard file references. The **Files** property returns a collection of strings that refer to .AW files. The following example returns the **AnswerWizardFiles** collection and displays the file count in a message box.

```
Dim customAnswerWizardFiles As AnswerWizardFiles
Set customAnswerWizardFiles = Application.AnswerWizard.Files
```

MsgBox customAnswerWizardFiles.Count

AnswerWizardFiles Collection Object

AnswerWizard (AnswerWizardFiles)

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

A collection of references to Answer Wizard files. The **AnswerWizardFiles** collection contains all of the Answer Wizard files (with the file name extension .AW) available to the active Microsoft Office application.

Using the AnswerWizardFiles Collection

Use the **Files** property to get the collection of Answer Wizard file references. The **Files** property returns a collection of strings that refer to .AW files. The following example returns the **AnswerWizardFiles** collection and displays the file count in a message box.

```
Dim customAnswerWizardFiles As AnswerWizardFiles
Set customAnswerWizardFiles = Application.AnswerWizard.Files
```

```
MsgBox customAnswerWizardFiles.Count
```

Use the **Add** method to make additional files available to the current Answer Wizard. The following example adds the file Custom_1.aw to the list of Answer Wizard files in the active Office application.

```
Dim customAnswerWizard As AnswerWizard
Set customAnswerWizard = Application.AnswerWizard
```

```
customAnswerWizard.Files.Add ("c:\awfiles\custom_1.aw")
```

Use the **Item** property to get the name of an existing Answer Wizard file reference. The following example displays a message box containing the name of the file referred to by Item(1).

MsgBox customAnswerWizard.Files.Item(1)

Assistant Object

<u>Assistant</u> <u>Balloon</u> <u>BalloonCheckBoxes (BalloonCheckBox)</u> <u>BalloonLabels (BalloonLabel)</u>

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

Represents the Microsoft Office Assistant.

Using the Assistant Object

Use the **Assistant** property to return the **Assistant** object. There isn't a collection for the **Assistant** object; only one **Assistant** object can be active at a time. Use the **Visible** property to display the Assistant, and use the **On** property to enable the Assistant.

Remarks

The default Assistant is Rocky. To select a different Assistant programmatically, use the **FileName** property.

The following example displays and animates the Assistant.

```
With Assistant
.Visible = True
.Animation = msoAnimationGreeting
End With
```

Balloon Object

<u>Assistant</u> <u>Balloon</u> <u>BalloonCheckBoxes (BalloonCheckBox)</u> <u>BalloonLabels (BalloonLabel)</u>

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

Represents the balloon where the Office Assistant displays information. A balloon can contain controls such as check boxes and labels.

Using the Balloon Object

Use the **NewBalloon** property to return a **Balloon** object. There isn't a collection for the **Balloon** object; only one balloon can be visible at a time. However, it's possible to define several balloons and display any one of them when needed. For more information, see "Defining and Reusing Balloons" later in this topic.

Use the <u>Show</u> method to make the specified balloon visible. Use the <u>Callback</u> property to run procedures based on selections from modeless balloons (balloons that remain visible while a user works in the application). Use the <u>Close</u> method to close modeless balloons.

The following example creates a balloon that contains tips for saving entered data.

```
With Assistant.NewBalloon
   .BalloonType = msoBalloonTypeBullets
   .Icon = msoIconTip
   .Button = msoButtonSetOk
   .Heading = "Tips for Saving Information."
   .Labels(1).Text = "Save your work often."
   .Labels(2).Text = "Install a surge protector."
   .Labels(3).Text = "Exit your application properly."
   .Show
End With
```

Defining and Reusing Balloons

You can reuse balloon objects you've already created by assigning the object to a variable and displaying the variable when you need it. This example defines balloon1 and balloon2 separately so that they can be reused.

```
Set balloon1 = Assistant.NewBalloon
balloon1.Heading = "First balloon"
Set balloon2 = Assistant.NewBalloon
balloon2.Heading = "Second balloon"
balloon1.Show
balloon2.Show
balloon1.Heading = "First balloon, new heading"
balloon1.Show
```

Alternatively, instead of using separate variables, you can place the balloon object into an array.

BalloonCheckBox Object

<u>Assistant</u> <u>Balloon</u> <u>BalloonCheckBoxes (BalloonCheckBox)</u> <u>BalloonLabels (BalloonLabel)</u>

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

Represents a check box in the Office Assistant balloon. The **BalloonCheckBox** object is a member of the **BalloonCheckBoxes** collection.

Using the BalloonCheckBox Object

Use **CheckBoxes**(*index*), where *index* is a number from 1 through 5, to return a single **BalloonCheckBox** object. There can be up to five check boxes in one balloon; each check box appears when a value is assigned to its **Text** property.

The following example creates a balloon with a heading, text, and three region choices. The user selects one or more check boxes and clicks **OK**. The example calls the specified procedure or procedures.

```
With Assistant.NewBalloon
    .Heading = "Regional Sales Data"
    .Text = "Select your region"
    For i = 1 To 3
        .CheckBoxes(i).Text = "Region " & i
    Next
    .Button = msoButtonSetOkCancel
    .Show
    If .CheckBoxes(1).Checked Then
        runregion1
    End If
    If .CheckBoxes(2).Checked Then
        runregion2
    End If
    If .CheckBoxes(3).Checked Then
        runregion3
    End If
End With
```

Remarks

Balloon check boxes display the user's choices until the user dismisses the balloon. You can use balloon labels to return a number corresponding to the user's choice in the **Select** method as soon as the user clicks the button beside the label. To pass values to the **Select** method based on the user's choice, you must have the balloon type set to **msoBalloonTypeButtons**.

BalloonCheckBoxes Collection Object

<u>Assistant</u> <u>Balloon</u> <u>BalloonCheckBoxes (BalloonCheckBox)</u> <u>BalloonLabels (BalloonLabel)</u>

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

A collection of **BalloonCheckBox** objects that represent all the check boxes in the Office Assistant balloon.

Using the BalloonCheckBoxes Collection

Use the **CheckBoxes** property to return the **BalloonCheckBoxes** collection.

Use **CheckBoxes**(*index*), where *index* is a number from 1 through 5, to return a single **BalloonCheckBox** object. You can specify up to five check boxes (and five labels) per balloon; each check box appears when a value is assigned to its **Text** property. If you specify more than five check boxes, a run-time error occurs.

The following example creates a balloon with a heading, text, and three region choices. When the user selects one or more check boxes and then clicks **OK**, the specified procedure or procedures are called.

```
With Assistant.NewBalloon
    .Heading = "Regional Sales Data"
    .Text = "Select your region"
    For i = 1 To 3
        .CheckBoxes(i).Text = "Region " & i
    Next
    .Button = msoButtonSetOkCancel
    .Show
    If .CheckBoxes(1).Checked Then
        runregion1
    End If
    If .CheckBoxes(2).Checked Then
        runregion2
    End If
    If .CheckBoxes(3).Checked Then
        runregion3
    End If
End With
```

You cannot add check boxes to or remove check boxes from the **BalloonCheckBoxes** collection after the balloon has been displayed.

Remarks

Balloon check boxes display the user's choices until the user dismisses the balloon. You can use balloon labels in conjunction with the **Select** method to return a number corresponding to the user's choice of check boxes as soon as the user clicks the button beside the label. To pass values to the **Select** method based on the user's choice, you must have the balloon type set to **msoBalloonTypeButtons**.

BalloonLabel Object

<u>Assistant</u> <u>Balloon</u> <u>BalloonCheckBoxes (BalloonCheckBox)</u> <u>BalloonLabels (BalloonLabel)</u>

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

Represents a label in the Office Assistant balloon. The **BalloonLabel** object is a member of the **BalloonLabels** collection.

Using the BalloonLabel Object

Use **Labels**(*index*), where *index* is a number from 1 through 5, to return a **BalloonLabel** object. There can be up to five labels on one balloon; each label appears when a value is assisgned to its **Text** property.

The following example creates a balloon that asks the user to click the label corresponding to his or her age.

```
With Assistant.NewBalloon
   .Heading = "Check Your Age Group."
   .Labels(1).Text = "Under 30."
   .Labels(2).Text = "30 to 50."
   .Labels(3).Text = "Over 50."
   .Text = "Which of the following " _
        & .Labels.Count & " choices apply to you?"
   .Show
End With
```

Remarks

Balloon check boxes display the user's choices until he or she dismisses the balloon. You can use balloon labels to return a number corresponding to the user's choice in the **Select** method as soon as the user clicks the button beside the label. To pass values to the **Select** method based on the user's choice, you must have the balloon type be set to **msoBalloonTypeButtons**.

BalloonLabels Collection Object

<u>Assistant</u> <u>Balloon</u> <u>BalloonCheckBoxes (BalloonCheckBox)</u> <u>BalloonLabels (BalloonLabel)</u>

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

A collection of **BalloonLabel** objects that represent all the labels in the Office Assistant balloon.

Using the BalloonLabels Collection

Use the **Labels** property to return the **BalloonLabels** collection.

Use **Labels**(*index*), where *index* is a number from 1 through 5, to return a **BalloonLabel** object. You can specify up to five labels (and five check boxes) per balloon; each label appears when a value is assisgned to its <u>Text</u> property. If you specify more than five labels, a run-time error occurs.

The following example creates a balloon containing three choices. The variable returnValue is set to the return value of the **Show** method, which will be 1, 2, or 3, corresponding to the label the user clicks. The example returns the value of the variable returnValue, which you can either pass to another procedure or use in a **Select Case** statement.

```
Set b = Assistant.NewBalloon
With b
    .Heading = "This is my heading"
    .Text = "Select one of these things:"
    .Labels(1).Text = "Choice One"
    .Labels(2).Text = "Choice Two"
    .Labels(3).Text = "Choice Three"
    returnValue = .Show
End With
```
Remarks

Balloon check boxes display the user's choices until the user dismisses the balloon. You can use balloon labels to return a number corresponding to the user's choice in the **Select** method as soon as the user clicks the button beside the label. To pass values to the **Select** method based on the user's choice, you must have the balloon type set to **msoBalloonTypeButtons**.

COMAddIn Object

COMAddIns (COMAddIn)

Represents a COM add-in in the Microsoft Office host application. The **COMAddIn** object is a member of the **COMAddIns** collection.

Using the COMAddIn Object

Use **COMAddIns.Item**(*index*), where *index* is either an ordinal value that returns the COM add-in at that position in the **COMAddIns** collection, or a **String** value that represents the ProgID of the specified COM add-in. The following example displays a COM add-in's description text in a message box.

MsgBox Application.COMAddIns.Item("msodraa9.ShapeSelect").Descriptic

Use the **ProgID** property of the **COMAddin** object to return the programmatic identifier for a COM add-in, and use the **Guid** property to return the globally unique identifier (GUID) for the add-in. The following example displays the ProgID and GUID for COM add-in one in a message box.

```
MsgBox "My ProgID is " & _
Application.COMAddIns(1).ProgID & _
" and my GUID is " & _
Application.COMAddIns(1).Guid
```

Use the **Connect** property to set or return the state of the connection to a specified COM add-in. The following example displays a message box that indicates whether COM add-in one is registered and currently connected.

```
If Application.COMAddIns(1).Connect Then
    MsgBox "The add-in is connected."
Else
MsgBox "The add-in is not connected."
End If
```

COMAddIns Collection Object

COMAddIns (COMAddIn)

A collection of **<u>COMAddIn</u>** objects that provide information about a COM addin registered in the Windows registry.

Using the COMAddIns Collection

Use the **COMAddIns** property of the **Application** object to return the **COMAddIns** collection for a Microsoft Office host application. This collection contains all of the COM add-ins that are available to a given Office host application, and the <u>Count</u> property of the **COMAddins** collection returns the number of available COM add-ins, as in the following example.

MsgBox Application.COMAddIns.Count

Use the **<u>Update</u>** method of the **COMAddins** collection to refresh the list of COM add-ins from the Windows registry, as in the following example.

Application.COMAddIns.Update

Use **COMAddIns.Item**(*index*), where *index* is either an ordinal value that returns the COM add-in at that position in the **COMAddIns** collection, or a **String** value that represents the ProgID of the specified COM add-in. The following example displays a COM add-in's description text and ProgID ("msodraa9.ShapeSelect") in a message box.

MsgBox Application.COMAddIns.Item("msodraa9.ShapeSelect").Descriptic

CommandBar Object

CommandBars (CommandBar) ^LCommandBarControls (CommandBarControl) <u>CommandBarButton</u> <u>CommandBarComboBox</u> <u>CommandBarPopup</u>

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

Represents a <u>command bar</u> in the container application. The **CommandBar** object is a member of the <u>CommandBars</u> collection.

Using the CommandBar Object

Use **CommandBars**(*index*), where *index* is the name or index number of a command bar, to return a single **CommandBar** object. The following example steps through the collection of command bars to find the command bar named "Forms." If it finds this command bar, the example makes it visible and protects its docking state. In this example, the variable cb represents a **CommandBar** object.

```
foundFlag = False
For Each cb In CommandBars
    If cb.Name = "Forms" Then
        cb.Protection = msoBarNoChangeDock
        cb.Visible = True
        foundFlag = True
    End If
Next cb
If Not foundFlag Then
        MsgBox "The collection does not contain a Forms command bar."
End If
```

You can use a name or index number to specify a menu bar or toolbar in the list of available menu bars and toolbars in the container application. However, you must use a name to specify a menu, shortcut menu, or submenu (all of which are represented by **CommandBar** objects). This example adds a new menu item to

the bottom of the **Tools** menu. When clicked, the new menu item runs the procedure named "qtrReport."

```
Set newItem = CommandBars("Tools").Controls.Add(Type:=msoControlButt
With newItem
    .BeginGroup = True
    .Caption = "Make Report"
    .FaceID = 0
    .OnAction = "qtrReport"
End With
```

If two or more custom menus or submenus have the same name,

CommandBars(*index*) returns the first one. To ensure that you return the correct menu or submenu, locate the <u>pop-up control</u> that displays that menu. Then apply the <u>CommandBar</u> property to the pop-up control to return the command bar that represents that menu.

Assuming that the third control on the toolbar named "Custom Tools" is a popup control, this example adds the **Save** command to the bottom of that menu.

```
Set viewMenu = CommandBars("Custom Tools").Controls(3)
viewMenu.Controls.Add ID:=3 'ID of Save command is 3
```

CommandBarButton Object

<u>CommandBars (CommandBar)</u> <u>CommandBarControls (CommandBarControl)</u>

CommandBarButton

<u>CommandBarPopup</u>

Represents a <u>button control</u> on a <u>command bar</u>.

Using the CommandBarButton Object

Use **Controls**(*index*), where *index* is the index number of the control, to return a **CommandBarButton** object. (The **Type** property of the control must be **msoControlButton**.)

Assuming that the second control on the command bar named "Custom" is a button, the following example changes the style of that button.

```
Set c = CommandBars("Custom").Controls(2)
With c
If .Type = msoControlButton Then
        If .Style = msoButtonIcon Then
        .Style = msoButtonIconAndCaption
        Else
        .Style = msoButtonIcon
        End If
End If
End With
```

You can also use the **FindControl** method to return a **CommandBarButton** object.

CommandBarComboBox Object

<u>CommandBars (CommandBar)</u> <u>CommandBarControls (CommandBarControl)</u>

CommandBarButton

<u>CommandBarComboBox</u>

-<u>CommandBarPopup</u>

Represents a <u>combo box control</u> on a <u>command bar</u>.

Using the CommandBarComboBox Object

Use **Controls**(*index*), where *index* is the index number of the control, to return a **CommandBarComboBox** object. (The **Type** property of the control must be **msoControlEdit**, **msoControlDropdown**, **msoControlComboBox**, **msoControlButtonDropdown**, **msoControlSplitDropdown**, **msoControlOCXDropdown**, **msoControlGraphicCombo**, or **msoControlGraphicDropdown**.)

The following example adds two items to the second control on the command bar named "Custom," and then it adjusts the size of the control.

You can also use the **FindControl** method to return a **CommandBarComboBox** object. The following example searches all command bars for a visible **CommandBarComboBox** object whose tag is "sheet assignments."

```
Set myControl = CommandBars.FindControl _
(Type:=msoControlComboBox, Tag:="sheet assignments", Visible:=True)
```

CommandBarControl Object

CommandBars (CommandBar) ^LCommandBarControls (CommandBarControl) CommandBarButton CommandBarComboBox CommandBarPopup

Represents a <u>command bar control</u>. The **CommandBarControl** object is a member of the <u>CommandBarControls</u> collection. The properties and methods of the **CommandBarControl** object are all shared by the <u>CommandBarButton</u>, <u>CommandBarComboBox</u>, and <u>CommandBarPopup</u> objects.

Note When writing Visual Basic code to work with custom command bar controls, you use the **CommandBarButton**, **CommandBarComboBox**, and **CommandBarPopup** objects. When writing code to work with built-in controls in the container application that cannot be represented by one of those three objects, you use the **CommandBarControl** object.

Using the CommandBarControl Object

Use **Controls**(*index*), where *index* is the index number of a control, to return a **CommandBarControl** object. (The **Type** property of the control must be **msoControlLabel**, **msoControlExpandingGrid**, **msoControlSplitExpandingGrid**, or **msoControlGauge**.)

Note Variables declared as **CommandBarControl** can be assigned **CommandBarButton**, **CommandBarComboBox**, and **CommandBarPopup** values.

You can also use the **FindControl** method to return a **CommandBarControl** object. The following example searches for a control of type **msoControlGauge**; if it finds one, it displays the index number of the control and the name of the command bar that contains it. In this example, the variable <code>lbl</code> represents a **CommandBarControl** object.

```
Set lbl = CommandBars.FindControl(Type:= msoControlGauge)
If lbl Is Nothing Then
    MsgBox "A control of type msoControlGauge was not found."
Else
    MsgBox "Control " & lbl.Index & " on command bar " _
        & lbl.Parent.Name & " is type msoControlGauge"
End If
```

CommandBarControls Collection Object

<u>CommandBars (CommandBar)</u> <u>CommandBarControls (CommandBarControl)</u>

CommandBarButton

<u>CommandBarComboBox</u>

<u>CommandBarPopup</u>

A collection of **<u>CommandBarControl</u>** objects that represent the <u>command bar</u> <u>controls</u> on a command bar.

Using the CommandBarControls Collection

Use the <u>Controls</u> property to return the **CommandBarControls** collection. The following example changes the caption of every control on the toolbar named "Standard" to the current value of the **Id** property for that control.

```
For Each ctl In CommandBars("Standard").Controls
    ctl.Caption = CStr(ctl.Id)
Next ctl
```

Use the <u>Add</u> method to add a new command bar control to the **CommandBarControls** collection. This example adds a new, blank button to the command bar named "Custom."

```
Set myBlankBtn = CommandBars("Custom").Controls.Add
```

Use **Controls**(*index*), where *index* is the caption or index number of a control, to return a **CommandBarControl**, **CommandBarButton**, **CommandBarComboBox**, or **CommandBarPopup** object. The following example copies the first control from the command bar named "Standard" to the command bar named "Custom."

```
Set myCustomBar = CommandBars("Custom")
Set myControl = CommandBars("Standard").Controls(1)
myControl.Copy Bar:=myCustomBar, Before:=1
```

CommandBarPopup Object

<u>CommandBars (CommandBar)</u> CommandBarControls (CommandBarControl)

CommandBarButton

<u>CommandBarComboBox</u>

-<u>CommandBarPopup</u>

Represents a <u>pop-up control</u> on a <u>command bar</u>.

Using the CommandBarPopup Object

Use **Controls**(*index*), where *index* is the number of the control, to return a **CommandBarPopup** object. (The **Type** property of the control must be **msoControlPopup**, **msoControlGraphicPopup**, **msoControlButtonPopup**, **msoControlSplitButtonPopup**.)

You can also use the **FindControl** method to return a **CommandBarPopup** object. The following example searches all command bars for a **CommandBarPopup** object whose tag is "Graphics."

```
Set myControl = Application.CommandBars.FindControl _
(Type:=msoControlPopup, Tag:="Graphics")
```

Remarks

Every pop-up control contains a **CommandBar** object. To return the command bar from a pop-up control, apply the **CommandBar** property to the **CommandBarPopup** object.

CommandBars Collection Object

<u>CommandBars (CommandBar)</u> <u>CommandBarControls (CommandBarControl)</u>

CommandBarButton

CommandBarPopup

A collection of **<u>CommandBar</u>** objects that represent the <u>command bars</u> in the container application.

Using the CommandBars Collection

Use the **CommandBars** property to return the **CommandBars** collection. The following example displays in the **Immediate** window both the name and local name of each menu bar and toolbar, and it displays a value that indicates whether the menu bar or toolbar is visible.

```
For Each cbar in CommandBars
Debug.Print cbar.Name, cbar.NameLocal, cbar.Visible
Next
```

Use the <u>Add</u> method to add a new command bar to the collection. The following example creates a custom toolbar named "Custom1" and displays it as a floating toolbar.

```
Set cbar1 = CommandBars.Add(Name:="Custom1", Position:=msoBarFloatin
cbar1.Visible = True
```

Use **CommandBars**(*index*), where *index* is the name or index number of a command bar, to return a single **CommandBar** object. The following example docks the toolbar named "Custom1" at the bottom of the application window.

```
CommandBars("Custom1").Position = msoBarBottom
```

Note You can use the name or index number to specify a menu bar or toolbar in the list of available menu bars and toolbars in the container application. However, you must use the name to specify a menu, shortcut menu, or submenu (all of which are represented by **CommandBar** objects).

If two or more custom menus or submenus have the same name,

CommandBars(*index*) returns the first one. To ensure that you return the correct menu or submenu, locate the <u>pop-up control</u> that displays that menu. Then apply the <u>CommandBar</u> property to the pop-up control to return the command bar that represents that menu.

DocumentProperties Collection Object

A collection of **DocumentProperty** objects. Each **DocumentProperty** object represents a built-in or custom property of a container document.

Using the DocumentProperties Collection

Use the **Add** method to create a new custom property and add it to the **DocumentProperties** collection. You cannot use the **Add** method to create a built-in document property.

Use **BuiltinDocumentProperties**(*index*), where *index* is the index number of the built-in document property, to return a single **DocumentProperty** object that represents a specific built-in document property. Use

CustomDocumentProperties(*index*), where *index* is the number of the custom document property, to return a **DocumentProperty** object that represents a specific custom document property.

DocumentProperty Object

Represents a custom or built-in document property of a container document. The **DocumentProperty** object is a member of the **DocumentProperties** collection.

Using the DocumentProperty Object

Use **BuiltinDocumentProperties**(*index*), where *index* is the name or index number of the built-in document property, to return a single **DocumentProperty** object that represents a specific built-in document property. Use **CustomDocumentProperties**(*index*), where *index* is the name or index number of the custom document property, to return a **DocumentProperty** object that represents a specific custom document property.

The following list contains the names of all the available built-in document properties:

Title	Number of Words
Subject	Number of Characters
Author	Security
Keywords	Category
Comments	Format
Template	Manager
Last Author	Company
Revision Number	Number of Bytes
Application Name	Number of Lines
Last Print Date	Number of Paragraphs
Creation Date	Number of Slides
Last Save Time	Number of Notes
Total Editing Time	Number of Hidden Slides
	Number of Multimedia

Container applications don't necessarily define a value for every built-in document property. If a given application doesn't define a value for one of the built-in document properties, returning the **Value** property for that document property causes an error.

FileDialog Object

<u>FileDialog</u> ^LMultiple objects

Provides file dialog box functionality similar to the functionality of the standard Open and Save dialog boxes found in Microsoft Office applications. With these dialog boxes, users of your solutions can easily specify the files and folders that your solution should use.

Using the FileDialog object

Use the **FileDialog** property to return a **FileDialog** object. The **FileDialog** property is located in each individual Office application's **Application** object. The property takes a single argument, *DialogType*, that determines the type of **FileDialog** object that the property returns. There are four types of **FileDialog** object:

- Open dialog box lets users select one or more files that you can then open in the host application using the **Execute** method.
- SaveAs dialog box lets users select a single file that you can then save the current file as using the **Execute** method.
- File Picker dialog box lets users select one or more files. The file paths that the user selects are captured in the **FileDialogSelectedItems** collection.
- Folder Picker dialog box lets users select a path. The path that the user selects is captured in the **FileDialogSelectedItems** collection.

Each host application can only instantiate a single instance of the **FileDialog** object. Therefore, many of the properties of the **FileDialog** object persist even when you create multiple **FileDialog** objects. Therefore, make sure that you've set all of the properties appropriately for your purpose before you display the dialog box.

In order to display a file dialog box using the **FileDialog** object, you must use the **Show** method. Once a dialog box is displayed, no code will execute until the user dismisses the dialog box. The following example creates and displays a File Picker dialog box and then displays each selected file in a message box.

```
Sub Main()
    'Declare a variable as a FileDialog object.
    Dim fd As FileDialog
    'Create a FileDialog object as a File Picker dialog box.
    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    'Declare a variable to contain the path
    'of each selected item. Even though the path is a String,
    'the variable must be a Variant because For Each...Next
    'routines only work with Variants and Objects.
```

Dim vrtSelectedItem As Variant 'Use a With...End With block to reference the FileDialog object. With fd 'Use the Show method to display the File Picker dialog box a 'The user pressed the action button. If .Show = -1 Then 'Step through each string in the FileDialogSelectedItems For Each vrtSelectedItem In .SelectedItems 'vrtSelectedItem is a String that contains the path 'You can use any file I/O functions that you want to 'This example simply displays the path in a message MsgBox "The path is: " & vrtSelectedItem Next vrtSelectedItem 'The user pressed Cancel. Else End If End With 'Set the object variable to Nothing. Set fd = Nothing

End Sub
FileDialogFilter Object

<u>FileDialogFilters</u> L<u>FileDialogFilter</u>

Represents a file filter in a file dialog box displayed through the **FileDialog** object. Each file filter determines which files are displayed in the file dialog box.

Using the FileDialogFilter object

Use the **Item** method with the **FileDialogFilters** collection to return a **FileDialogFilter** object. Use the **Add** method to add a **FileDialogFilter** object to the **FileDialogFilters** collection. You can return the extensions that a **FileDialogFilter** object uses to filter files with the **Extensions** property and you can return the description of the filter with the **Description** property; however, both of these properties are read-only. If you want to set the extension or description you must use the **Add** method.

The following example iterates through the default filters of the SaveAs dialog box and displays the description of each filter that includes a Microsoft Excel file.

```
Sub Main()
    'Declare a variable as a FileDialogFilters collection.
    Dim fdfs As FileDialogFilters
    'Declare a variable as a FileDialogFilter object.
    Dim fdf As FileDialogFilter
    'Set the FileDialogFilters collection variable to
    'the FileDialogFilters collection of the SaveAs dialog box.
    Set fdfs = Application.FileDialog(msoFileDialogSaveAs).Filters
    'Iterate through the description and extensions of each
    'default filter in the SaveAs dialog box.
    For Each fdf In fdfs
        'Display the description of filters that include
        'Microsoft Excel files.
        If InStr(1, fdf.Extensions, "xls", vbTextCompare) > 0 Then
            MsgBox "Description of filter: " & fdf.Description
        End If
    Next fdf
End Sub
```

FileDialogFilters Collection

<u>FileDialog</u> <u>FileDialogFilters</u>

A collection of **FileDialogFilter** objects that represent the types of files that can be selected in a file dialog box that is displayed using the **FileDialog** object.

Using the FileDialogFilters collection

Use the **Filters** property of the **FileDialog** object to return a **FileDialogFilters** collection. The following code returns the **FileDialogFilters** collection for the File Open dialog box.

```
Application.FileDialog(msoFileDialogOpen).Filters
```

Use the **Add** method to add **FileDialogFilter** objects to the **FileDialogFilters** collection. The following example uses the **Clear** method to clear the collection and then adds filters to the collection. The **Clear** method completely empties the collection; however, if you don't add any filters to the collection after you clear it, the "All files (*.*)" filter is added automatically.

```
Sub Main()
    'Declare a variable as a FileDialog object.
    Dim fd As FileDialog
    'Create a FileDialog object as a File Picker dialog box.
    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    'Declare a variable to contain the path
    'of each selected item. Even though the path is a String,
    'the variable must be a Variant because For Each...Next
    'routines only work with Variants and Objects.
    Dim vrtSelectedItem As Variant
    'Use a With...End With block to reference the FileDialog object.
    With fd
        'Change the contents of the Files of Type list.
        'Empty the list by clearing the FileDialogFilters collection
        .Filters.Clear
        'Add a filter that includes all files.
        .Filters.Add "All files", "*.*"
        'Add a filter that includes GIF and JPEG images and make it
        .Filters.Add "Images", "*.gif; *.jpg; *.jpeg", 1
        'Use the Show method to display the File Picker dialog box a
        'The user pressed the action button.
        If .Show = -1 Then
```

```
'Step through each String in the FileDialogSelectedItems
For Each vrtSelectedItem In .SelectedItems
    'vrtSelectedItem is a String that contains the path
    'You can use any file I/O functions that you want to
    'This example simply displays the path in a message
    MsgBox "Path name: " & vrtSelectedItem
    Next vrtSelectedItem
    'The user pressed Cancel.
    Else
    End If
End With
  'Set the object variable to Nothing.
    Set fd = Nothing
```

End Sub

When changing the **FileDialogFilters** collection, remember that each application can only instantiate a single **FileDialog** object. This means that the **FileDialogFilters** collection will reset to its default filters whenever you call the **FileDialog** method with a new dialog box type.

The following example iterates through the default filters of the SaveAs dialog box and displays the description of each filter that includes a Microsoft Excel file.

```
Sub Main()
    'Declare a variable as a FileDialogFilters collection.
    Dim fdfs As FileDialogFilters
    'Declare a variable as a FileDialogFilter object.
    Dim fdf As FileDialogFilter
    'Set the FileDialogFilters collection variable to
    'the FileDialogFilters collection of the SaveAs dialog box.
    Set fdfs = Application.FileDialog(msoFileDialogSaveAs).Filters
    'Iterate through the description and extensions of each
    'default filter in the SaveAs dialog box.
    For Each fdf In fdfs
    'Display the description of filters that include
    'Microsoft Excel files
```

End Sub

Note A run-time error will occur if the **Filters** property is used in conjunction with the **Clear**, **Add**, or **Delete** methods when applied to a Save As **FileDiaog** object. For example,

Application.FileDialog(msoFileDialogSaveAs).Filters.Clear will result
in a run-time error.

FileDialogSelectedItems Collection

FileDialog FileDialogSelectedItems

A collection of **String** values that correspond to the paths of the files or folders that a user has selected from a file dialog box displayed through the **FileDialog** object.

Using the FileDialogSelectedItems collection

Use the **SelectedItems** property with the **FileDialog** object to return a **FileDialogSelectedItems** collection. The following example displays a File Picker dialog box and displays each selected file in a message box.

```
Sub Main()
    'Declare a variable as a FileDialog object.
    Dim fd As FileDialog
    'Create a FileDialog object as a File Picker dialog box.
    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    'Declare a variable to contain the path
    'of each selected item. Even though the path is a String,
    'the variable must be a Variant because For Each...Next
    'routines only work with Variants and Objects.
    Dim vrtSelectedItem As Variant
    'Use a With...End With block to reference the FileDialog object.
   With fd
         'Allow the selection of multiple file.
        .AllowMultiSelect = True
        'Use the Show method to display the File Picker dialog box a
        'The user pressed the action button.
        If .Show = -1 Then
            'Step through each string in the FileDialogSelectedItems
            For Each vrtSelectedItem In .SelectedItems
                'vrtSelectedItem is a String that contains the path
                'You can use any file I/O functions that you want to
                'This example simply displays the path in a message
                MsgBox "Selected item's path: " & vrtSelectedItem
            Next vrtSelectedItem
        'The user pressed Cancel.
        Else
        End If
    End With
    'Set the object variable to Nothing.
    Set fd = Nothing
```

End Sub

FileSearch Object

<u>FileSearch</u> Multiple objects

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

Represents the functionality of the **Open** dialog box (**File** menu).

Using the FileSearch Object

Use the **FileSearch** property to return the **FileSearch** object. The following example searches for files and displays the number of files found and the name of each file.

```
With Application.FileSearch
    If .Execute() > 0 Then
        MsgBox "There were " & .FoundFiles.Count & _
            "file(s) found."
        For i = 1 To .FoundFiles.Count
            MsgBox .FoundFiles(i)
            Next i
        Else
            MsgBox "There were no files found."
        End If
End With
```

Use the **NewSearch** method to reset the search criteria to the default settings. All property values are retained after each search is run, and by using the **NewSearch** method you can selectively set properties for the next file search without manually resetting previous property values. The following example resets the search criteria to the default settings before beginning a new search.

```
With Application.FileSearch
    .NewSearch
    .LookIn = "C:\My Documents"
    .SearchSubFolders = True
    .FileName = "Run"
    .MatchTextExactly = True
    .FileType = msoFileTypeAllFiles
End With
```

FileTypes Collection

<u>FileSearch</u> L<u>FileTypes</u>

A collection of values of the type **msoFileType** that determine which types of files are returned by the **Execute** method of the **FileSearch** object.

Using the FileTypes collection

Use the **FileTypes** property with the **FileSearch** object to return a **FileTypes** collection; for example:

Set ft = Application.FileSearch.FileTypes

Note The **FileType** property of the **FileSearch** object clears the **FileTypes** collection and sets the first item in the collection to the file type defined by the **FileType** property.

There is only one **FileTypes** collection for all searches so it's important to clear the **FileTypes** collection before executing a search unless you wish to search for file types from previous searches. The easiest way to clear the collection is to set the **FileType** property to the first file type for which you want to search. You can also remove individual types using the **Remove** method. To determine the file type of each item in the collection, use the **Item** method to return the **msoFileType** value.

The following example searches for all HTML and Microsoft Excel files on the C:\ drive.

```
Sub SearchForFiles()
    'Declare a variable to act as a generic counter
    Dim lngCount As Long
    'Use a With...End With block to reference the
    'FileSearch object
    With Application.FileSearch
        'Clear all the parameters of the previous searches.
        'This method doesn't clear the LookIn property or
        'the SearchFolders collection.
        .NewSearch
        'Setting the FileType property clears the
        'FileTypes collection and sets the first
        'item in the collection to the file type
        'defined by the FileType property.
        .FileType = msoFileTypeWebPages
        'Add a second item to the FileTypes collection
```

```
.FileTypes.Add msoFileTypeExcelWorkbooks
    'Display the number of FileTypes in the collection.
   MsgBox "You are about to search for " & .FileTypes.Count & _
        " file types."
    'Set up the search to look in all subfolders on the C:\ driv
    LookIn = "C: \
    .SearchSubFolders = True
    'Execute the search and test to see if any files
    'were found.
    If .Execute <> 0 Then
        'Display the number of files found.
        MsgBox "Files found: " & .FoundFiles.Count
        'Loop through the list of found files and
        'display the path of each one in a message box.
        For lngCount = 1 To .FoundFiles.Count
            If MsgBox(.FoundFiles.Item(lngCount), vbOKCancel,
                "Found files") = vbCancel Then
                'Break out of the loop
                lngCount = .FoundFiles.Count
            End If
        Next lngCount
    Else
        MsgBox "No files found."
   End If
End With
```

```
End Sub
```

The following example loops through the **FileTypes** collection and removes any file types that aren't Microsoft Word or Microsoft Excel files (in general, it's simpler to clear the **FileTypes** collection and start from scratch).

```
Sub RemoveFileTypeFromCollection()
   'Define an integer to use as a counter
   'when iterating through the FileTypes collection.
   Dim intFileIndex As Integer
   'Use a With...End With block to reference the FileSearch object.
   With Application.FileSearch
    'Loop through all of the items in the FileTypes collection.
```

```
intFileIndex = 1
        Do While intFileIndex <= .FileTypes.Count</pre>
            Select Case .FileTypes.Item(intFileIndex)
                Case msoFileTypeWordDocuments, msoFileTypeExcelWorkb
                Case Else
                    'If the file type isn't a Microsoft Word or
                    'Microsoft Excel file, remove it.
                    .FileTypes.Remove intFileIndex
                    'Decrement the counter so that no file types are
                    intFileIndex = intFileIndex - 1
            End Select
            'Increment the counter to test the next file type.
            intFileIndex = intFileIndex + 1
        Loop
    End With
End Sub
```

FoundFiles Object

FileSearch PropertyTests (PropertyTest)

Represents the list of files returned from a file search.

Using the FoundFiles Object

Use the **FoundFiles** property to return the **FoundFiles** object. This example steps through the list of files that are found and displays the path and file name of each file. Use **FoundFiles**(*index*), where *index* is the index number, to return the path and file name of a specific file found during the search.

```
With Application.FileSearch
    For i = 1 To .FoundFiles.Count
        MsgBox .FoundFiles(i)
        Next I
End With
```

Use the **Execute** method to begin the file search and update the **FoundFiles** object. The following example searches the My Documents folder for all files whose names begin with "Cmd" and displays the name and location of each file that's found. The example also sorts the returned files in ascending alphabetic order by file name.

```
Set fs = Application.FileSearch
With fs
    .LookIn = "C:\My Documents"
    .FileName = "cmd*"
If .Execute(SortBy:=msoSortbyFileName, _
    SortOrder:=msoSortOrderAscending) > 0 Then
    MsgBox "There were " & .FoundFiles.Count & _
        " file(s) found."
    For i = 1 To .FoundFiles.Count
        MsgBox .FoundFiles(i)
        Next i
    Else
        MsgBox "There were no files found."
    End If
End With
```

HTMLProject Object

HTMLProject LHTMLProjectItems (HTMLProjectItem)

Represents a top-level project branch, as in the Project Explorer in the Microsoft Script Editor.

Using the HTMLProject Object

Use the **Open** method of the **HTMLProject** object to open an HTML project in the Microsoft Script Editor. The project is opened in source view or text view for the active Microsoft Word document, Excel workbook, or PowerPoint presentation. The following example opens an HTML project in the active Word document in source view.

ActiveDocument.HTMLProject.Open (msoHTMLProjectOpenSourceView)

Use the **HTMLProjectItems** property to return the collection of **HTMLProjectItem** objects in the HTML project. Use the **RefreshDocument** method to refresh the HTML document in the host application. Use the **RefreshProject** method to refresh the project in the Microsoft Script Editor. Use the **State** method to determine whether the HTML project needs to be refreshed.

HTMLProjectItem Object

HTMLProject LHTMLProjectItems (HTMLProjectItem)

Represents an individual project item that's a project item branch in the Project Explorer in the Microsoft Script Editor. The **HTMLProjectItem** object is a member of the **HTMLProjectItems** collection.

Using the HTMLProjectItem Object

Use **HTMLProjectItems**(*index*), where *index* is the name or index number of a project item, to return a single **HTMLProjectItem** object. Use the **Name** property to return the display name of the project item. The following example returns the name of the first project item in the **HTMLProjectItems** collection for the active document.

```
MsgBox "The first item is " & _
ActiveDocument.HTMLProject.HTMLProjectItems(1).Name
```

Use the **Open** method to open a project item in source view or text view, and use the **IsOpen** property to determine whether the project item is currently open. The following example opens the project item named "ItemOne" (in the active document) in the default view and then displays a message box stating whether the item was opened successfully.

```
ActiveDocument.HTMLProject.HTMLProjectItems("ItemOne").Open
If ActiveDocument.HTMLProject. _
HTMLProjectItems("ItemOne").IsOpen Then
MsgBox "Opened project item " & ActiveDocument.HTMLProject.HTMLProje
End If
```

Use the **SaveCopyAs** method to save the project item using a new file name. The following example saves a copy of ItemOne as "NewItem".

```
ActiveDocument.HTMLProject.HTMLProjectItems("ItemOne") _
.Open (msoHTMLProjectOpenTextView)
ActiveDocument.HTMLProject.HTMLProjectItems("ItemOne") _
.SaveCopyAs("C:\NewItem.txt")
```

Assuming that the text file C:\NewText.txt exists, the following example uses the **LoadFromFile** property to set the text of ItemOne to the text contained in the file. The following example uses the **Text** property to display the new text in a message box.

```
MsgBox ActiveDocument.HTMLProject.HTMLProjectItems _
    ("ItemOne").Text
ActiveDocument.HTMLProject.HTMLProjectItems _
    ("ItemOne").LoadFromFile("C:\NewText.txt")
MsgBox ActiveDocument.HTMLProject.HTMLProjectItems _
    ("ItemOne").Text
```

HTMLProjectItems Collection Object

<u>HTMLProject</u> <u>HTMLProjectItems (HTMLProjectItem)</u>

A collection of **HTMLProjectItem** objects that represent the HTML project items contained in the **HTMLProject** object.

Using the HTMLProjectItems Collection

Use the **HTMLProjectItems** property of the **HTMLProject** object to return the **HTMLProjectItems** collection. Use the **Count** property of the **HTMLProjectItems** collection to return the number of project items in the HTML project for the specified document. Use the **Item** method of the **HTMLProjectItems** collection to return an individual project item. The following example returns the name of the first project item in the **HTMLProjectItems** collection for the active document.

MsgBox "The first item is " & _ ActiveDocument.HTMLProject.HTMLProjectItems(1).Name

LanguageSettings Object

LanguageSettings

Returns information about the language settings in a Microsoft Office application.

Using the LanguageSettings Object

Use **Application.LanguageSettings.LanguageID**(*MsoAppLanguageID*), where *MsoAppLanguageID* is a constant used to return locale identifier (LCID) information to the specified application.

MsoAppLanguageID can be one of these MsoAppLanguageID constants.

```
msoLanguageIDExeMode
msoLanguageIDHelp
msoLanguageIDInstall
msoLanguageIDUI
msoLanguageIDUI
```

The following example returns the install language, user interface language, and Help language LCIDs in a message box.

```
MsgBox "The following locale IDs are registered " & _
  "for this application: Install Language - " & _
  Application.LanguageSettings.LanguageID(msoLanguageIDInstall) &
  " User Interface Language - " & _
  Application.LanguageSettings.LanguageID(msoLanguageIDUI) & _
  " Help Language - " & _
  Application.LanguageSettings.LanguageID(msoLanguageIDHelp)
```

Use Application.LanguageSettings.LanguagePreferredForEditing to determine which LCIDs are registered as preferred editing languages for the application, as in the following example.

```
If Application.LanguageSettings. _
LanguagePreferredForEditing(msoLanguageIDEnglishUS) Then
```

MsgBox "U.S. English is one of the chosen editing languagess." End If

MsoEnvelope Object

MsoEnvelope ^LCommandBars

Provides access to functionality that lets you send documents as emails directly from Microsoft Office applications.

Using the MsoEnvelope object

Use the **MailEnvelope** property of the **Document** object, **Chart** object or **Worksheet** object (depending on the application you are using) to return a **MsoEnvelope** object.

The following example sends the active Microsoft Word document as an e-mail to the e-mail address that you pass to the subroutine.

```
Sub SendMail(ByVal strRecipient As String)
    'Use a With...End With block to reference the MsoEnvelope object
   With Application.ActiveDocument.MailEnvelope
        'Add some introductory text before the body of the e-mail.
        .Introduction = "Please read this and send me your comments.
        'Return a Microsoft Outlook MailItem object that
        'you can use to send the document.
        With .Item
            'All of the mail item settings are saved with the docume
            'When you add a recipient to the Recipients collection
            'or change other properties, these settings will persist
            .Recipients.Add strRecipient
            .Subject = "Here is the document."
            'The body of this message will be
            'the content of the active document.
            .Send
        End With
    End With
End Sub
```

NewFile Object

NewFile

The **NewFile** object represents items listed on the **New** *Item* task pane available in several Microsoft Office applications. The following table shows the property to use to access the **NewFile** object in each of the applications.

Application	Property
Microsoft Access	<u>NewFileTaskPane</u>
Microsoft Excel	<u>NewWorkbook</u>
Microsoft FrontPage	<u>NewPageOrWeb</u>
Microsoft PowerPoint	NewPresentation
Microsoft Word	NewDocument

Note The examples below are for Word, but you can change the **NewDocument** property for any of the properties listed above and use the code in the corresponding application.

Using the NewFile object

Use the **Add** method to add a new item to the **New** *Item* task pane. The following example adds an item to Word's **New Document** task pane.

```
Sub AddNewDocToTaskPane()
    Application.NewDocument.Add FileName:="C:\NewDocument.doc", _
        Section:=msoNew, DisplayName:="New Document"
        CommandBars("Task Pane").Visible = True
End Sub
```

Use the **Remove** method to remove an item from the **New** *Item* task pane. The following example removes the document added in the above example from Word's **New Document** task pane.

```
Sub RemoveDocFromTaskPane()
    Application.NewDocument.Remove FileName:="C:\NewDocument.doc", _
        Section:=msoNew, DisplayName:="New Document"
        CommandBars("Task Pane").Visible = True
End Sub
```

ODSOColumn Object

ODSOColumns ODSOColumn

Represents a field in a data source. The **ODSOColumn** object is a member of the **ODSOColumns** collection. The **ODSOColumns** collection includes all the data fields in a mail merge data source (for example, Name, Address, and City).

Using the ODSOColumn object

Use <u>Columns</u>(*index*), where *index* is the data field name or index number, to return a single **ODSOColumn** object. The index number represents the position of the data field in the mail merge data source. This example retrieves the name and value of the first field of the first record in the data source attached to the active publication.

Remarks

You cannot add fields to the **ODSOColumns** collection. All data fields in a data source are automatically included in the **ODSOColumns** collection.

ODSOColumns Object

OfficeDataSourceObject ODSOColumns

A collection of **ODSOColumn** objects that represent the data fields in a mail merge data source.

Using the ODSOColumns object

Use the **Columns** property to return the **ODSOColumns** collection. The following example displays the field names in the data source attached to the active publication.

```
Sub ShowFieldNames()
   Dim appOffice As OfficeDataSourceObject
   Dim intCount As Integer
   Set appOffice = Application.OfficeDataSourceObject
   appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName
      "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
   With appOffice.Columns
      For intCount = 1 To .Count
        MsgBox "Column Name: " & .Item(intCount).Name
        Next
   End With
End Sub
```

Use **Columns**(*index*), where *index* is the data field name or the index number, to return a single **ODSOColumn** object. The index number represents the position of the data field in the mail merge data source. This example retrieves the name of the first field and value of the first record of the FirstName field in the data source attached to the active publication.

```
Sub GetDataFromSource()
   Dim appOffice As OfficeDataSourceObject
   Set appOffice = Application.OfficeDataSourceObject
   appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName
      "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
   With appOffice.Columns
      MsgBox "Field Name: " & .Columns(1).Name & _
      "Value: " & .Columns("FirstName").Value
   End With
End Sub
```

ODSOFilter Object

ODSOFilters LODSOFilter

Represents a filter to be applied to an attached mail merge data source. The **ODSOFilter** object is a member of the **ODSOFilters** object.
Using the ODSOFilter object

Each filter is a line in a query string. Use the <u>Column</u>, <u>Comparison</u>, <u>CompareTo</u>, and <u>Conjunction</u> properties to return or set the data source query criterion. The following example changes an existing filter to remove from the mail merge all records that do not have a Region field equal to "WA".

```
Sub SetQueryCriterion()
    Dim appOffice As Office.OfficeDataSourceObject
    Dim intItem As Integer
    Set appOffice = Application.OfficeDataSourceObject
    appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName
        "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
    With appOffice.Filters
        For intItem = 1 To .Count
            With .Item(intItem)
                If .Column = "Region" Then
                    .Comparison = msoFilterComparisonNotEqual
                    .CompareTo = "WA"
                    If .Conjunction = "Or" Then .Conjunction = "And"
                End If
            End With
        Next intItem
    End With
End Sub
```

Use the **Add** method of the **ODSOFilters** object to add a new filter criterion to the query. This example adds a new line to the query string and then applies the combined filter to the data source.

```
Sub SetQueryCriterion()
   Dim appOffice As OfficeDataSourceObject
   Set appOffice = Application.OfficeDataSourceObject
   appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName
    "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
   With appOffice.Filters
    .Add Column:="Region", ______
    Comparison:=msoFilterComparisonIsBlank, ______
    Conjunction:=msoFilterConjunctionAnd
    .ApplyFilter
   End With
```

End Sub

ODSOFilters Object

OfficeDataSourceObject CODSOFilters

Represents all the filters to apply to the data source attached to the mail merge publication. The **ODSOFilters** object is comprised of **ODSOFilter** objects.

Using the ODSOFilters object

Use the **Add** method of the **ODSOFilters** object to add a new filter criterion to the query. This example adds a new line to the query string and then applies the combined filter to the data source.

```
Sub SetQueryCriterion()
    Dim appOffice As OfficeDataSourceObject
    Set appOffice = Application.OfficeDataSourceObject
    appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName
        "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
    With appOffice.Filters
        .Add Column:="Region", _
        Comparison:=msoFilterComparisonIsBlank, _
        Conjunction:=msoFilterConjunctionAnd
        .ApplyFilter
    End With
End Sub
```

Use the **Item** method to access an individual filter criterion. This example loops through all the filter criterion and if it finds one with a value of "Region", changes it to remove from the mail merge all records that are not equal to "WA".

```
Sub SetQueryCriterion()
    Dim appOffice As Office.OfficeDataSourceObject
    Dim intItem As Integer
    Set appOffice = Application.OfficeDataSourceObject
    appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName
        "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
    With appOffice.Filters
        For intItem = 1 To .Count
            With .Item(intItem)
                If .Column = "Region" Then
                    .Comparison = msoFilterComparisonNotEqual
                    .CompareTo = "WA"
                    If .Conjunction = "Or" Then .Conjunction = "And"
                End If
            End With
        Next intItem
    End With
End Sub
```

OfficeDataSourceObject Object

OfficeDataSourceObject Multiple objects

Represents the mail merge data source in a mail merge operation.

Using the OfficeDataSourceObject object

To work with the **OfficeDataSourceObject** object, dimension a variable as an **OfficeDataSourceObject** object. You can then work with the different properties and methods associated with the object. Use the **SetSortOrder** method to specify how to sort the records in a data source. The following example sorts the data source first according to ZIP code in descending order, then on last name and first name in ascending order.

```
Sub SetDataSortOrder()
    Dim appOffice As OfficeDataSourceObject
    Set appOffice = Application.OfficeDataSourceObject
    appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName
        "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
    appOffice.SetSortOrder SortField1:="ZipCode", _
        SortAscending1:=False, SortField2:="LastName",
        SortField3:="FirstName"
```

End Sub

Use the **Column**, **Comparison**, **CompareTo**, and **Conjunction** properties to return or set the data source query criterion. The following example changes an existing filter to remove from the mail merge all records that do not have a Region field equal to "WA".

```
Sub SetQueryCriterion()
    Dim appOffice As Office.OfficeDataSourceObject
    Dim intItem As Integer
    appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName
        "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
    With appOffice.Filters
        For intItem = 1 To .Count
            With .Item(intItem)
                If .Column = "Region" Then
                    .Comparison = msoFilterComparisonNotEqual
                    .CompareTo = "WA"
                    If .Conjunction = "Or" Then .Conjunction = "And"
                End If
            End With
        Next intItem
    End With
```

End Sub

PropertyTest Object

<u>FileSearch</u> PropertyTests (PropertyTest) <u>FoundFiles</u>

Represents a single file search criterion. Search criteria are listed in the **Advanced Find** dialog box (**File** menu, **Open** command, **Advanced Find** button). The **PropertyTest** object is a member of the **PropertyTests** collection.

Using the PropertyTest Object

Use **PropertyTests**(*index*), where *index* is the index number, to return a single **PropertyTest** object. The following example displays all the search criteria for the first property test in the **PropertyTests** collection.

```
With Application.FileSearch.PropertyTests(1)
myString = "This is the search criteria: " _
    & " The name is: " & .Name & ". The condition is: " _
    & .Condition
If .Value <> "" Then
    myString = myString & ". The value is: " & .Value
    If .SecondValue <> "" Then
        myString = myString _
            & ". The second value is: " _
            & .SecondValue & ", and the connector is" _
            & .Connector
    End If
End If
End If
MsgBox myString
End With
```

PropertyTests Collection Object

<u>FileSearch</u> PropertyTests (PropertyTest) <u>FoundFiles</u>

A collection of **PropertyTest** objects that represent all the search criteria of a file search. Search criteria are listed in the **Advanced Find** dialog box (**File** menu, **Open** command, **Advanced Find** button).

Using the PropertyTests Collection

Use the **PropertyTests** property to return the **PropertyTests** collection. The following example displays the number of advanced-find search criteria that will be used for one file search.

Application.FileSearch.PropertyTests.Count

Use the **Add** method to add a new **PropertyTest** object to the **PropertyTests** collection. The following example adds two property tests to the search criteria. The first criterion specifies that the files that are found can be of any file type, and the second criterion specifies that these files must have been modified between January 1, 1996, and June 30, 1996. The example displays the number of files found and displays the name of each file in a message box.

```
Set fs = Application.FileSearch
fs.NewSearch
With fs.PropertyTests
    .Add Name:="Files of Type", _
        Condition:=msoConditionFileTypeAllFiles,
        Connector:=msoConnectorOr
    .Add Name:="Last Modified", _
        Condition:=msoConditionAnytimeBetween, __
        Value:="1/1/96", SecondValue:="6/1/96", _
        Connector:=msoConnectorAnd
End With
    If fs.Execute() > 0 Then
        MsgBox "There were " & fs.FoundFiles.Count & _
            " file(s) found."
        For i = 1 To fs.FoundFiles.Count
            MsqBox fs.FoundFiles(i)
        Next i
    Else
        MsgBox "There were no files found."
    End If
```

Use **PropertyTests**(*index*), where *index* is the index number, to return a single **PropertyTest** object. The following example displays all the search criteria for the first property test in the **PropertyTests** collection.

```
With Application.FileSearch.PropertyTests(1)
myString = "This is the search criteria: " _
& " The name is: " & .Name & ". The condition is: " _
```

ScopeFolder Object

Multiple objects ^L<u>ScopeFolder</u> <u>ScopeFolders</u>

Corresponds to a searchable folder. **ScopeFolder** objects are intended for use with the **SearchFolders** collection. The **SearchFolders** collection defines the folders that are searched when using the **FileSearch** object. When you want to search specific folders you can use the methods and properties of the **SearchScope** object and **ScopeFolders** collection to retrieve **ScopeFolder** objects and add them to the **SearchFolders** collection.

Using the ScopeFolder object

Use the **ScopeFolder** property of the **SearchScope** object to return the root **ScopeFolder** object of a search scope; for example:

Set sf = Application.FileSearch.SearchScopes.Item(1).ScopeFolder

Use the **Item** method of the **ScopeFolders** collection to return a subfolder of a root **ScopeFolder** object; for example:

Set sf = Application.FileSearch.SearchScopes.Item(1).ScopeFolder.Sco

Use the **Item** method of the **SearchFolders** collection to return a folder that will be searched the next time the **Execute** method of the **FileSearch** object is called; for example:

```
Set sf = Application.FileSearch.SearchFolders.Item(1)
```

In each **ScopeFolder** object there is a **ScopeFolders** collection that contains the subfolders of the parent **ScopeFolder** object. You can traverse the entire folder structure of a search scope (for example, all local drives) by looping through these **ScopeFolders** collections and returning all of the lower-level **ScopeFolder** objects. A **ScopeFolder** object with no subfolders contains an empty **ScopeFolders** collection.

For an example that demonstrates how to loop through all of the **ScopeFolder** objects in a search scope, see the <u>SearchFolders</u> collection topic.

You can use the **Add** method of the **SearchFolders** collection to add a **ScopeFolder** object to the **SearchFolders** collection, however, it is usually simpler to use the **AddToSearchFolders** method of the **ScopeFolder** that you want to add, as there is only one **SearchFolders** collection for all searches.

For an example that demonstrates how to add a **ScopeFolder** to the **SearchFolders** collection, see the <u>SearchFolders</u> collection topic.

The following example displays the root path of each directory in My Computer. To retrieve this information, the example first gets the **ScopeFolder** object at the root of My Computer. The path of this **ScopeFolder** object will always be "*".

As with all **ScopeFolder** objects, the root object contains a **ScopeFolders** collection. This example loops through this **ScopeFolders** collection and displays the path of each **ScopeFolder** object in it. The paths of these **ScopeFolder** objects will be "A:\", "C:\", etc.

```
Sub DisplayRootScopeFolders()
    'Declare variables that reference a
    'SearchScope and a ScopeFolder object.
    Dim ss As SearchScope
    Dim sf As ScopeFolder
    'Use a With...End With block to reference the
    'FileSearch object.
    With Application.FileSearch
        'Loop through the SearchScopes collection
        'and display all of the root ScopeFolders collections in
        'the My Computer scope.
        For Each ss In .SearchScopes
            Select Case ss.Type
                Case msoSearchInMyComputer
                     'Loop through each ScopeFolder object in
                     'the ScopeFolders collection of the
                     'SearchScope object and display the path.
                    For Each sf In ss.ScopeFolder.ScopeFolders
                        MsgBox "ScopeFolder object's path: " & sf.Pa
                    Next sf
                Case Else
            End Select
        Next
    End With
End Sub
```

ScopeFolders Collection

ScopeFolder ^LScopeFolders

A collection of **ScopeFolder** objects. Only **ScopeFolder** objects contain **ScopeFolders** collections. Each **ScopeFolders** collection contains the **ScopeFolder** objects that correspond to the subfolders of the parent **ScopeFolder** object.

Using the ScopeFolders collection

Use the **ScopeFolders** property of the **ScopeFolder** object to return a **ScopeFolders** collection.

Dim sfs as ScopeFolders
Set sfs = Application.FileSearch.SearchScopes.Item(1).ScopeFolder.Sc

You can't add or remove **ScopeFolder** objects from a **ScopeFolders** collection.

Script Object

Scripts (Script)

Represents a block of HTML script in a Microsoft Word document, on a Microsoft Excel spreadsheet, or on a Microsoft PowerPoint slide. The **Script** object is a member of the **Scripts** collection.

Using the Script Object

Use **Scripts.Item**(*index*), where *index* is the name, ID, or index number of a script, to return a single **Script** object. Each **Script** object is identified by the **Id** property, which provides a convenient name you can use to access the script. The following example adds a single script to the **Scripts** collection for the active document and displays the ID of the script at index value 1.

```
myScript = ActiveDocument.Scripts.Add( _
   , msoScriptLocationInBody, _
    msoScriptLanguageVisualBasic, _
    "ScriptOne", , _
    "MsgBox (""This is ScriptOne. "")")
MsgBox (ActiveDocument.Scripts(1).Id)
```

You can specify the scripting language used in the script by changing the **Language** property. The following example changes the scripting language of script one to Active Server Pages (ASP).

```
ActiveDocument.Scripts.Item("ScriptOne") _
.Language = msoScriptLanguageASP
```

You can check the location of the script anchor shape within an HTML document by using the <u>Location</u> property. The following example checks to determine whether ScriptOne is in the body of the active HTML document.

```
If ActiveDocument.Scripts("ScriptOne").Location = _
    msoScriptLocationInBody Then
    MsgBox ("Script is in the HTML document body.")
Else
    MsgBox ("Script is located in the header. ")
```

End If

You can check or set attributes added to the <SCRIPT> tag (with the exception of the LANGUAGE and ID attributes) by using the **Extended** property. The following example checks for additional attributes in script one in the active document.

```
If ActiveDocument.Scripts(1).Extended = "" Then
    MsgBox ("No additional attributes are present " & _
    "in Script " &
    ActiveDocument.Scripts(1).Id)
```

You can check or set the script text associated with a given script by using the **<u>ScriptText</u>** property. The following example displays a message box containing the script text associated with script one in the active document.

MsgBox (ActiveDocument.Scripts("ScriptOne").ScriptText)

Scripts Collection Object

Scripts (Script)

A collection of <u>Script</u> objects that represent the collection of HTML scripts in the specified document.

Using the Scripts Collection

The **Scripts** collection contains all of the **Script** objects in a given document, in source order (the order in which **Script** objects were added to the source file). Source order isn't affected by the location (header or body text) of the script in the document. You can use **Script** objects to access a script or to add a script to a Microsoft Word document, a Microsoft Excel worksheet, or a Microsoft PowerPoint slide. You can also use the **Scripts** collection to access any HTML page or script that's opened in a Microsoft Office application.

Note Microsoft Access doesn't use this shared Office component.

Adding a Script

When you add a **Script** object to the **Scripts** collection, a **Shape** object of type **msoScriptAnchor** is automatically added to the document. On an Excel worksheet or a PowerPoint slide, the shape is added to the **Shapes** collection; in a Word Document, the shape is added to the **InlineShapes** collection. You add a **Script** to a document by using the **Add** method. The following example adds a simple script to the active Word document.

```
myScript = ActiveDocument.Scripts.Add( _
   , msoScriptLocationInBody, _
   msoScriptLanguageVisualBasic, _
   "ScriptOne", , _
   "MsgBox ""This is ScriptOne.""")
```

To access a particular item in the **Scripts** collection, use the **Item** method, and supply either the ID attribute of the <SCRIPT> tag or the index number that indicates the position of the script in the collection. The ID must be unique within the document. In the case of duplicate ID attributes, the first script found that has that ID is returned. The following example displays a message box indicating the language of the first script found that uses the ID "ScriptOne".

```
MsgBox (ActiveDocument.Scripts.Item("ScriptOne").Language)
```

Use the **<u>Count</u>** property to determine the number of **Script** objects in the specified document. The following example displays the number of scripts in the active document.

```
If ActiveDocument.Scripts.Count = 0 Then
    MsgBox ("There are no " & _
    "scripts in this document. ")
End If
If ActiveDocument.Scripts.Count = 1 Then
    MsgBox ("There is " & _
    ActiveDocument.Scripts.Count & _
    " script in this document. ")
End If
If ActiveDocument.Scripts.Count > 1 Then
    MsgBox ("There are " & _
    ActiveDocument.Scripts.Count & _
    " scripts in this document. ")
End If
If ActiveDocument.Scripts.Count = 1 Then
    MsgBox ("There are " & _
    ActiveDocument.Scripts.Count > 1 Then
    MsgBox ("There are " & _
    ActiveDocument.Scripts.Count & _
    " scripts in this document. ")
End If
```

Use the **Delete** method to remove a script from the **Scripts** collection, as in the following example.

ActiveDocument.Scripts("ScriptOne").Delete

SearchFolders Collection

<u>FileSearch</u> <u>SearchFolders</u>

A collection of **ScopeFolder** objects that determines which folders are searched when the **Execute** method of the **FileSearch** object is called.

Using the SearchFolders collection

Use the **SearchFolders** property with the **FileSearch** object to return the **SearchFolders** collection; for example:

Set sfs = Application.FileSearch.SearchFolders

For each application there is only a single **SearchFolders** collection. The contents of the collection remains after the code that calls it has finished executing. Consequently, it is important to clear the collection unless you want to include folders from previous searches in your search.

You can use the **Add** method of the **SearchFolders** collection to add a **ScopeFolder** object to the **SearchFolders** collection, however, it is usually simpler to use the **AddToSearchFolders** method of the **ScopeFolder** that you want to add, as there is only one **SearchFolders** collection for all searches.

The **SearchFolders** collection can be seen as a compliment to the **LookIn** property of the **FileSearch** object. Both specify the folders to search and both are used when the search is executed. However, if you only want to use the **LookIn** property, you should make sure that the **SearchFolders** collection is empty. Conversely, if you only want to use the **SearchFolders** collection, set the **LookIn** property to the path of the first member of the **SearchFolders** collection before you call the **Execute** method.

The following example searches every folder named "1033" on the local machine for all HTML and Microsoft Excel files. The example makes use of the **SearchFolders** collection, **SearchScopes** collection, and **ScopeFolders** collection.

This example consists of two routines. The SearchEveryFolder routine is the routine that you should run. The OutputPaths routine is separate from the main routine because it calls itself recursively in order to traverse the entire directory structure of the local machine.

```
Sub SearchEveryFolder()
    'Declare variables that reference a
    'SearchScope and a ScopeFolder object.
```

```
Dim ss As SearchScope
Dim sf As ScopeFolder
'Declare a variable to act as a generic counter.
Dim lngCount As Long
'Use a With...End With block to reference the
'FileSearch object.
With Application.FileSearch
    'Clear all the parameters of the previous searches.
    'This method doesn't clear the LookIn property or
    'the SearchFolders collection.
    .NewSearch
    'Specify the type of file for which to search.
    'Use the FileType property to specify the first type
    'and then add additional types to the FileTypes collection.
    .FileType = msoFileTypeWebPages
    .FileTypes.Add msoFileTypeExcelWorkbooks
    'Clear the SearchFolder collection by
    'looping through each ScopeFolder object
    'and removing it.
    For lngCount = 1 To .SearchFolders.Count
        .SearchFolders.Remove lngCount
    Next lngCount
    'Loop through the SearchScopes collection to find
    'the scope in which you want to search. In this
    'case the scope is the local machine.
    For Each ss In .SearchScopes
        Select Case ss.Type
            Case msoSearchInMyComputer
                'Loop through each ScopeFolder in
                'the ScopeFolders collection of the
                'SearchScope object.
                For Each sf In ss.ScopeFolder.ScopeFolders
                    'Call a function that loops through all
                    'of the subfolders of the root ScopeFolder.
                    'This function adds any folders named "1033"
                    'SearchFolders collection.
                    Call OutputPaths(sf.ScopeFolders, "1033")
                Next sf
            Case Else
        End Select
```

```
Next ss
```

```
'Test to see if any ScopeFolders collections were added to
        'the SearchFolders collection.
        If .SearchFolders.Count > 0 Then
            'Set the LookIn property to the path of
            'the first ScopeFolder object in the SearchFolders
            'collection. This is here so that any previous
            'setting of the LookIn property doesn't affect
            'the search.
            .LookIn = .SearchFolders.Item(1).Path
            'Execute the search and test to see if any files
            'were found.
            If .Execute <> 0 Then
                'Display the number of files found.
                MsgBox "Files found: " & .FoundFiles.Count
                'Loop through the list of found files and
                'display the path of each one in a message box.
                For lngCount = 1 To .FoundFiles.Count
                    If MsqBox(.FoundFiles.Item(lngCount), vbOKCancel
                        "Found files") = vbCancel Then
                        'Break out of the loop
                        lngCount = .FoundFiles.Count
                    End If
                Next lngCount
            End If
        End If
    End With
End Sub
'This subroutine loops through all of the ScopeFolders collections
'in a given ScopeFolders collection. It adds any folder
'that has the same name as the value of strFolder
'to the SearchFolders collection.
Sub OutputPaths(ByVal sfs As ScopeFolders,
    ByRef strFolder As String)
    'Declare a variable as a ScopeFolder object
    Dim sf As ScopeFolder
    'Loop through each ScopeFolder object in the
    'ScopeFolders collection.
    For Each sf In sfs
```

'Test to see if the folder name of the ScopeFolder 'matches the value of strFolder. Use LCase to ensure 'that case does not affect the match. If LCase(sf.Name) = LCase(strFolder) Then

'Add the ScopeFolder to the SearchFolders collection. sf.AddToSearchFolders

End If

'Include a DoEvents call because there is the potential for 'loop to last a long time. The DoEvents call allows this pro 'continue handling events. DoEvents

'Test to see if the ScopeFolders collection in the 'current ScopeFolder is empty. If it isn't empty, then 'that means that the current ScopeFolder object contains sub If sf.ScopeFolders.Count > 0 Then

'This subroutine recursively calls itself so that 'it can add the subfolders of the current ScopeFolder ob 'to the SearchFolders collection. Call OutputPaths(sf.ScopeFolders, strFolder)

End If Next sf End Sub

SearchScope Object

SearchScopes LScopeFolder

Corresponds to a type of folder tree that can be searched by using the **FileSearch** object. For example, the local drives on this computer represent a single search scope. Network folders and Microsoft Outlook folders are also separate search scopes that may be available. Each **SearchScope** object contains a single **ScopeFolder** object that corresponds to the root folder of the search scope.

Using the SearchScope object

Use the **Item** method of the **SearchScopes** collection to return a **SearchScope** object; for example:

```
Dim ss As SearchScope
Set ss = Application.FileSearch.SearchScopes.Item(1)
```

Ultimately, the **SearchScope** object is intended to provide access to **ScopeFolder** objects that can be added to the **SearchFolders** collection. For an example that demonstrates how this is accomplished, see the **SearchFolders** collection topic.

See the **<u>ScopeFolder</u>** object topic to see a simple example of how to return a **ScopeFolder** object from a **SearchScope** object.

The following example displays all of the currently available **SearchScope** objects.

```
Sub DisplayAvailableScopes()
    'Declare a variable that references a
    'SearchScope object.
    Dim ss As SearchScope
    'Use a With...End With block to reference the
    'FileSearch object.
    With Application.FileSearch
        'Loop through the SearchScopes collection.
        For Each ss In .SearchScopes
            Select Case ss.Type
                Case msoSearchInMyComputer
                    MsgBox "My Computer is an available search scope
                Case msoSearchInMyNetworkPlaces
                    MsgBox "My Network Places is an available search
                Case msoSearchInOutlook
                    MsgBox "Outlook is an available search scope."
                Case msoSearchInCustom
                    MsqBox "A custom search scope is available."
                Case Else
                    MsgBox "Can't determine search scope."
            End Select
        Next ss
```

End With End Sub

SearchScopes Collection

<u>FileSearch</u> <u>SearchScopes</u>

A collection of **SearchScope** objects.

Using the SearchScopes collection

Use the **SearchScopes** property of the **FileSearch** object to return a **SearchScopes** collection; for example:

Dim sss As SearchScopes
Set sss = Application.FileSearch.SearchScopes

You can't add or remove **SearchScope** objects from the **SearchScopes** collection.

Signature Object

<u>SignatureSet</u> <u>Signature</u>

Corresponds to a digital signature that is attached to a document. **Signature** objects are contained in the **SignatureSet** collection of the **Document** object.

Using the Signature object

You can add a **Signature** object to a **SignatureSet** collection using the **Add** method and you can return an existing member using the **Item** method. To remove a **Signature** from a **SignatureSet** collection, use the **Delete** method of the **Signature** object.

The following example prompts the user to select a digital signature with which to sign the active document in Microsoft Word. To use this example, open a document in Word and pass this function the name of a certificate issuer and the name of a certificate signer that match the **Issued By** and **Issued To** fields of a digital certificate in the **Digital Certificates** dialog box. This example will test to make sure that the digital signature that the user selects meets certain criteria, such as not having expired, before the new signature is committed to the disk.

```
Function AddSignature(ByVal strIssuer As String, _
    strSigner As String) As Boolean
    On Error GoTo Error_Handler
    Dim sig As Signature
    'Display the dialog box that lets the
    'user select a digital signature.
    'If the user selects a signature, then
    'it is added to the Signatures
    'collection. If the user doesn't, then
    'an error is returned.
    Set sig = ActiveDocument.Signatures.Add
    'Test several properties before commiting the Signature object t
    If sig.Issuer = strIssuer And
        sig.Signer = strSigner And _
        sig.IsCertificateExpired = False And _
        sig.IsCertificateRevoked = False And _
        sig.IsValid = True Then
        MsgBox "Signed"
        AddSignature = True
    'Otherwise, remove the Signature object from the SignatureSet co
    Else
        siq.Delete
        MsgBox "Not signed"
        AddSignature = False
```
End If

'Commit all signatures in the SignatureSet collection to the dis ActiveDocument.Signatures.Commit

```
Exit Function
Error_Handler:
AddSignature = False
MsgBox "Action cancelled."
End Function
```

SignatureSet Collection

<u>SignatureSet</u> <u>Signature</u>

A collection of **Signature** objects that correspond to the digital signatures attached to a document.

Using the SignatureSet collection

Use the **Signatures** property of the **Document** object to return a **SignatureSet** collection; for example:

Set sigs = ActiveDocument.Signatures

Note Changes that you make to the **SignatureSet** collection of a document will not persist unless you call the **Commit** method.

You can add a **Signature** object to a **SignatureSet** collection using the **Add** method and you can return an existing member using the **Item** method. To remove a **Signature** from a **SignatureSet** collection, use the **Delete** method of the **Signature** object.

The following example prompts the user to select a digital signature with which to sign the active document in Microsoft Word. To use this example, open a document in Word and pass this function the name of a certificate issuer and the name of a certificate signer that match the **Issued By** and **Issued To** fields of a digital certificate in the **Digital Certificates** dialog box. This example will test to make sure that the digital signature that the user selects meets certain criteria, such as not having expired, before the new signature is committed to the disk.

```
sig.IsCertificateRevoked = False And _
        sig.IsValid = True Then
       MsgBox "Signed"
        AddSignature = True
    'Otherwise, remove the Signature object from the SignatureSet co
    Else
        sig.Delete
       MsgBox "Not signed"
        AddSignature = False
    End If
    'Commit all signatures in the SignatureSet collection to the dis
   ActiveDocument.Signatures.Commit
    Exit Function
Error_Handler:
   AddSignature = False
    MsgBox "Action cancelled."
End Function
```

WebPageFont Object

WebPageFonts (WebPageFont)

Represents the default font used when documents are saved as Web pages for a particular character set.

Using the WebPageFont Object

Use the **WebPageFont** object to describe the proportional font, proportional font size, fixed-width font, and fixed-width font size for any available <u>character set</u>.

The following character sets are supported.

```
msoCharacterSetArabic
msoCharacterSetCyrillic
msoCharacterSetEnglishWesternEuropeanOtherLatinScript
msoCharacterSetGreek
msoCharacterSetHebrew
msoCharacterSetJapanese
msoCharacterSetJapanese
msoCharacterSetKorean
msoCharacterSetMultilingualUnicode
msoCharacterSetSimplifiedChinese
msoCharacterSetThai
msoCharacterSetThai
msoCharacterSetTraditionalChinese
msoCharacterSetVietnamese
```

The following example sets the proportional font and proportional font size for the **WebPageFont** object myFont.

```
With myFont
ProportionalFont = Verdana
ProportionalFontSize = 14
```

WebPageFonts Collection Object

WebPageFonts (WebPageFont)

A collection of <u>WebPageFont</u> objects that describe the proportional font, proportional font size, fixed-width font, and fixed-width font size used when documents are saved as Web pages. You can specify a different set of Web page font properties for each available character set.

Using the WebPageFonts Collection

The **WebPageFonts** collection contains one **WebPageFont** object for each <u>character set</u>.

The following character sets are supported. msoCharacterSetArabic msoCharacterSetCyrillic msoCharacterSetEnglishWesternEuropeanOtherLatinScript msoCharacterSetGreek msoCharacterSetHebrew msoCharacterSetJapanese msoCharacterSetJapanese msoCharacterSetKorean msoCharacterSetKorean msoCharacterSetMultilingualUnicode msoCharacterSetSimplifiedChinese msoCharacterSetThai msoCharacterSetTraditionalChinese msoCharacterSetVietnamese

The following example uses the **Item** property to set myFont to the **WebPageFont** object for the English/Western European/Other Latin Script character set in the current application.

```
Dim myFont As WebPageFont
Set myFont = _
Application.DefaultWebOptions.Fonts.Item_
(msoCharacterSetEnglishWesternEuropeanOtherLatinScript)
```

ActivateWizard Method

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

Resumes or suspends Office Assistant Help during a custom wizard.

Note You should use this method only with the **<u>StartWizard</u>** method.

expression.ActivateWizard(WizardID, Act, Animation)

expression Required. An expression that returns an **Assistant** object.

WizardID Required Long. The number returned by the **StartWizard** method.

Act Required <u>MsoWizardActType</u>. Specifies the change to the Office Assistant Help session.

MsoWizardActType can be one of these MsoWizardActType constants. msoWizardActActive msoWizardActInactive msoWizardActResume msoWizardActSuspend

Animation Optional **Variant**. The animation the Office Assistant performs when it is suspended or resumed.

Example

This example suspends the wizard session that was started with the **StartWizard** method. The variable lHelpForWiz was set to the return value of the **StartWizard** method.

Assistant.**ActivateWizard** WizardID:=lHelpForWiz, _ Act:=msoWizardActSuspend, Animation:=msoAnimationGoodbye - <u>Show All</u>

Add Method

• Add method as it applies to the **NewFile** object.

Adds a new item to the **New** *Item* task pane. Returns a **Boolean**. **True** if the item was successfully added.

expression.Add(FileName, Section, DisplayName, Action)

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

FileName Required **String**. The name of the file to add to the list of files on the task pane.

Section Optional **Variant**. The section to which to add the file. Can be any **msoFileNewSection** constant.

DisplayName Optional Variant. The text to display in the task pane.

Action Optional **Variant**. The action to take when a user clicks on the item. Can be any **msoFileNewAction** constant.

• Add method as it applies to the **CommandBars** object.

Creates a new <u>command bar</u> and adds it to the collection of command bars. Returns a **CommandBar** object.

expression.Add(Name, Position, MenuBar, Temporary)

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

Name Optional **Variant**. The name of the new command bar. If this argument is omitted, a default name is assigned to the command bar (such as Custom 1).

Position Optional Variant. The position or type of the new command bar. Can be one of the **MsoBarPosition** constants listed in the following table.

Constant	Description
msoBarLeft, msoBarTop, msoBarRight, msoBarBottom	Indicates the left, top, right, and bottom coordinates of the new command bar
msoBarFloating	Indicates that the new command bar won't be docked
msoBarPopup	Indicates that the new command bar will be a shortcut menu
msoBarMenuBar	Macintosh only

MenuBar Optional Variant. True to replace the active menu bar with the new command bar. The default value is **False**.

Temporary Optional Variant. True to make the new command bar temporary. Temporary command bars are deleted when the container application is closed. The default value is **False**.

Add method as it applies to the CommandBarControls object.

Creates a new **CommandBarControl** object and adds it to the collection of controls on the specified <u>command bar</u>.

expression.Add(Type, Id, Parameter, Before, Temporary)

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

Type Optional Variant. The type of control to be added to the specified command bar. Can be one of the following **MsoControlType** constants: **msoControlButton**, **msoControlEdit**, **msoControlDropdown**, **msoControlComboBox**, or **msoControlPopup**.

Id Optional **Variant**. An integer that specifies a built-in control. If the value of this argument is 1, or if this argument is omitted, a blank custom control of the specified type will be added to the command bar.

Parameter Optional **Variant**. For built-in controls, this argument is used by the container application to run the command. For custom controls, you can use this argument to send information to Visual Basic procedures, or you can use it to store information about the control (similar to a second **Tag** property value).

Before Optional **Variant**. A number that indicates the position of the new control on the command bar. The new control will be inserted before the control at this position. If this argument is omitted, the control is added at the end of the specified command bar.

Temporary Optional Variant. True to make the new control temporary. Temporary controls are automatically deleted when the container application is closed. The default value is **False**.

• Add method as it applies to the **DocumentProperties** object.

Creates a new custom document property. You can only add a new document property to the custom **DocumentProperties** collection.

expression.Add(Name, LinkToContent, Type, Value, LinkSource)

expression Required. The custom **DocumentProperties** object.

Name Required **String**. The name of the property.

LinkToContent Required **Boolean**. Specifies whether the property is linked to the contents of the container document. If this argument is **True**, the *LinkSource* argument is required; if it's **False**, the value argument is required.

Type Optional Variant. The data type of the property. Can be one of the following MsoDocProperties constants: msoPropertyTypeBoolean, msoPropertyTypeDate, msoPropertyTypeFloat, msoPropertyTypeNumber, or msoPropertyTypeString.

Value Optional Variant. The value of the property, if it's not linked to the contents of the container document. The value is converted to match the data type specified by the type argument, if it can't be converted, an error occurs. If *LinkToContent* is **True**, the *Value* argument is ignored and the new document property is assigned a default value until the linked property values are updated by the container application (usually when the document is saved).

LinkSource Optional Variant. Ignored if *LinkToContent* is False. The source of the linked property. The container application determines what types of source linking you can use.

Remarks

If you add a custom document property to the **DocumentProperties** collection that's linked to a given value in a Microsoft Office document, you must save the document to see the change to the **DocumentProperty** object.

• Add method as it applies to the **FileDialogFilters** object.

Adds a new file filter to the list of filters in the **Files of type** drop down list box in the File dialog box. Returns a **FileDialogFilter** object that represents the newly added file filter.

expression.Add(Description, Extensions, Position)

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

Description Required **String**. The text representing the description of the file extension you want to add to the list of filters.

Extensions Required String. The text representing the file extension you want to add to the list of filters. More than one extension may be specified and each must be separated by a semi-colon (;). For example, the Extensions argument can be assigned to the string: "*.txt; *.htm". Note Parentheses do not need to be added around the extensions. Office will automatically add parentheses around the extensions string when the description and extensions strings are concatenated into one file filter item.

Position Optional **Variant**. A number that indicates the position of the new control in the filter list. The new filter will be inserted before the filter at this position. If this argument is omitted, the filter is added at the end of the list.

Remarks

Each filter in a list is made up of two parts: the file extension (e.g. .txt) and the text description of the file extension (e.g. Text Files). Together, the file filter would appear in the **Files of type** drop down list box as: Text Files (*.txt).

Note that when a filter is added to the list, the default filters are not removed.

Filters are only displayed when the **Windows** option is checked.

If *Position* is invalid, an out of range runtime error is displayed. If the **Description and Extensions** value are invalid, a runtime error (parse) is displayed.

Folder picker dialogs do not have filters, therefore, filter methods do not apply to the folder picker.

• Add method as it applies to the **Scripts** object.

Adds a **Script** object to the **Scripts** collection of one of the following objects: a **Document** or **Range** object in Microsoft Word; a **Worksheet** or **Chart** object in Microsoft Excel; or a **Slide**, **SlideRange**, slide **Master**, or title **Master** object in Microsoft PowerPoint. Returns a **Script** object.

expression.Add(Anchor, Location, Language, Id, Extended, ScriptText)

expression Required. The **Scripts** collection.

Anchor Optional **Range** (Microsoft Excel only). The *Anchor* argument accepts an Excel **Range** object, which specifies the placement of the script anchor on an Excel **Worksheet**. You cannot insert script anchors into Excel charts.

Location Optional <u>MsoScriptLocation</u>. Specifies the location of the script anchor in a document. If you've specified the *Anchor* argument, the *Location* argument isn't used; the location of the *Anchor* argument determines the location of the script anchor.

MsoScriptLocation can be one of these MsoScriptLocation constants.

msoScriptLocationInBody default msoScriptLocationInHead

Language Optional <u>MsoScriptLanguage</u>. Specifies the script language.

MsoScriptLanguage can be one of these MsoScriptLanguage constants. msoScriptLanguageASP msoScriptLanguageJava msoScriptLanguageOther

msoScriptLanguageVisualBasic default

Id Optional **String**. The ID of the <SCRIPT> tag in HTML. The *Id* argument specifies an SGML identifier used for naming elements. Valid identifiers include any string that begins with a letter and is composed of alphanumeric characters; the string can also include the underscore character (_). The ID must be unique within the HTML document. This parameter is exported as the ID attribute in the <SCRIPT> tag.

Extended Optional String. Specifies attributes that are to be added to the <SCRIPT> tag (LANGUAGE and ID attributes are exported through the *Language* and *Id* parameters and should not be exported through the *Extended* parameter). The default is the empty string. Attributes are separated by spaces, the same as in HTML. The Microsoft Office host application doesn't provide any means of checking the syntax of passed attributes.

ScriptText Optional *String*. Specifies the text contained in a block of script. The default is the empty string. The Microsoft Office host application doesn't check the syntax of the script.

Remarks

A shape associated with a script block isn't exported or printed as a shape in HTML; only the script block gets exported.

You cannot use the **Add** method to add a script anchor to a PowerPoint slide range that contains more than one slide.

• Add method as it applies to the **SignatureSet** object.

Returns a **Signature** object that represents a new e-mail signature.

expression.Add

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

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

Adds a new filter to the **ODSOFilters** collection.

expression.Add(Column, Comparison, Conjunction, bstrCompareTo, DeferUpdate)

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

Column Required **String**. The name of the table in the data source.

Comparison Required <u>MsoFilterComparison</u>. How the data in the table is filtered.

MsoFilterComparison can be one of these MsoFilterComparison constants. msoFilterComparisonContains msoFilterComparisonEqual msoFilterComparisonGreaterThan msoFilterComparisonGreaterThanEqual msoFilterComparisonIsBlank msoFilterComparisonIsBlank msoFilterComparisonLessThan msoFilterComparisonLessThanEqual msoFilterComparisonNotContains msoFilterComparisonNotEqual

Conjunction Required <u>MsoFilterConjunction</u>. Determines how this filter relates to other filters in the **ODSOFilters** object.

MsoFilterConjunction can be one of these MsoFilterConjunction constants. msoFilterConjunctionAnd msoFilterConjunctionOr

bstrCompareTo Optional String. If the *Comparison* argument is something other than **msoFilterComparisonIsBlank** or **msoFilterComparisonIsNotBlank**, a string to which the data in the table is compared.

DeferUpdate Optional **Boolean**. Default is **False**.

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

Creates a new reference (a **String** value) to an Answer Wizard file and adds it to the **AnswerWizardFiles** collection.

expression.Add(FileName)

expression Required. An expression that returns an **AnswerWizardFiles** collection.

FileName Required **String**. The fully qualified path to the specified Answer Wizard file.

• Add method as it applies to the **FileTypes** object.

Adds a new file type to a file search.

```
expression.Add(FileType)
```

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

FileType Required <u>MsoFileType</u>. Specifies the type of file for which to search.

MsoFileType can be one of these MsoFileType constants.

msoFileTypeAllFiles msoFileTypeBinders msoFileTypeCalendarItem msoFileTypeContactItem msoFileTypeCustom **msoFileTypeDatabases msoFileTypeDataConnectionFiles msoFileTypeDesignerFiles** msoFileTypeDocumentImagingFiles msoFileTypeExcelWorkbooks **msoFileTypeJournalItem msoFileTypeMailItem msoFileTypeNoteItem msoFileTypeOfficeFiles msoFileTypeOutlookItems msoFileTypePhotoDrawFiles** msoFileTypePowerPointPresentations **msoFileTypeProjectFiles msoFileTypePublisherFiles msoFileTypeTaskItem msoFileTypeTemplates** msoFileTypeVisioFiles **msoFileTypeWebPages msoFileTypeWordDocuments**

• Add method as it applies to the **PropertyTests** object.

Adds a **<u>PropertyTest</u>** object to the **PropertyTests** collection.

expression.Add(Name, Condition, Value, SecondValue, Connector)

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

Name Required **String**. The name of the property criterion. The name corresponds to a value in the **Property** box in the **Find** dialog box, which you open from the **Tools** menu in the application's **Open** dialog box (**File** menu).

Condition Required <u>MsoCondition</u>. The condition of the search criteria.

MsoCondition can be one of these MsoCondition constants. msoConditionAnyNumberBetween **msoConditionAnytime** msoConditionAnytimeBetween msoConditionAtLeast msoConditionAtMost **msoConditionBeginsWith** msoConditionDoesNotEqual **msoConditionEndsWith msoConditionEquals** msoConditionEqualsCompleted msoConditionEqualsDeferred msoConditionEqualsHigh msoConditionEqualsInProgress **msoConditionEqualsLow** msoConditionEqualsNormal msoConditionEqualsNotStarted msoConditionEqualsWaitingForSomeoneElse msoConditionFileTypeAllFiles msoConditionFileTypeBinders msoConditionFileTypeCalendarItem msoConditionFileTypeContactItem msoConditionFileTypeDatabases msoConditionFileTypeDataConnectionFiles msoConditionFileTypeDesignerFiles msoConditionFileTypeDocumentImagingFiles msoConditionFileTypeExcelWorkbooks msoConditionFileTypeJournalItem

msoConditionFileTypeMailItem msoConditionFileTypeNoteItem msoConditionFileTypeOfficeFiles msoConditionFileTypeOutlookItems msoConditionFileTypePhotoDrawFiles msoConditionFileTypePowerPointPresentations msoConditionFileTypeProjectFiles msoConditionFileTypePublisherFiles msoConditionFileTypeTaskItem msoConditionFileTypeTemplates msoConditionFileTypeVisioFiles msoConditionFileTypeWebPages msoConditionFileTypeWordDocuments **msoConditionFreeText** msoConditionIncludes msoConditionIncludesFormsOf msoConditionIncludesNearEachOther msoConditionIncludesPhrase msoConditionInTheLast msoConditionInTheNext **msoConditionIsExactly** msoConditionIsNo msoConditionIsNot msoConditionIsYes msoConditionLastMonth **msoConditionLastWeek** msoConditionLessThan **msoConditionMoreThan** msoConditionNextMonth msoConditionNextWeek msoConditionNotEqualToCompleted msoConditionNotEqualToDeferred msoConditionNotEqualToHigh

msoConditionNotEqualToInProgress msoConditionNotEqualToLow msoConditionNotEqualToNormal msoConditionNotEqualToNotStarted msoConditionNotEqualToWaitingForSomeoneElse msoConditionOn msoConditionOnOrAfter msoConditionOnOrBefore msoConditionThisMonth msoConditionThisWeek msoConditionToday msoConditionToday msoConditionTomorrow msoConditionYesterday

Value Optional *Variant*. The value of the search criterion.

SecondValue Optional Variant. An upper value for the search range. You can use this argument only if *Condition* is **msoConditionAnyTimeBetween** or **msoConditionAnyNumberBetween**.

Connector Optional <u>MsoConnector</u>. Specifies the way two search criteria are combined.

MsoConnector can be one of these MsoConnector constants. **msoConnectorAnd** *default* **msoConnectorOr**

• Add method as it applies to the **SearchFolders** object.

Adds a search folder to a file search.

expression.Add(ScopeFolder)

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

ScopeFolder Required *ScopeFolder* object. The folder to add to the search.

Example

```
As it applies to the AnswerWizardFiles object.
```

This example prepares the Answer Wizard to accept a custom file list and adds two custom Answer Wizard files. First, the example clears the file list, and then it adds two custom Answer Wizard files and checks the file count and the file names to ensure that the files were added correctly.

```
Dim customAnswerWizard As AnswerWizard
Set customAnswerWizard = Application.AnswerWizard
customAnswerWizard.ClearFileList
customAnswerWizard.Files.Add ("c:\awfiles\custom_1.aw")
customAnswerWizard.Files.Add ("c:\awfiles\custom_2.aw")
If customAnswerWizard.Files.Count = 2 Then
    MsgBox "Files " & customAnswerWizard.Files.Item(1) & _
    " and " & customAnswerWizard.Files(2) & _
    " were added sucessfully."
End If
```

```
• As it applies to the CommandBarControls object.
```

This example creates a custom editing toolbar that contains buttons (controls) for cutting, copying, and pasting.

```
Dim customBar As CommandBar
Dim newButton As CommandBarButton
Set customBar = CommandBars.Add("Custom")
Set newButton = customBar.Controls _
.Add(msoControlButton, CommandBars("Edit") _
.Controls("Cut").Id)
Set newButton = customBar.Controls _
.Add(msoControlButton, CommandBars("Edit") _
.Controls("Copy").Id)
Set newButton = customBar.Controls _
.Add(msoControlButton, CommandBars("Edit") _
.Controls("Paste").Id)
customBar.Visible = True
```

• As it applies to the **DocumentProperties** object.

This example, which is designed to run in Microsoft Word, adds three custom document properties to the **DocumentProperties** collection.

```
With ActiveDocument.CustomDocumentProperties
.Add Name:="CustomNumber", _
LinkToContent:=False, _
Type:=msoPropertyTypeNumber, _
Value:=1000
.Add Name:="CustomString", _
LinkToContent:=False, _
Type:=msoPropertyTypeString, _
Value:="This is a custom property."
.Add Name:="CustomDate", _
LinkToContent:=False, _
Type:=msoPropertyTypeDate, _
Value:=Date
End With
```

▶ <u>As it applies to the **PropertyTests** object.</u>

This example adds two property tests to the search criteria. The first test is that the files must be Microsoft Word documents, and the second test is that they must have been modified between January 1, 1996, and June 30, 1996. The example also displays a message box that shows the total number of files found, if any, and the name of each file found.

```
Set fs = Application.FileSearch
fs.NewSearch
With fs.PropertyTests
    .Add Name:="Files of Type", _
        Condition:=msoConditionFileTypeWordDocuments,
        Connector:=msoConnectorOr
    .Add Name:="Last Modified", _
        Condition:=msoConditionAnytimeBetween,
        Value:="1/1/98", SecondValue:="6/30/98", _
        Connector:=msoConnectorAnd
End With
If fs.Execute() > 0 Then
        For i = 1 To fs.FoundFiles.Count
            strFound = strFound & fs.FoundFiles(i) & vbCrLf
        Next i
        MsgBox "Search found the following " _
            & fs.FoundFiles.Count &
            " file(s):" & vbCrLf & strFound
Else
    MsgBox "There were no files found."
End If
```

• As it applies to the **Scripts** object.

This example adds a new **Script** to the specified range on worksheet one in the active workbook.

```
Dim rngScriptAnchorRange As Range
Dim objNewScript As Script
Set rngScriptAnchorRange = ActiveWorkbook. _
Worksheets(1).Range("B5")
Set objNewScript = ActiveWorkbook. _
Worksheets(1).Scripts.Add(rngScriptAnchorRange, _
msoScriptLocationInBody, _
msoScriptLocationInBody, _
msoScriptLanguageVisualBasic, _
"MyNewScript", , _
"MsgBox (""Added Script object MyNewScript"")")
```

- <u>Show All</u>

AddItem Method

-

Adds a list item to the specified command bar <u>combo box control</u>. The combo box control must be a custom control and must be a drop-down list box or a combo box.

Note This method will fail if it's applied to an edit box or a built-in combo box control.

expression.AddItem(Text, Index)

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

Text Required **String**. The text added to the control.

Index Optional **Variant**. The position of the item in the list. If this argument is omitted, the item is added to the end of the list.

Example

This example adds a combo box control to a command bar. Two items are added to the control, and the number of line items and the width of the combo box are set.

```
Set myBar = CommandBars("Custom")
Set myControl = myBar.Controls.Add(Type:=msoControlComboBox, Id:=1)
With myControl
    .AddItem "First Item", 1
    .AddItem "Second Item", 2
    .DropDownLines = 3
    .DropDownWidth = 75
    .ListHeaderCount = 0
End With
```

AddToSearchFolders Method

Adds a **<u>ScopeFolder</u>** object the <u>SearchFolders</u> collection.

expression.AddToSearchFolders

-

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

Remarks

Although you can use the **SearchFolders** collection's **Add** method to add a **ScopeFolder** object to the **SearchFolders** collection, it is usually simpler to use the **AddToSearchFolders** method of the **ScopeFolder** object that you want to add, because there is only one **SearchFolders** collection for all searches.

Example

The following example adds the root **ScopeFolder** object to the **SearchFolders** collection. For a longer example that uses the **AddToSearchFolders** method, see the **SearchFolders** collection topic.

Application.FileSearch.SearchScopes(1).ScopeFolder.AddToSearchFolder

ApplyFilter Method

Applies a filter to a mail merge data source to filter specified records meeting specified criteria.

expression.ApplyFilter

-

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

Example

This example adds a new filter that removes all records with a blank Region field and then applies the filter to the active publication.

```
Sub OfficeFilters()
Dim appOffice As OfficeDataSourceObject
Dim appFilters As ODSOFilters
Set appOffice = Application.OfficeDataSourceObject
appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName
    "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
Set appFilters = appOffice.Filters
MsgBox appOffice.RowCount
appFilters.Add Column:="Region", Comparison:=msoFilterComparison
    Conjunction:=msoFilterConjunctionAnd, bstrCompareTo:="WA"
    appOffice.RowCount
MsgBox appOffice.RowCount
```

End Sub

- <u>Show All</u>

Clear Method

Removes all list items from a command bar <u>combo box control</u> (drop-down list box or combo box) and clears the text box (edit box or combo box).

Note This method will fail if it's applied to a built-in command bar control.

expression.Clear

-

expression Required. An expression that returns a **CommandBarComboBox** object.
This example checks the number of items in the combo box control on the command bar named "Custom." If there are fewer than three items in the list in the combo box, the example clears the list, adds a new first item to the list, and then displays this new item as the default for the combo box control.

```
Set myBar = CommandBars("Custom Bar")
Set myControl = myBar.Controls _
   Type:=msoControlComboBox)
With myControl
   .AddItem "First Item", 1
   .AddItem "Second Item", 2
End With
If myControl.ListCount < 3 Then
   myControl.Clear
   myControl.AddItem Text:="New Item", Index:=1
End If</pre>
```

ClearFileList Method

Clears the list of files for the current AnswerWizard, including the default list of files for the Microsoft Office host application.

expression.ClearFileList

-

expression An expression that returns an **AnswerWizard** object.

Remarks

Use this method to remove all entries from the current file list for the specified application. You can also use it to ensure that none of the default AnswerWizard files for the host application are available to users. You can then build a custom list of files by using the <u>Add</u> method of the <u>AnswerWizardFiles</u> collection.

To restore the default AnswerWizard file set for the host application, use the **<u>ResetFileList</u>** method.

This example prepares the AnswerWizard to accept a custom file list and then adds two custom AnswerWizard files. First, the example clears the file list, and then it adds two custom AnswerWizard files and checks the file count and the file names to ensure that the files were added correctly.

```
Dim customAnswerWizard As AnswerWizard
Set customAnswerWizard = Application.AnswerWizard
customAnswerWizard.ClearFileList
customAnswerWizard.Files.Add ("c:\awfiles\custom_1.aw")
customAnswerWizard.Files.Add ("c:\awfiles\custom_2.aw")
If customAnswerWizard.Files.Count = 2 Then
    MsgBox "Files " & customAnswerWizard.Files.Item(1) & _
    " and " & customAnswerWizard.Files(2) & _
    " were added sucessfully."
End If
```

Close Method

Closes the active modeless balloon. You should use this method only in callback procedures.

expression.Close

-

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

This example displays a balloon that contains a button for each of three printers. Whenever the user clicks one of these buttons, the ProcessPrinter callback procedure is run and the balloon is closed.

```
Sub selectPrinter()
Set bln = Assistant.NewBalloon
With bln
    .Heading = "Select a Printer."
    .Labels(1).Text = "Network Printer"
    .Labels(2).Text = "Local Printer"
    .Labels(3).Text = "Local Color Printer"
    .BalloonType = msoBalloonTypeButtons
    .Mode = msoModeModeless
    .Callback = "ProcessPrinter"
    .Show
End With
End Sub
Sub ProcessPrinter(bln As Balloon, lbtn As Long, _
 lPriv As Long)
    Assistant.Animation = msoAnimationPrinting
    Select Case 1btn
    Case -1
        ' Insert network printer-specific code.
    Case -2
        ' Insert local printer-specific code.
    Case -3
        ' Insert color printer-specific code.
    End Select
    bln.Close
End Sub
```

Commit Method

Commits all changes of the specified **SignatureSet** collection to disk. Until the **Commit** method is executed, none of the changes to the **SignatureSet** collection are saved.

expression.Commit

-

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

The following example prompts the user to select a digital signature with which to sign the active document in Microsoft Word. To use this example, open a document in Word and call this function. The function will test to make sure that the digital signature that the user selects will not expire in less than 12 months. If it will expire, the certificate isn't attached.

```
Function AddSignature() As Boolean
    On Error GoTo Error Handler
    Dim sig As Signature
    'Display the dialog box that lets the
    'user select a digital signature.
    'If the user selects a signature, then
    'it is added to the Signatures
    'collection. If the user doesn't, then
    'an error is returned.
    Set sig = ActiveDocument.Signatures.Add
    'Test to make sure that the new Signature object
    'doesn't expire too soon. This expression calculates
    'the number of months until the Signature object expires.
    If DateDiff("m", sig.SignDate, sig.ExpireDate) < 12 Then</pre>
        MsgBox "This certificate will expire in less than 1 year." &
        "Please use a newer certificate."
        AddSignature = False
        sig.Delete
    Else
        AddSignature = True
    End If
    'Commit all signatures in the SignatureSet collection to the dis
    ActiveDocument.Signatures.Commit
    Exit Function
Error Handler:
    AddSignature = False
    MsgBox "Action cancelled."
End Function
```

- <u>Show All</u>

Copy Method

Copies a <u>command bar control</u> to an existing <u>command bar</u>.

expression.Copy(Bar, Before)

expression Required. An expression that returns a **CommandBarControl**, **CommandBarButton**, **CommandBarPopup**, or **CommandBarComboBox** object.

Bar Optional Variant. A CommandBar object that represents the destination command bar. If this argument is omitted, the control is copied to the command bar where the control already exists.

Before Optional **Variant**. A number that indicates the position for the new control on the command bar. The new control will be inserted before the control at this position. If this argument is omitted, the control is copied to the end of the command bar.

This example copies the first control from the command bar named "Standard" to the first control on the command bar named "Custom".

```
Set myCustomBar = CommandBars("Custom")
Set myControl = CommandBars("Standard").Controls(1)
With myControl
    .Copy Bar:=myCustomBar, Before:=1
    .SetFocus
End With
```

- <u>Show All</u>

CopyFace Method

Copies the face of a command bar <u>button control</u> to the Clipboard.

expression.CopyFace

-

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

Remarks

Use the **PasteFace** method to paste the contents of the Clipboard onto a button face.

This example finds the built-in **Open** button, copies the button face to the Clipboard, and then pastes the face onto the **Spelling and Grammar** button.

Set myControl = CommandBars.FindControl(Type:=msoControlButton, Id:=
myControl.CopyFace
Set myControl = CommandBars.FindControl(Type:=msoControlButton, ID:=
myControl.PasteFace

- <u>Show All</u>

Delete Method

Delete method as it applies to the **DocumentProperty** object.

Removes a custom document property.

expression. Delete

÷.,

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

Remarks

You cannot delete a built-in document property.

▶ <u>Delete method as it applies to the **AnswerWizardFiles** object.</u>

Deletes the specified object from its collection.

expression.Delete(FileName)

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

FileName Required **String**. The name of the file to be deleted, including the fully-qualified path, file name, and extension.

• Delete method as it applies to the **FileDialogFilters** object.

Removes a file dialog filter.

expression.**Delete**(*filter*)

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

filter Optional **Variant**. The filter to be removed.

• Delete method as it applies to the **ODSOFilters** object.

Deletes a filter object from the **ODSOFilters** collection.

expression.Delete(Index, DeferUpdate)

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

Index Required **Long**. The number of the filter to delete.

DeferUpdate Optional Boolean.

Delete method as it applies to the CommandBar, Script, Scripts, and Signature objects. Deletes the specified object from the collection.

expression.**Delete**

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

Remarks

For the **Scripts** collection, using the **Delete** method removes all scripts from the specified Word document, Excel worksheet, or PowerPoint slide. A script anchor is represented by a shape in the host application. Therefore, the **Shape** object associated with each script anchor of type **msoScriptAnchor** is deleted from the **Shapes** collection in Excel and PowerPoint and from the **InlineShapes** and **Shapes** collections in Word.

Delete method as it applies to the CommandBarButton, CommandBarComboBox, CommandBarControl, and CommandBarPopup objects.

Deletes the specified object from its collection.

expression.Delete(Temporary)

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

Temporary Optional **Variant**. **True** to delete the control for the current session. The application will display the control again in the next session.

```
• As it applies to the CommandBar object.
```

This example deletes all custom command bars that aren't visible.

```
foundFlag = False
delBars = 0
For Each bar In CommandBars
    If (bar.BuiltIn = False) And _
      (bar.Visible = False) Then
      bar.Delete
      foundFlag = True
      delBars = delBars + 1
    End If
Next bar
If Not foundFlag Then
      MsgBox "No command bars have been deleted."
Else
      MsgBox delBars & " custom bar(s) deleted."
End If
```

• As it applies to the **DocumentProperty** object.

This example deletes a custom document property. For this example to run properly, you must have a custom **DocumentProperty** object named "CustomNumber".

ActiveDocument.CustomDocumentProperties("CustomNumber").Delete

- <u>Show All</u>

DoAlert Method

Displays an alert and returns a **Long** that indicates which button the user pressed. You can choose to display this alert either through the Microsoft Office Assistant or as a normal message box.

expression.DoAlert(bstrAlertTitle, bstrAlertText, alb, alc, ald, alq, varfSysAlert)

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

bstrAlertTitle Required **String**. Sets the title of the alert.

bstrAlertText Required **String**. Sets the text of the alert.

alb Required <u>MsoAlertButtonType</u>. Determines which buttons are displayed on the alert.

MsoAlertButtonType can be one of these MsoAlertButtonType constants. msoAlertButtonAbortRetryIgnore msoAlertButtonOKCancel msoAlertButtonRetryCancel msoAlertButtonYesAllNoCancel Only use this when the varfSysAlert argument is set to False. msoAlertButtonYesNo msoAlertButtonYesNoCancel

alc Required <u>MsoAlertIconType</u>. Determines the icon that is displayed on the alert.

MsoAlertIconType can be one of these MsoAlertIconType constants. **msoAlertIconCritical**

msoAlertIconInfo msoAlertIconNoIcon msoAlertIconQuery msoAlertIconWarning

ald Required <u>MsoAlertDefaultType</u>. Determines which button is set as the default button of the alert. If this argument is set to a value greater than the number of buttons, an error is returned.

MsoAlertDefaultType can be one of these MsoAlertDefaultType constants. msoAlertDefaultFifth msoAlertDefaultFirst msoAlertDefaultFourth msoAlertDefaultSecond msoAlertDefaultThird

alq Required <u>MsoAlertCancelType</u>. Always set this to **msoAlertCancelDefault**. Any other setting may return an error.

MsoAlertCancelType can be one of these MsoAlertCancelType constants. msoAlertCancelDefault msoAlertCancelFifth msoAlertCancelFirst msoAlertCancelFourth msoAlertCancelSecond msoAlertCancelThird

varfSysAlert Required **Boolean**. **True** if the alert is displayed in a message box or **False** if the alert is displayed through the Office Assistant.

Remarks

The return values of the **DoAlert** method corresponds to the values of the **vbMsgBoxResult** enumerated type (for example, **vbYes**, **vbNo**, or **vbCancel**). In addition to these values, the following values may also be returned:

- "Yes to all" = 8
- "Try again" = 10
- "Continue" = 11

The following example displays an alert through the Office Assistant and displays a message box indicating which button the user pressed. If the assistant is disabled, the alert is displayed in a normal message box.

```
Sub AssistantAlert()
   With Application.Assistant
        Select Case _
            .DoAlert(
            "Test", _
            "Click a button.", _
            msoAlertButtonYesAllNoCancel, _
            msoAlertIconCritical, _
            msoAlertDefaultSecond, _
            msoAlertCancelFirst, _
            False)
            Case vbYes: MsgBox "The user clicked Yes."
            Case vbNo: MsgBox "The user clicked No."
            Case vbCancel: MsgBox "The user clicked Cancel."
            Case 8: MsgBox "The user clicked Yes To All" 'This is th
            Case Else
        End Select
    End With
End Sub
```

EndWizard Method

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

Releases the variable returned by the **StartWizard** method.

Note You should use this method only with the **<u>StartWizard</u>** method.

expression.EndWizard(WizardID, varfSuccess, Animation)

expression Required. An expression that returns an **Assistant** object.

WizardID Required Long. The number returned by the **StartWizard** method.

varfSuccess Required **Boolean**. **True** to indicate that the user completed the wizard successfully.

Animation Optional **Variant**. The animation the Office Assistant performs if *varfSuccess* is set to **True**. The default value is **msoAnimationCharacterSuccessMajor**.

This example closes the Office Assistant for a wizard session that was completed successfully by the user. The variable lHelpForWiz was assigned the return value of the **StartWizard** method.

```
Assistant.EndWizard WizardId:=lHelpForWiz, _ varfSuccess:=True, Animation:=msoAnimationGoodbye
```

- <u>Show All</u>

Execute Method

• Execute method as it applies to the FileSearch object.

Begins the search for the specified file(s). Returns a **Long**; zero (0) if no files are found, or a positive number if one or more files are found.

expression.Execute(SortBy, SortOrder, AlwaysAccurate)

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

SortBy Optional <u>MsoSortBy</u>. The method used to sort the returned file(s).

MsoSortBy can be one of these MsoSortBy constants. msoSortByFileName default msoSortByFileType msoSortByLastModified msoSortByNone msoSortBySize

SortOrder Optional <u>MsoSortOrder</u>. The order in which the returned file(s) are sorted.

MsoSortOrder can be one of these MsoSortOrder constants. msoSortOrderAscending *default* msoSortOrderDescending

AlwaysAccurate Optional **Boolean**. **True** to have the file search include files that have been added, modified, or deleted since the file index was last updated. The default value is **True**.

Execute method as it applies to the CommandBarButton, CommandBarComboBox, CommandBarControl, CommandBarPopup, and FileDialog objects. For the command bar objects, runs the procedure or built-in command assigned to the specified <u>command bar control</u>. For custom controls, use the <u>OnAction</u> property to specify the procedure to be run.

For **FileDialog** objects of type **msoFileDialogOpen** or **msoFileDialogSaveAs**, carries out a user's action right after the **Show** method is invoked.

expression. Execute

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

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

This example searches for all files in the My Documents folder that end with the file name extension ".doc" and then displays the location and name of each file found. The example also sorts the list of returned file names in ascending alphabetic order.

```
Set fs = Application.FileSearch
With fs
    .LookIn = "C:\My Documents"
    .FileName = "*.doc"
If .Execute(SortBy:=msoSortbyFileName, _
        SortOrder:=msoSortOrderAscending) > 0 Then
        MsgBox "There were " & .FoundFiles.Count & _
        " file(s) found."
        For i = 1 To .FoundFiles.Count
            MsgBox .FoundFiles(i)
            Next i
        Else
            MsgBox "There were no files found."
        End If
End With
```

```
▶ As it applies to the CommandBarButton, CommandBarComboBox, CommandBarControl, and CommandBarPopup objects.
```

This Microsoft Excel example creates a command bar and then adds a built-in command bar button control to it. The button executes the Excel **AutoSum** function. This example uses the **Execute** method to total the selected range of cells when the command bar appears.

```
Dim cbrCustBar As CommandBar
Dim ctlAutoSum As CommandBarButton
Set cbrCustBar = CommandBars.Add("Custom")
Set ctlAutoSum = cbrCustBar.Controls _
.Add(msoControlButton, CommandBars("Standard") _
.Controls("AutoSum").Id)
cbrCustBar.Visible = True
ctlAutoSum.Execute
```

FindControl Method

Returns a **CommandBarControl** object that fits a specified criteria.

expression.FindControl(Type, Id, Tag, Visible, Recursive)

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

Type Optional <u>MsoControlType</u>. The type of control.

MsoControlType type can be one of these MsoControlType constants. **msoControlActiveX msoControlCustom msoControlButton msoControlEdit msoControlDropdown** msoControlComboBox **msoControlButtonDropdown msoControlSplitDropdown** msoControlGenericDropdown msoControlGraphicCombo msoControlSplitButtonMRUPopup msoControlSplitExpandingGrid msoControlGraphicDropdown **msoControlPopup** msoControlGraphicPopup **msoControlButtonPopup msoControlGauge** msoControlLabel **msoControlExpandingGrid** msoControlGrid

msoControlOCXDropDown msoControlSplitButtonPopup msoControlPane

Id Optional **Variant**. The identifier of the control.

Tag Optional **Variant**. The tag value of the control.

Visible Optional Variant. True to include only visible command bar controls in the search. The default value is **False**. Visible command bars include all visible toolbars and any menus that are open at the time the **FindControl** method is executed.

Recursive Optional **Boolean**. **True** to include the command bar and all of its pop-up subtoolbars in the search. This argument only applies to the **CommandBar** object. The default value is **False**.
If the **CommandBars** collection contains two or more controls that fit the search criteria, **FindControl** returns the first control that's found. If no control that fits the criteria is found, **FindControl** returns **Nothing**.

This example finds the first control on the command bar named "Custom". If the control is a button, the example uses the **FindControl** method to find the **Copy** button (on the **Standard** toolbar) and then copies the face from the **Copy** button and pastes it onto the control.

```
Set oldCtrl = CommandBars("Custom").Controls(1)
If oldCtrl.Type = 1 Then
   Set newCtrl = CommandBars.FindControl(Type:= _
        MsoControlButton, ID:= _
        CommandBars("Standard").Controls("Copy").ID)
   NewCtrl.CopyFace
   OldCtrl.PasteFace
End If
```

FindControls Method

Returns the **CommandBarControls** collection that fits the specified criteria.

expression.FindControls(Type, Id, Tag, Visible)

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

Type Optional <u>MsoControlType</u>. The type of control.

MsoControlType type can be one of these MsoControlType constants. **msoControlActiveX msoControlCustom msoControlButton msoControlEdit msoControlDropdown** msoControlComboBox **msoControlButtonDropdown msoControlSplitDropdown** msoControlGenericDropdown **msoControlGraphicCombo** msoControlSplitButtonMRUPopup msoControlSplitExpandingGrid msoControlGraphicDropdown **msoControlPopup** msoControlGraphicPopup **msoControlButtonPopup** msoControlGauge **msoControlLabel** msoControlExpandingGrid

msoControlGrid msoControlOCXDropDown msoControlSplitButtonPopup msoControlPane Id Optional Variant. The control's identifier.

Tag Optional **Variant**. The control's tag value.

Visible Optional Variant. True to include only visible command bar controls in the search. The default value is **False**.

If no controls that fits the criteria are found, the **FindControls** method returns **Nothing**.

This example uses the **FindControls** method to return all members of the **CommandBars** collection that have an ID of 18 and displays (in a message box) the number of controls that meet the search criteria.

```
Dim myControls As CommandBarControls
Set myControls = CommandBars.FindControls(Type:=msoControlButton, ID
MsgBox "There are " & myControls.Count & _
        " controls that meet the search criteria."
```

Help Method

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

Displays the Office Assistant and the built-in "What would you like to do?" Assistant balloon for standard Office online Help.

expression.Help

-

expression Required. An expression that returns an **Assistant** object.

This example displays the built-in "What would you like to do?" Assistant balloon when the user checks the "I need more information" check box.

```
Set b = Assistant.NewBalloon
With b
    .Heading = "User Information"
    .Text = "Select your skill level"
    .CheckBoxes(1).Text = "Beginner."
    .CheckBoxes(2).Text = "Advanced."
    .CheckBoxes(3).Text = "I need more information."
    .Show
End With
If b.CheckBoxes(3).Checked = True Then
    Assistant.Help
End If
```

- <u>Show All</u>

Item Method

Item method as it applies to the COMAddIns object.

Returns a member of the specified **COMAddIns** collection.

expression.Item(Index)

expression Required. The specified COMAddIns collection.

Index Required **Variant**. Either an ordinal value that returns the COM add-in at that position in the **COMAddIns** collection, or a **String** value that represents the ProgID of the specified COM add-in.

▶ Item method as it applies to the **FileDialogFilters** object.

Returns a **FileDialogFilter** object that is a member of the specified **FileDialogFilters** collection.

expression.Item(Index)

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

Index Required **Long**. The index number of the **FileDialogFilter** object to be returned.

Item method as it applies to the HTMLProjectItems object.

Returns the **HTMLProjectItem** object that represents a particular project in the Microsoft Script Editor.

expression.**Item**(*Index*)

expression Required. An HTMLProjectItems collection.

Index Required **Variant**. The name or index number of the HTML project item

to be returned.

▶ <u>Item method as it applies to the **Scripts** object.</u>

Returns a member of the **Scripts** collection. Accepts the index number or ID of the script you want the Microsoft Office application to return. The **Item** method accepts a **Variant** value that can accept either an ordinal number (index value), which returns the script stored at that position in the **Scripts** collection, or a **String** value that represents the name or ID of the script.

expression.Item(Index)

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

Index Required **Variant**. The ID or index number of the script to be returned.

The **Scripts** collection contains all of the scripts in a given document, in source order (the order in which **Script** objects appear in the source file). Scripts are maintained in source order regardless of their location in the document—that is, whether they're in the header or the body text.

You can use the **Item** method to access a script in the **Scripts** collection by using the ID of the <SCRIPT> tag. The ID attribute of the <SCRIPT> tag is identical to the **Id** property of the **Script** object. If there are duplicate or multiple IDs in the document and you use the **Id** property of a **Script** object to access a script by using the **Item** method, Office returns the first script that matches the ID; additional scripts with the same ID are ignored.

New script anchors added to the collection are appended to the end of the **Scripts** collection in the order in which they were added to the document. The script anchors remain in this order until the document is saved as HTML, closed, and then opened again in the host application. Following these steps causes the **Scripts** collection to be indexed in the order in which the script anchors appear in the document, which may be different than the order in which they were added to it. Therefore, you're advised to use the **Id** property of the **Script** object, rather than the script's position in the collection, to ensure positive identification of the script.

Item method as it applies to the FileDialogSelectedItems object.

Returns a **String** that corresponds to the path of one of the files that the user selected from a file dialog box that was displayed using the **Show** method of the **FileDialog** object. The **FileDialogSelectedItems** collection is a collection of strings.

expression.Item(Index)

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

Index Required **Long**. The index number of the string to be returned.

LoadFromFile Method

Updates the text in the Microsoft Script Editor with text from the specified file (on disk).

expression.LoadFromFile(Filename)

-

expression An HTMLProjectItem object.

Filename Required **String**. The fully qualified path of the text file that contains the text to be loaded.

This example determines whether the specified HTML project item is open; if the item is open, the example then loads script from the specified file.

```
If ActiveWorkbook.HTMLProject.HTMLProjectItems _
    .Item(1).IsOpen Then
    ActiveWorkbook.HTMLProject.HTMLProjectItems _
    .Item(1).LoadFromFile ("C:\MyScript.txt")
Else
    MsgBox "The HTMLProjectItem is not open."
End If
```

- <u>Show All</u>

Move Method

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

Move method as it applies to the CommandBarButton, CommandBarComboBox, CommandBarControl, and CommandBarPopup objects.

Moves the specified <u>command bar control</u> to an existing <u>command bar</u>.

expression.Move(Bar, Before)

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

Bar Optional Variant. A CommandBar object that represents the destination command bar for the control. If this argument is omitted, the control is moved to the end of the command bar where the control currently resides.

Before Optional **Variant**. A number that indicates the position for the control. The control is inserted before the control currently occupying this position. If this argument is omitted, the control is inserted on the same command bar.

• Move method as it applies to the Assistant object.

Moves the Office Assistant to the specified location.

expression.Move(xLeft, yTop)

expression Required. An expression that returns an Assistant object.

xLeft Required **Integer**. The left position of the Office Assistant window, in points.

yTop Required **Integer**. The top position of the Office Assistant window, in points.

As it applies to the CommandBarButton, CommandBarComboBox, CommandBarControl, and CommandBarPopup objects.

This example moves the first combo box control on the command bar named Custom to the position before the seventh control on that command bar. The example sets the tag to "Selection box" and assigns the control a low priority so that it will likely be dropped from the command bar if all the controls don't fit in one row.

```
Set allcontrols = CommandBars("Custom").Controls
For Each ctrl In allControls
    If ctrl.Type = msoControlComboBox Then
        With ctrl
        .Move Before:=7
        .Tag = "Selection box"
        .Priority = 5
        End With
        Exit For
        End If
Next
```

```
• As it applies to the Assistant object.
```

This example displays the Office Assistant in the specified location and sets several options before making it visible.

```
With Assistant
   .Reduced = True
   .Move xLeft:= 400, yTop:= 300
   .MoveWhenInTheWay = True
   .TipOfDay = True
   .Visible = True
   .Animation = msoAnimationGreeting
End With
```

NewSearch Method

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

Resets all the search criteria settings to their default settings.

expression.NewSearch

-

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

Search criteria settings are retained throughout an application session. Use this method every time you change search criteria. This method will not reset the value of the **LookIn** property.

This example uses the **NewSearch** method to reset the default search criteria before beginning a new search.

```
With Application.FileSearch
    .NewSearch
    .LookIn = "C:\My Documents"
    .SearchSubFolders = True
    .FileName = "run"
    .TextOrProperty = "San*"
    .MatchAllWordForms = True
    .FileType = msoFileTypeAllFiles
    If .Execute() > 0 Then
        MsgBox "There were " & .FoundFiles.Count & _
        " file(s) found."
        For i = 1 To .FoundFiles.Count
            MsgBox .FoundFiles(i)
        Next i
    Else
        MsgBox "There were no files found."
    End If
End With
```

- <u>Show All</u>

Open Method

Opens the specified HTML project or HTML project item in the Microsoft Script Editor in one of the views specified by the optional **MsoHTMLProjectOpen** constants listed below. If one of the constants is not specified, the project item is opened in the default view.

expression.Open(OpenKind)

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

OpenKind Optional <u>MsoHTMLProjectOpen</u>. The view in which the specified project or project item is opened.

MsoHTMLProjectOpen can be one of these MsoHTMLProjectOpen constants. msoHTMLProjectOpenSourceView msoHTMLProjectOpenTextView

The default view is determined by whether or not the Microsoft Script Editor is open when the **Open** method is executed. If the Script Editor is not open, the **Open** method starts the Script Editor in source view. If the Script Editor is already open, the **Open** method activates the Script Editor and displays the script in the current view.

This example opens the HTML project in the active workbook in source view.

AppActiveWorkbook.HTMLProject.Open _
 (msoHTMLProjectOpenSourceView)

This example opens the first HTML project item in the active workbook in text view.

```
ActiveWorkbook.HTMLProject.HTMLProjectItems _
    Item(1).Open (msoHTMLProjectOpenTextView)
```

- <u>Show All</u>

PasteFace Method

Pastes the contents of the Clipboard onto a command bar <u>button control</u>.

expression.PasteFace

-

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

This example finds the built-in **FileOpen** button and pastes the face from the **Spelling and Grammar** button onto it from the Clipboard.

Set myControl = CommandBars.FindControl(Type:=msoControlButton, Id:=
myControl.CopyFace
Set myControl = CommandBars.FindControl(Type:=msoControlButton, Id:=
myControl.PasteFace

RefreshDocument Method

Refreshes the specified HTML project in the Microsoft Office host application.

expression.RefreshDocument(Refresh)

-

expression An expression that returns an **HTMLProject** object.

Refresh Required **Boolean**. **True** if all changes are to be saved; **False** if all changes are to be ignored.

Using this method is equivalent to clicking the **Refresh** button on the **Refresh** toolbar in the Office host application. If you refresh the document by setting the **RefreshDocument** method to **True**, all changes to the HTML source made in the Microsoft Script Editor are saved in the Office host application. If you set **RefreshDocument** to **False**, all changes to the HTML source are ignored. Note that the value returned by the **State** method is affected by the **RefreshDocument** method. If you call RefreshDocument (True), the **State** method returns **msoHTMLProjectStateDocumentProjectUnlocked** if it is called after the refresh operation.

This example refreshes the HTML project in the active workbook in the host application.

ActiveWorkbook.HTMLProject.RefreshDocument (True)

RefreshProject Method

Refreshes the specified HTML project in the Microsoft Script Editor.

expression.RefreshProject

-

expression An expression that returns an **HTMLProject** object.

Using this method is equivalent to clicking the **Refresh** button on the **Refresh** toolbar in the Microsoft Script Editor. If you refresh the document by setting **RefreshDocument** to **True**, all changes to the HTML source made in the Office host application are saved to the HTML project in the Microsoft Script Editor. If you set **RefreshDocument** to **False**, all changes to the HTML source are ignored.

This example refreshes the HTML project in the Microsoft Script Editor.

ActiveWorkbook.HTMLProject.RefreshProject (True)

RefreshScopes Method

Refreshes the list of currently available **ScopeFolder** objects.

expression.RefreshScopes

-

expression Required. An expression that returns one of the objects in the Applies To list.
The following example displays all of the currently available **ScopeFolder** objects on the C:\ drive in the My Computer scope.

```
Sub TestRefreshScopesMethod()
' Displays what happens before and after the RefreshScopes
' method is called when a new folder is added to the list
' of scope folders.
    ' List before the folder is created.
    Call ListFolderNames
    ' Create a new folder on the C:\ drive in My Computer.
    ' An error will occur if this folder already exists.
    MkDir Path:="C:\Delete_After_Using"
    ' List after the folder is created.
    Call ListFolderNames
    ' Refresh the list of folders.
    Application.FileSearch.RefreshScopes
    ' The newly-created folder now appears in the list.
    Call ListFolderNames
End Sub
Sub ListFolderNames()
    Dim strResults As String
    ' Loop through all the folder names on the C:\ drive
     in My Computer and report the results.
    '.SearchScopes.Item(1) = "My Computer"
    ' .ScopeFolders.Item(2) = "C:\"
   With Application.FileSearch.SearchScopes.Item(1). _
        ScopeFolder.ScopeFolders.Item(2)
        For i = 1 To .ScopeFolders.Count
            strResults = strResults & .ScopeFolders. _
                Item(i).Name & vbCrLf
        Next i
        MsgBox "Folder Names on C:\...." & vbCrLf & strResults
```

End With

End Sub

- <u>Show All</u>

ReleaseFocus Method

Releases the user interface focus from all <u>command bars</u>.

expression.ReleaseFocus

-

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

This example adds three blank buttons to the command bar named "Custom" and sets the focus to the center button. The example then waits five seconds before releasing the user interface focus from all command bars.

```
Set myBar = CommandBars
    .Add(Name:="Custom", Position:=msoBarTop, _
    Temporary:=True)
With myBar
    .Controls.Add Type:=msoControlButton
    .Controls.Add Type:=msoControlButton
    .Controls.Add Type:=msoControlButton
    .Visible = True
End With
Set myControl = CommandBars("Custom").Controls(2)
With myControl
    .SetFocus
End With
PauseTime = 5 ' Set duration.
    Start = Timer ' Set start time.
    Do While Timer < Start + PauseTime
       DoEvents ' Yield to other processes.
    Loop
    Finish = Timer
CommandBars.ReleaseFocus
```

- <u>Show All</u>

Remove Method

▶ <u>Remove method as it applies to the **NewFile** object.</u>

Removes an item from the **New** *Item* task pane. Returns a **Boolean**.

expression.Remove(FileName, Section, DisplayName, Action)

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

FileName Required String. The name of the file reference.

Section Optional **Variant**. The section of the task pane where the file reference exists. Can be any **msoFileNewSection** constant.

DisplayName Optional Variant. The display text of the file reference.

Action Optional **Variant**. The action taken when a user clicks on the item. Can be any **msoFileNewAction** constant.

Remove method as it applies to the FileTypes, PropertyTests, and SearchFolders objects.

Removes the specified object from the collection.

expression.Remove(Index)

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

Index Required **Long**. The index number of the property test to be removed.

▶ <u>As it applies to the **FileTypes**</u>, **PropertyTests**, and **SearchFolders** objects.

This example removes the first search criterion from the collection.

Application.FileSearch.PropertyTests.Remove(1)

▶ <u>As it applies to the **NewFile** object</u>.

This example removes the specified item from Word's **NewDocument** task pane.

```
Sub RemoveDocFromTaskPane()
    Application.NewDocument.Remove FileName:="C:\Newfile.doc", _
        Section:=msoNewfromTemplate, DisplayName:="NewFile"
        CommandBars("Task Pane").Visible = True
End Sub
```

- <u>Show All</u>

RemoveItem Method

Removes an item from a command bar <u>combo box control</u>.

Note The property fails when applied to controls other than list controls.

expression.RemoveItem(Index)

_

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

Index Required **Long**. The item to be removed from the list.

This example determines whether there are more than three items in the specified combo box. If there are more than three items, the example removes the second item, alters the style, and sets a new value. It also sets the **Tag** property of the parent object (the **CommandBarControl** object) to show that the list has changed.

```
Set myBar = CommandBars
    .Add(Name:="Custom", Position:=msoBarTop, _
    Temporary:=True)
With myBar
    .Controls.Add Type:=msoControlComboBox, ID:=1
    .Visible = True
End With
With CommandBars("Custom").Controls(1)
    .AddItem "Get Stock Quote", 1
    .AddItem "View Chart", 2
    .AddItem "View Fundamentals", 3
    .AddItem "View News", 4
    .Caption = "Stock Data"
    .DescriptionText = "View Data For Stock"
End With
Set myControl = myBar.Controls(1)
With myControl
    If .ListCount > 3 Then
        .RemoveItem 2
        .Style = msoComboNormal
        .Text = "New Default"
         Set ctrl = .Parent
    End If
End With
```

- <u>Show All</u>

Reset Method

Resets a built-in <u>command bar</u> to its default configuration, or resets a built-in <u>command bar control</u> to its original function and face.

expression.Reset

-

expression Required. An expression that returns a **CommandBar**, **CommandBarControl**, **CommandBarButton**, **CommandBarPopup**, or **CommandBarComboBox** object.

Remarks

Resetting a built-in control restores the actions originally intended for the control and resets each of the control's properties back to its original state. Resetting a built-in command bar removes custom controls and restores built-in controls.

This example uses the value of user to adjust the command bars according to the user level. If user is "Level 1," the command bar named "Custom" is displayed. If user is any other value, the built-in Visual Basic command bar is reset to its default state and the command bar named "Custom" is disabled.

```
Set myBar = CommandBars("Custom")
If user = "Level 1" Then
    myBar.Visible = True
Else
    CommandBars("Visual Basic").Reset
    myBar.Enabled = False
End If
```

ResetFileList Method

Resets the list of files for the current AnswerWizard to the default list of files for the Microsoft Office host application.

expression.ResetFileList

-

expression An expression that returns an **AnswerWizard** object.

Remarks

Use this method to restore all entries in the current AnswerWizard file list to the list in the Windows registry for the host application. You can establish a custom default file list in the registry by adding the names of the custom files to the appropriate registry key.

This example resets the file list for the current AnswerWizard and then displays both the file count and the file names in a message box.

```
Dim customAnswerWizard As AnswerWizard
Dim strFileList As String
Dim intCounter As Integer
Dim intNumFiles As Integer
Set customAnswerWizard = Application.AnswerWizard
intCounter = 1
customAnswerWizard.ResetFileList
strFileList = ""
intNumFiles = customAnswerWizard.Files.Count
For intCounter = 1 To (intNumFiles)
    strFileList = strFileList & _
    customAnswerWizard.Files.Item(intCounter) & Chr(13)
Next
MsgBox "There are " & customAnswerWizard.Files.Count & _
    " files avaialble through this AnswerWizard: " & _
    Chr(13) & strFileList
```

ResetTips Method

Some of the content in this topic may not be applicable to some languages. Resets the application tips that appear in the Office Assistant balloon.

expression.ResetTips

-

expression Required. An expression that returns an **Assistant** object.

Remarks

The **ResetTips** method corresponds to the **Reset my tips** button on the **Options** tab in the **Office Assistant** dialog box.

This example resets the application tips before making the Office Assistant visible. A confirmation balloon will appear, telling the user that his or her application tips have been reset.

```
With Application.Assistant
    .On = True
    .Visible = True
    .Animation = msoAnimationGreeting
    .ResetTips
End With
```

SaveCopyAs Method

Saves the specified HTML project item using a new file name.

expression.SaveCopyAs(Filename)

-

expression An HTMLProjectItem object.

Filename Required **String**. The fully qualified path of the file to which you want to save the HTML project item.

This example saves a copy of the text of the current HTML project item to the file NewScript.txt.

ActiveWorkbook.HTMLProject.HTMLProjectItems. _
 Item(1).SaveCopyAs("C:\NewScript.txt")

SetAvoidRectangle Method

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

Prevents the Office Assistant balloon from being displayed in a specified area of the screen.

expression.SetAvoidRectangle(Left, Top, Right, Bottom)

expression Required. An expression that returns an Assistant object.

Left, Top, Right, Bottom Required **Long**. The coordinates (in points and relative to the screen) of the area of the screen that the Office Assistant balloon will avoid when it's displayed.

Remarks

This property is intended to prevent the Office Assistant balloon from overlapping custom dialog boxes and wizards.

This example prevents the Office Assistant balloon represented by the variable myBalloon from being displayed in the region of the screen denoted by the specified coordinates.

```
Set myBalloon = Assistant.NewBalloon
With myBalloon
.SetAvoidRectangle 300, 250, 700, 500
.Text = "Cannot display in coordinates " & _
            "300, 250, 700, 500."
.Show
End With
```

- <u>Show All</u>

SetFocus Method

-

Moves the keyboard focus to the specified <u>command bar control</u>. If the control is disabled or isn't visible, this method will fail.

Remarks

The focus on the control is subtle. After you use this method, you will notice a three dimensional highlight on the control. Pressing the arrow keys will navigate in the toolbars, as if you had arrived at the control by pressing only keyboard controls.

expression.SetFocus

expression Required. An expression that returns a **CommandBarControl**, **CommandBarButton**, **CommandBarPopup**, or **CommandBarComboBox** object.

This example creates a command bar named "Custom" and adds a **ComboBox** control and a **Button** control to it. The example then uses the **SetFocus** method to set the focus to the **ComboBox** control.

```
Set focusBar = CommandBars.Add(Name:="Custom")
With CommandBars("Custom")
    .Visible = True
    .Position = msoBarTop
End With
Set testComboBox = CommandBars("Custom").Controls _
    .Add(Type:=msoControlComboBox, ID:=1)
With testComboBox
    .AddItem "First Item", 1
    .AddItem "Second Item", 2
End With
Set testButton = CommandBars("Custom").Controls _
    .Add(Type:=msoControlButton)
testButton.FaceId = 17
' Set the focus to the combo box.
testComboBox.SetFocus
```

SetSortOrder Method

Sets the sort order for mail merge data.

expression.SetSortOrder(SortField1, SortAscending1, SortField2, SortAscending2, SortField3, SortAscending3)

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

SortField1 Required *String*. The first field on which to sort the mail merge data.

SortAscending1 Optional **Boolean**. **True** (default) to perform an ascending sort on *SortField1*; **False** to perform a descending sort.

SortField2 Optional *String*. The second field on which to sort the mail merge data. Default is an empty string.

SortAscending2 Optional **Boolean**. **True** (default) to perform an ascending sort on *SortField2*; **False** to perform a descending sort.

SortField3 Optional *String*. The third field on which to sort the mail merge data. Default is an empty string.

SortAscending3 Optional **Boolean**. **True** (default) to perform an ascending sort on *SortField3*; **False** to perform a descending sort.

The following example sorts the data source first according to ZIP code in descending order, then on last name and first name in ascending order.

```
Sub SetDataSortOrder()
Dim appOffice As OfficeDataSourceObject
Set appOffice = Application.OfficeDataSourceObject
appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName
"UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
appOffice.SetSortOrder SortField1:="ZipCode", _
SortAscending1:=False, SortField2:="LastName", _
SortField3:="FirstName"
End Sub
```

- <u>Show All</u>

Show Method

▶ <u>Show method as it applies to the</u> **Balloon** object.

Displays the specified balloon object. Returns an **MsoBalloonButtonType** constant that indicates which button or label the user clicks. Read-only.

MsoBalloonButtonType can be one of these MsoBalloonButtonType constants.

msoBalloonButtonAbort msoBalloonButtonBack **msoBalloonButtonCancel** msoBalloonButtonClose msoBalloonButtonIgnore msoBalloonButtonNext msoBalloonButtonNo msoBalloonButtonNull **msoBalloonButtonOK msoBalloonButtonOptions msoBalloonButtonRetry msoBalloonButtonSearch** msoBalloonButtonSnooze **msoBalloonButtonTips** msoBalloonButtonYes msoBalloonButtonYesToAll

expression.Show

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

Show method as it applies to the FileDialog object.

Displays a file dialog box and returns a **Long** indicating whether the user

pressed the action button (-1) or the cancel button (0). When you call the **Show** method, no more code will execute until the user dismisses the file dialog box. In the case of SaveAs and Open dialog boxes, use the **Execute** method right after the **Show** method to carry out the user's action.

expression.Show

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

• As it applies to the **Balloon** object.

This example creates a balloon containing two balloon label choices for setting printer orientation: **Portrait** and **Landscape**. The example uses the **Show** method in a **Select Case** statement to determine which orientation the user has chosen.

```
Set balNew = Assistant.NewBalloon
With balNew
    .Heading = "Please choose a printer orientation"
    .Labels(1).Text = "Portrait"
    .Labels(2).Text = "Landscape"
    .Button = msoButtonSetNone
End With
Select Case balNew.Show
    Case 1
        ' Insert code to set printer to Portrait.
    Case 2
        ' Insert code to set printer to Landscape.
End Select
```

This example creates a balloon containing three command buttons: **Yes**, **No**, and **Cancel**. The example uses the **Show** method in a **Select Case** statement to determine the return value of the button clicked by the user.

```
Set balNew = Assistant.NewBalloon
With balNew
    .Heading = "Are you sure you want to set the " & _
    "printer orientation to Landscape?"
    .BalloonType = msoBalloonTypeButtons
    .Button = msoButtonSetYesNoCancel
End With
Select Case balNew.Show
             ' User selected Cancel button.
    Case -2
        returnValue = MsgBox("Operation canceled.", _
       vbOKOnly, "Printer Message")
                ' User selected Yes button.
    Case -3
        returnValue = MsgBox("Printer set to " & _
        "Landscape.", vbOKOnly, "Printer Message")
    Case -4 'User selected No button.
```
```
returnValue = MsgBox("Printer orientation not " & _
"reset.", vbOKOnly, "Printer Message")
End Select
```

• As it applies to the **FileDialog** object.

The following example displays a File Picker dialog box using the **FileDialog** object and displays each selected file in a message box.

```
Sub Main()
    'Declare a variable as a FileDialog object.
    Dim fd As FileDialog
    'Create a FileDialog object as a File Picker dialog box.
    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    'Declare a variable to contain the path
    'of each selected item. Even though the path is a String,
    'the variable must be a Variant because For Each...Next
    'routines only work with Variants and Objects.
    Dim vrtSelectedItem As Variant
    'Use a With...End With block to reference the FileDialog object.
   With fd
        'Use the Show method to display the File Picker dialog box a
        'The user pressed the action button.
        If . Show = -1 Then
            'Step through each string in the FileDialogSelectedItems
            For Each vrtSelectedItem In .SelectedItems
                'vrtSelectedItem is a string that contains the path
                'You can use any file I/O functions that you want to
                'This example simply displays the path in a message
                MsgBox "The path is: " & vrtSelectedItem
            Next vrtSelectedItem
        'The user pressed Cancel.
        Else
        End If
    End With
    'Set the object variable to nothing.
    Set fd = Nothing
```

End Sub

- <u>Show All</u>

ShowPopup Method

Displays a <u>command bar</u> as a shortcut menu at the specified coordinates or at the current pointer coordinates.

Note If the **Position** property of the command bar is not set to **msoBarPopup**, this method fails.

```
expression.ShowPopup(x, y)
```

_

expression Required. An expression that returns a CommandBar object.

x Optional **Variant**. The x-coordinate for the location of the shortcut menu. If this argument is omitted, the current x-coordinate of the pointer is used.

y Optional **Variant**. The y-coordinate for the location of the shortcut menu. If this argument is omitted, the current y-coordinate of the pointer is used.

This example creates a shortcut menu containing two controls. The **ShowPopup** method is used to make the shortcut menu visible.

```
Set myBar = CommandBars _
.Add(Name:="Custom", Position:=msoBarPopup, Temporary:=False)
With myBar
.Controls.Add Type:=msoControlButton, Id:=3
.Controls.Add Type:=msoControlComboBox
End With
myBar.ShowPopup
```

StartWizard Method

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

Starts the Office Assistant and returns a **Long** value that identifies the session. You should use this method only to run the Office Assistant in a custom wizard.

Note The number returned by **StartWizard** method is used by the **ActivateWizard** and **EndWizard** methods.

expression.StartWizard(On, Callback, PrivateX, Animation, CustomTeaser, Top, Left, Bottom, Right)

expression Required. An expression that returns an **Assistant** object.

On Required **Boolean**. **True** to display the Office decision balloon. The Office decision balloon asks the user whether he or she wants help with the active custom wizard. It isn't necessary to use the <u>Visible</u> property to display the Office Assistant if you specify **True** for this argument.

Callback Required String. The name of the callback procedure run by the Office decision balloon and the branch balloon. The branch balloon allows the user to choose between custom Help you've provided for the wizard and standard Office Help.

PrivateX Required Long. A number that identifies the balloon that initiated the callback procedure.

Animation Optional **Variant**. The animation the Office Assistant performs when this method is used. The default value is **msoAnimationGetWizardy**.

CustomTeaser Optional Variant. False to display the Office decision balloon.

Top, *Left*, *Bottom*, *Right* Optional Variant. The position of the corners (in points and relative to the screen) of the custom wizard form the Office Assistant will avoid when the Office Assistant appears.

Remarks

Unlike callback procedures used by standard modeless balloons, the callback procedure called by the modeless decision and branch balloons displayed during an Office Assistant wizard session takes only two arguments: an **MsoWizardMsgType** constant, and the unique value specified by the **PrivateX** argument of the **StartWizard** method.

If the user clicks the left button in the decision or branch balloon, the constant **msoWizardMsgShowHelp** is passed to the first argument of the callback procedure. If the user clicks the right button, the constant **msoWizardLocalStateOff** is passed. (The other **MsoWizardMsgType** constants are passed by the **ActivateWizard** method if you've specified **msoWizardActResume** or **msoWizardActSuspend** for the **Act** argument.) In the case of **msoWizardMsgShowHelp**, the callback procedure should display the appropriate balloon for the current panel of the custom wizard. And in the case of **msoWizardLocalStateOff**, the callback procedure should hide the visible balloon.

This example starts the Office Assistant as part of a process to provide information while a custom wizard is running. The variable lHelpForWiz is set to the return value of the **StartWizard** method, which is **Long**.

lHelpForWiz = Assistant.StartWizard(On:=True, _ Callback:="myCallback", PrivateX:=23)

Update Method

Updates the contents of the **COMAddIns** collection from the list of add-ins stored in the Windows registry.

expression.Update

_

expression The **COMAddIns** collection.

Remarks

Before you can use a given COM add-in in a Microsoft Office application, that add-in must be registered in the Windows registry as a COM component with a corresponding Component Category ID.

The following example updates the contents of the **COMAddIns** collection from the list of add-ins stored in the Windows registry.

Application.COMAddIns.Update

ActionControl Property

-

Returns the **CommandBarControl** object whose **OnAction** property is set to the running procedure. If the running procedure was not initiated by a command bar control, this property returns **Nothing**. Read-only.

This example creates a command bar named "Custom", adds three buttons to it, and then uses the **ActionControl** property and the **Tag** property to determine which command bar button was last clicked.

```
Set myBar = CommandBars
    .Add(Name:="Custom", Position:=msoBarTop, _
    Temporary:=True)
Set buttonOne = myBar.Controls.Add(Type:=msoControlButton)
With buttonOne
    FaceId = 133
    .Tag = "RightArrow"
    .OnAction = "whichButton"
End With
Set buttonTwo = myBar.Controls.Add(Type:=msoControlButton)
With buttonTwo
    .FaceId = 134
    .Tag = "UpArrow"
    .OnAction = "whichButton"
End With
Set buttonThree = myBar.Controls.Add(Type:=msoControlButton)
With buttonThree
    .FaceId = 135
    .Tag = "DownArrow"
    .OnAction = "whichButton"
End With
myBar.Visible = True
```

The whichButton subroutine responds to the **OnAction** method and determines which command bar button was last clicked.

```
Sub whichButton()
Select Case CommandBars.ActionControl.Tag
    Case "RightArrow"
        MsgBox ("Right Arrow button clicked.")
    Case "UpArrow"
        MsgBox ("Up Arrow button clicked.")
    Case "DownArrow"
        MsgBox ("Down Arrow button clicked.")
End Select
End Sub
```

ActiveMenuBar Property

-

Returns a **<u>CommandBar</u>** object that represents the active menu bar in the container application. Read-only.

This example adds a temporary pop-up control named "Custom" to the end of the active menu bar, and adds a control named "Import" to the pop-up control.

```
Set myMenuBar = CommandBars.ActiveMenuBar
Set newMenu = myMenuBar.Controls.Add(Type:=msoControlPopup, Temporar
newMenu.Caption = "Custom"
Set ctrl1 = newMenu.CommandBar.Controls _
    .Add(Type:=msoControlButton, Id:=1)
With ctrl1
    .Caption = "Import"
    .TooltipText = "Import"
    .Style = msoButtonCaption
End With
```

AdaptiveMenu Property

_

True if a personalized menu is enabled. Read/write **Boolean**.

This example sets the **AdaptiveMenu** property to **False** for the **File** menu in the Microsoft Office application you're working in.

CommandBars("File").AdaptiveMenu = False

AdaptiveMenus Property

-

True if adaptive menus are enabled. Read/write **Boolean**.

This example sets three options for all command bars in Microsoft Office, including custom command bars and the controls on them.

```
With CommandBars
.LargeButtons = True
.DisplayFonts = True
.AdaptiveMenus = True
End With
```

AllowMultiSelect Property

True if the user is allowed to select multiple files from a file dialog box. Read/write **Boolean**.

$expression. {\bf Allow MultiSelect}$

-

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

Remarks

This property has no effect on Folder Picker dialog boxes or SaveAs dialog boxes because users should never be able to select multiple files in these types of file dialog boxes.

The following example displays a File Picker dialog box using the **FileDialog** object and displays each selected file in a message box.

```
Sub Main()
    'Declare a variable as a FileDialog object.
    Dim fd As FileDialog
    'Create a FileDialog object as a File Picker dialog box.
    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    'Declare a variable to contain the path
    'of each selected item. Even though the path is a String,
    'the variable must be a Variant because For Each...Next
    'routines only work with Variants and Objects.
    Dim vrtSelectedItem As Variant
    'Use a With...End With block to reference the FileDialog object.
   With fd
        'Allow the selection of multiple files.
        .AllowMultiSelect = True
        'Use the Show method to display the file picker dialog and r
        'If the user presses the action button...
        If .Show = -1 Then
            'Step through each string in the FileDialogSelectedItems
            For Each vrtSelectedItem In .SelectedItems
                'vrtSelectedItem is a String that contains the path
                'You can use any file I/O functions that you want to
                'This example simply displays the path in a message
                MsgBox "Selected item's path: " & vrtSelectedItem
            Next
        'If the user presses Cancel...
        Else
        End If
    End With
    'Set the object variable to Nothing.
    Set fd = Nothing
```

End Sub

Animation Property

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

Returns or sets an animation action for the Office Assistant. When this property is applied to the **Assistant** object and the Assistant supports the specified animation, the Assistant is animated immediately (if the Assistant is visible and enabled). When this property is applied to the **Balloon** object, the Assistant is animated only while the balloon is displayed. Read/write <u>MsoAnimationType</u>.

MsoAnimationType can be one of these MsoAnimationType constants. **msoAnimationAppear msoAnimationBeginSpeaking** msoAnimationCharacterSuccessMajor msoAnimationCheckingSomething **msoAnimationDisappear msoAnimationEmptyTrash msoAnimationGestureDown msoAnimationGestureLeft msoAnimationGestureRight msoAnimationGestureUp msoAnimationGetArtsy** msoAnimationGetAttentionMajor **msoAnimationGetAttentionMinor msoAnimationGetTechy msoAnimationGetWizardy msoAnimationGoodbye msoAnimationGreeting msoAnimationIdle** msoAnimationListensToComputer **msoAnimationLookDown**

msoAnimationLookDownLeft msoAnimationLookDownRight msoAnimationLookLeft msoAnimationLookRight msoAnimationLookUp msoAnimationLookUpLeft msoAnimationPrinting msoAnimationPrinting msoAnimationRestPose msoAnimationSaving msoAnimationSearching msoAnimationSendingMail msoAnimationThinking msoAnimationWorkingAtSomething

Remarks

"Clippit" is the default Assistant, and **msoAnimationIdle** is the default animation type for the Assistant.

Depending on the selected Assistant, setting the **Animation** property may or may not result in an obvious animation. However, all **MsoAnimationType** constants are valid for all Assistants. Note that different constants may produce the same animation.

The following **MsoAnimationType** constants represent animations that repeat the specified action until the Assistant is dismissed or until the **Animation** property is reset with another animation:

msoAnimationCheckingSomething	
	msoAnimationThinking
msoAnimationGetTechy	
	msoAnimationWorkingAtSomething
msoAnimationListensToComputer	
-	msoAnimationWritingNotingSomething
msoAnimationSearching	

This example displays the Office Assistant in a specific location and it sets several options before making the Assistant visible.

```
With Assistant
   .On = True
   .Visible = True
   .Move xLeft:= 400, yTop:= 300
   .MoveWhenInTheWay = True
   .TipOfDay = True
   .Animation = msoAnimationGreeting
End With
```

Application Property

Returns an **Application** object that represents the container application for the object (you can use this property with an Automation object to return that object's container application).

expression. Application

-

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

This example returns the name of the application in which the command bar named Standard was created and displays this result in a message box.

Set Appobj = CommandBars("Standard").Application
MsgBox Appobj

AssistWithAlerts Property

-

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

True if the Office Assistant balloon delivers application alerts when the Office Assistant is visible. Read/write **Boolean**.

Remarks

The **AssistWithAlerts** property corresponds to the **Display alerts** option under **Use the Office Assistant** on the **Options** tab in the **Office Assistant** dialog box.

If this property is set to **False**, the application displays alerts in dialog boxes.

This example sets the Office Assistant to be displayed whenever an application alert is generated.

```
With Assistant
    .On = True
    .Visible = True
    .AssistWithHelp = True
    .Animation = msoAnimationGetAttentionMajor
End With
```

AssistWithHelp Property

-

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

True if the Office Assistant appears when the user presses the F1 key to display Help. Read/write **Boolean**.

Remarks

The **AssistWithHelp** property corresponds to the **Respond to F1 key** option under **Use the Office Assistant** on the **Options** tab in the **Office Assistant** dialog box.

If this property is set to **False**, the **Help Topics** dialog box appears instead of the Office Assistant.

This example displays the Office Assistant whenever the user presses the F1 key to display Help.

```
With Assistant
    .On = True
    .Visible = True
    .AssistWithHelp = True
    .AssistWithAlerts = True
    .Animation = msoAnimationGetAttentionMajor
End With
```
AssistWithWizards Property

-

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

True if the Office Assistant provides online Help with wizards. Read/write **Boolean**.

Remarks

The **AssistWithWizards** property corresponds to the **Help with wizards** option under **Use the Office Assistant** on the **Options** tab in the **Office Assistant** dialog box.

This example sets the Office Assistant to provide Help information about wizards.

Assistant.**AssistWithWizards** = True

AttachCertificate Property

True if the digital certificate that corresponds to the specified **Signature** object is attached to the document. Read/write **Boolean**.

expression.AttachCertificate

-

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

The following example prompts the user to select a digital signature with which to sign the active document in Microsoft Word. To use this example, open a document in Word and call this function. The function will test to make sure that the digital signature that the user selects will not expire in less than 12 months. If it will expire, the certificate isn't attached.

```
Function AddSignature() As Boolean
    On Error GoTo Error Handler
    Dim sig As Signature
    'Display the dialog box that lets the
    'user select a digital signature.
    'If the user selects a signature, then
    'it is added to the Signatures
    'collection. If the user doesn't, then
    'an error is returned.
    Set sig = ActiveDocument.Signatures.Add
    sig.AttachCertificate = True
    'Test to make sure that the new Signature object
    'doesn't expire too soon. This expression calculates
    'the number of months until the Signature object expires.
    If DateDiff("m", sig.SignDate, sig.ExpireDate) < 12 Then</pre>
        MsgBox "This certificate will expire in less than 1 year." &
        "Please use a newer certificate."
        AddSignature = False
        sig.Delete
    Else
        AddSignature = True
    End If
    'Commit all signatures in the SignatureSet collection to the dis
    ActiveDocument.Signatures.Commit
    Exit Function
Error Handler:
    AddSignature = False
    MsgBox "Action cancelled."
End Function
```

BalloonError Property

Returns a value that indicates the last recorded balloon error. Read-only **MsoBalloonErrorType**.

MsoBalloonErrorType can be one of these MsoBalloonErrorType constants. **msoBalloonErrorBadCharacter** The balloon contains an ASCII control character other than CR or LF and less than 32.

msoBalloonErrorBadPictureRef The balloon contains a graphic that couldn't be displayed because the file doesn't exist or because the graphic isn't a valid .BMP or .WMF file.

msoBalloonErrorBadReference The balloon contains an unrecognized or unsupported reference.

msoBalloonErrorButtonlessModal The balloon you attempted to display is modal, but it contains no buttons. The balloon won't be shown because it can't be dismissed.

msoBalloonErrorButtonModeless The balloon you attempted to display is modeless, contains buttons, and has no procedure assigned to the **Callback** property. The balloon won't be shown because a callback procedure is required for modeless balloons.

msoBalloonErrorCharNotTopmostForModal The modal balloon was requested by an application that isn't the active application. Microsoft Office renders balloons for the active (topmost) application only.

msoBalloonErrorCOMFailure The balloon could not be displayed because of a COM failure.

msoBalloonErrorNone No error was encountered.

msoBalloonErrorOther The balloon won't appear because some other error occurred, such as another modal balloon is already active.

msoBalloonErrorOutOfMemory The balloon won't appear because there is insufficient memory.

msoBalloonErrorTooBig The balloon is too big to appear on the screen.

msoBalloonErrorTooManyControls The balloon contains more than twenty

controls (check boxes or labels).

This example creates a balloon that generates an error. The error is generated because the balloon is created without a way to dismiss it: the button type is set to **msoButtonSetNone** and the default balloon mode is **msoModeModal**, resulting in a buttonless, modal balloon. Note that there's no way to dismiss a buttonless modal balloon.

```
With Application.Assistant
With .NewBalloon
    .Heading = "This will never show."
    .Text = "Imagine a balloon here."
    .Button = msoButtonSetNone
    .Show
End With
.Visible = True
If .BalloonError = msoBalloonErrorButtonlessModal Then
    MsgBox "You need a button to dismiss the balloon."
End If
End With
```

BalloonType Property

_

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

Returns or sets the type of balloon the Office Assistant uses. When you create a **Balloon** object, this property is initially set to **msoBalloonTypeButtons**. Read/write **MsoBalloonType**.

MsoBalloonType can be one of these MsoBalloonType constants. msoBalloonTypeBullets msoBalloonTypeButtons msoBalloonTypeNumbers

This example creates an instruction balloon that explains how to select a printer. The balloon is modeless, so the user can follow the instructions in the balloon and keep the balloon visible as he or she works.

```
Set bln = Assistant.NewBalloon
With bln
.Heading = "Instructions for Choosing a Printer."
.Text = "Click OK when you've chosen a printer."
.Labels(1).Text = "From the File menu, choose Print."
.Labels(2).Text = "Click Setup."
.Labels(3).Text = "Select the name of the printer."
.BalloonType = msoBalloonTypeNumbers
.Mode = msoModeModeless
.Callback = "ProcessPrinter"
.Button = msoButtonSetOK
.Show
End With
```

- <u>Show All</u>

BeginGroup Property

-

True if the specified <u>command bar control</u> appears at the beginning of a group of controls on the command bar. Read/write **Boolean**.

This example begins a new group with the last control on the active menu bar.

```
Set myMenuBar = CommandBars.ActiveMenuBar
Set lastMenu = myMenuBar _
    .Controls(myMenuBar.Controls.Count)
lastMenu.BeginGroup = True
```

- <u>Show All</u>

BuiltIn Property

-

True if the specified <u>command bar</u> or <u>command bar control</u> is a built-in command bar or control of the container application. **False** if it's a custom command bar or control, or if it's a built-in control whose **OnAction** property has been set. Read-only **Boolean**.

This example deletes all custom command bars that aren't visible.

```
foundFlag = False
deletedBars = 0
For Each bar In CommandBars
    If (bar.BuiltIn = False) And (bar.Visible = False) Then
        bar.Delete
        foundFlag = True
        deletedBars = deletedBars + 1
        End If
Next
If Not foundFlag Then
        MsgBox "No command bars have been deleted."
Else
        MsgBox deletedBars & " custom command bar(s) deleted."
End If
```

- <u>Show All</u>

BuiltInFace Property

-

True if the face of a command bar <u>button control</u> is its original built-in face. This property can only be set to **True**, which will reset the face to the built-in face. Read/write **Boolean**.

This example determines whether the face of the first control on the command bar named "Custom" is its built-in button face. If it is, the example copies the button face to the Clipboard.

Button Property

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

Returns or sets the type of button displayed at the bottom of the Office Assistant balloon. When you create a **Balloon** object, this property is initially set to **msoButtonSetOK**. Read/write **MsoButtonSetType**.

MsoButtonSetType can be one of these MsoButtonSetType constants.

msoButtonSetAbortRetryIgnore msoButtonSetBackClose msoButtonSetBackNextClose msoButtonSetBackNextSnooze msoButtonSetCancel msoButtonSetCancel msoButtonSetNone msoButtonSetNone msoButtonSetOKCancel msoButtonSetOkCancel msoButtonSetRetryCancel msoButtonSetSearchClose msoButtonSetTipsOptionsClose msoButtonSetYesAllNoCancel msoButtonSetYesNo

This example displays a balloon that contains a heading, text, three region choices, and two command buttons (**OK** and **Cancel**).

```
With Assistant.NewBalloon
   .Heading = "Regional Sales Data"
   .Text = "Select a region"
   For i = 1 To 3
        .CheckBoxes(i).Text = "Region " & i
   Next
   .Button = msoButtonSetOkCancel
   .Show
End With
```

ButtonName Property

Sets or returns a **String** representing the text that is displayed on the action button of a file dialog box. By default, this property is set to the standard text for the type of file dialog box. For example, in the case of the Open dialog box, the property is set to "Open" by default. Read/write.

expression.ButtonName

-

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

The following example displays a File Picker dialog box using the **FileDialog** object and displays each selected file in a message box.

```
Sub Main()
    'Declare a variable as a FileDialog object.
    Dim fd As FileDialog
    'Create a FileDialog object as a File Picker dialog box.
    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    'Declare a variable to contain the path
    'of each selected item. Even though the path is a String,
    'the variable must be a Variant because For Each...Next
    'routines only work with Variants and Objects.
    Dim vrtSelectedItem As Variant
    'Use a With...End With block to reference the FileDialog object.
   With fd
        'Change the text on the action button.
        .ButtonName = "Archive"
        'Use the Show method to display the File Picker dialog box a
        'If the user presses the action button...
        If .Show = -1 Then
            'Step through each String in the FileDialogSelectedItems
            For Each vrtSelectedItem In .SelectedItems
                'vrtSelectedItem is a String that contains the path
                'You can use any file I/O functions that you want to
                'This example simply displays the path in a message
                MsgBox "Selected item's path: " & vrtSelectedItem
            Next vrtSelectedItem
        'If the user presses Cancel...
        Else
        End If
    End With
    'Set the object variable to Nothing.
    Set fd = Nothing
```

End Sub

Callback Property

-

Sets the name of the procedure to run from a modeless balloon. Read/write **String**.

Remarks

The procedure you specify for the **Callback** property must be written to receive either two or three arguments, depending on what you use the property with. If you use the **Callback** property with a wizard, you must write the procedure to receive two arguments: a long integer that represents the **msoBalloonButtonType** value of the button that the user clicked, and a long integer that uniquely identifies the balloon. If you use the **Callback** property with a modeless balloon, you must write the procedure to receive three arguments: the **Balloon** object that called the procedure; a long integer that represents the **msoBalloonButtonType** value of the button the user clicked; and a long integer that uniquely identifies the balloon that called the procedure, as denoted in the balloon's **Private** property.

The callback procedure must contain at least one condition under which the **Close** method is applied to the **Balloon** object that is passed to it; otherwise, the modeless balloon cannot be dismissed.

If you specify a procedure that is stored in a separate class module, you must include the module name in the value assigned to the **Callback** property (for example, "Sheet1.MyCallback).

This example displays a balloon that contains a button for each of three printers. Whenever the user clicks one of these buttons, the ProcessPrinter callback procedure is run and the balloon is closed.

```
Sub selectPrinter()
Set bln = Assistant.NewBalloon
With bln
    .Heading = "Select a Printer."
    .Labels(1).Text = "Network Printer"
    .Labels(2).Text = "Local Printer"
    .Labels(3).Text = "Local Color Printer"
    .BalloonType = msoBalloonTypeButtons
    .Mode = msoModeModeless
    .Callback = "ProcessPrinter"
    .Show
End With
End Sub
Sub ProcessPrinter(bln As Balloon, lbtn As Long, _
 lPriv As Long)
    Assistant.Animation = msoAnimationPrinting
    Select Case 1btn
    Case -1
        ' Insert network printer-specific code.
    Case -2
        ' Insert local printer-specific code.
    Case -3
        ' Insert color printer-specific code.
    End Select
    bln.Close
End Sub
```

- <u>Show All</u>

Caption Property

-

Returns or sets the caption text for a <u>command bar control</u>. Read/write **String**.

Note A control's caption is also displayed as its default ScreenTip.

This example adds a command bar control with a spelling checker button face to a custom command bar, and then it sets the caption to "Spelling checker."

```
Set myBar = CommandBars.Add(Name:="Custom", _
Position:=msoBarTop, Temporary:=True)
myBar.Visible = True
Set myControl = myBar.Controls _
.Add(Type:=msoControlButton, Id:=2)
With myControl
.DescriptionText = "Starts the spelling checker"
.Caption = "Spelling checker"
End With
```

Checkboxes Property

-

Returns the **BalloonCheckboxes** collection that represents all the check boxes contained in the specified balloon. Read-only.

For information about returning a single member of a collection, see <u>Returning</u> <u>an Object from a Collection</u>.

This example creates a balloon with a heading, text, and three region choices. When the user clicks **OK** in the balloon, data for the selected region or regions is printed.

```
With Assistant.NewBalloon
    .Heading = "Regional Sales Data"
    .Text = "Select the region(s) you want to print."
    For i = 1 To 3
        .CheckBoxes(i).Text = "Region " & i
    Next
    .Button = msoButtonSetOkCancel
    If .Show = msoBalloonButtonOK Then
        dataPrinted = 0
        For i = 1 To 3
            If .CheckBoxes(i).Checked = True Then
                ' Code to print region data.
                dataPrinted = dataPrinted + 1
                MsgBox "Region " & i & " data printed."
            End If
        Next
        If dataPrinted = 0 Then MsgBox "No data printed."
    End If
End With
```

Checked Property

-

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

True if the specified check box in the Office Assistant balloon is checked. Read/write **Boolean**.

This example creates a balloon with a heading, text, and three region choices. When the user clicks **OK** in the balloon, data for the selected region or regions is printed.

```
With Assistant.NewBalloon
    .Heading = "Regional Sales Data"
    .Text = "Select the region(s) you want to print."
    For i = 1 To 3
        .CheckBoxes(i).Text = "Region " & i
    Next
    .Button = msoButtonSetOkCancel
    If .Show = msoBalloonButtonOK Then
        dataPrinted = 0
        For i = 1 To 3
            If .CheckBoxes(i).Checked = True Then
                ' Code to print region data.
                dataPrinted = dataPrinted + 1
                MsgBox "Region " & i & " data printed."
            End If
        Next
        If dataPrinted = 0 Then MsgBox "No data printed."
    End If
End With
```

Column Property

Returns or sets a **String** that represents the name of the field in the mail merge data source to use in the filter. Read/write.

expression.Column

-

expression Required. An expression that returns one of the objects in the Applies To list.
The following example changes an existing filter to remove from the mail merge all records that do not have a Region field equal to "WA".

```
Sub SetQueryCriterion()
    Dim appOffice As Office.OfficeDataSourceObject
    Dim intItem As Integer
    Set appOffice = Application.OfficeDataSourceObject
    appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName
        "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
   With appOffice.Filters
        For intItem = 1 To .Count
            With .Item(intItem)
                If .Column = "Region" Then
                    .Comparison = msoFilterComparisonNotEqual
                    .CompareTo = "WA"
                    If .Conjunction = "Or" Then .Conjunction = "And"
                End If
            End With
        Next intItem
    End With
End Sub
```

Columns Property

Returns an **ODSOColumns** object that represents the fields in a data source.

expression.Columns

-

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

The following example displays the field names in the data source attached to the active publication.

```
Sub ShowFieldNames()
   Dim appOffice As OfficeDataSourceObject
   Dim intCount As Integer
   Set appOffice = Application.OfficeDataSourceObject
   appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName
      "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
   With appOffice.Columns
      For intCount = 1 To .Count
           MsgBox "Field Name: " & .Item(intCount).Name
           Next
   End With
End Sub
```

- <u>Show All</u>

CommandBar Property

-

Returns a **CommandBar** object that represents the menu displayed by the specified <u>pop-up control</u>. Read-only.

This example sets the variable fourthLevel to the fourth control on the command bar named "Drawing."

```
Set fourthLevel = CommandBars("Drawing") _
.Controls(1).CommandBar.Controls(4)
```

CommandBars Property

Returns a **<u>CommandBars</u>** collection.

expression.CommandBars

-

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

The following example return the **CommandBars** collection from the **MsoEnvelope** object in Microsoft Word.

Dim cbars As CommandBars
Set cbars = Application.ActiveDocument.MailEnvelope.Commandbars

CompareTo Property

Returns or sets a **String** that represents the text to compare in the query filter criterion. Read/write.

expression.CompareTo

-

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

The following example changes an existing filter to remove from the mail merge all records that do not have a Region field equal to "WA".

```
Sub SetQueryCriterion()
    Dim appOffice As Office.OfficeDataSourceObject
    Dim intItem As Integer
    Set appOffice = Application.OfficeDataSourceObject
    appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName
        "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
   With appOffice.Filters
        For intItem = 1 To .Count
            With .Item(intItem)
                If .Column = "Region" Then
                    .Comparison = msoFilterComparisonNotEqual
                    .CompareTo = "WA"
                    If .Conjunction = "Or" Then .Conjunction = "And"
                End If
            End With
        Next intItem
    End With
End Sub
```

- <u>Show All</u>

Comparison Property

Returns or sets an **MsoFilterComparison** constant that represents how to compare the **Column** and **CompareTo** properties. Read/write.

MsoFilterComparison can be one of these MsoFilterComparison constants.

msoFilterComparisonContains msoFilterComparisonEqual msoFilterComparisonGreaterThan msoFilterComparisonGreaterThanEqual msoFilterComparisonIsBlank msoFilterComparisonIsNotBlank msoFilterComparisonLessThan msoFilterComparisonLessThanEqual msoFilterComparisonNotContains msoFilterComparisonNotEqual

expression.Comparison

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

The following example changes an existing filter to remove from the mail merge all records that do not have a Region field equal to "WA".

```
Sub SetQueryCriterion()
    Dim appOffice As Office.OfficeDataSourceObject
    Dim intItem As Integer
    Set appOffice = Application.OfficeDataSourceObject
    appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName
        "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
   With appOffice.Filters
        For intItem = 1 To .Count
            With .Item(intItem)
                If .Column = "Region" Then
                    .Comparison = msoFilterComparisonNotEqual
                    .CompareTo = "WA"
                    If .Conjunction = "Or" Then .Conjunction = "And"
                End If
            End With
        Next intItem
    End With
End Sub
```

Condition Property

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

Returns the condition of the specified search criteria. Read-only MsoCondition.

MsoCondition can be one of these MsoCondition constants.

msoConditionAnyNumberBetween **msoConditionAnytime msoConditionAnytimeBetween msoConditionAtLeast msoConditionAtMost msoConditionBeginsWith** msoConditionDoesNotEqual **msoConditionEndsWith msoConditionEquals** msoConditionFileTypeAllFiles **msoConditionFileTypeBinders** msoConditionFileTypeCalendarItem msoConditionFileTypeContactItem msoConditionFileTypeDatabases msoConditionFileTypeDataConnectionFiles msoConditionFileTypeDesignerFiles msoConditionFileTypeEPaperFiles msoConditionFileTypeExcelWorkbooks msoConditionFileTypeJournalItem msoConditionFileTypeMailItem msoConditionFileTypeNoteItem msoConditionFileTypeOfficeFiles msoConditionFileTypeOutlookItems

msoConditionFileTypePhotoDrawFiles msoConditionFileTypePowerPointPresentations msoConditionFileTypeProjectFiles msoConditionFileTypePublisherFiles msoConditionFileTypeTaskItem msoConditionFileTypeTemplates msoConditionFileTypeVisioDocuments msoConditionFileTypeWebPages msoConditionFileTypeWordDocuments msoConditionFreeText msoConditionIncludes msoConditionIncludesFormsOf msoConditionIncludesNearEachOther msoConditionIncludesPhrase msoConditionInTheLast msoConditionInTheNext **msoConditionIsExactly** msoConditionIsNo msoConditionIsNot **msoConditionIsYes** msoConditionLastMonth msoConditionLastWeek msoConditionLessThan msoConditionMoreThan msoConditionNextMonth **msoConditionNextWeek** msoConditionOn **msoConditionOnOrAfter** msoConditionOnOrBefore **msoConditionThisMonth msoConditionThisWeek msoConditionToday msoConditionTomorrow**

msoConditionYesterday

This example returns the condition value for search criteria for the first property test.

```
With Application.FileSearch.PropertyTests(1)
MsgBox "The condition you've set is: " & .Condition
End With
```

- <u>Show All</u>

Conjunction Property

Returns or sets an **MsoFilterConjunction** constant that represents how a filter criterion relates to other filter criteria in the **ODSOFilters** object. Read/write.

MsoFilterConjunction can be one of these MsoFilterConjunction constants. msoFilterConjunctionAnd msoFilterConjunctionOr

expression.Conjunction

-

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

The following example changes an existing filter to remove from the mail merge all records that do not have a Region field equal to "WA".

```
Sub SetQueryCriterion()
    Dim appOffice As Office.OfficeDataSourceObject
    Dim intItem As Integer
    Set appOffice = Application.OfficeDataSourceObject
    appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName
        "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
   With appOffice.Filters
        For intItem = 1 To .Count
            With .Item(intItem)
                If .Column = "Region" Then
                    .Comparison = msoFilterComparisonNotEqual
                    .CompareTo = "WA"
                    If .Conjunction = "Or" Then .Conjunction = "And"
                End If
            End With
        Next intItem
    End With
End Sub
```

Connect Property

-

Returns or sets the state of the connection for the specified **COMAddIn** object. Read/write **Boolean**.

Remarks

The **Connect** property returns **True** if the add-in is active; it returns **False** if the add-in is inactive. An active add-in is registered and connected; an inactive add-in is registered but not currently connected.

The following example displays a message box that indicates whether COM add-in one is registered and currently connected.

```
If Application.COMAddIns(1).Connect Then
    MsgBox "The add-in is connected."
Else
    MsgBox "The add-in is not connected."
End If
```

Connector Property

-

Returns the connector between two similar property test values. The default value is **msoConnectorAnd**. Read-only <u>MsoConnector</u>.

MsoConnector can be one of these MsoConnector constants. msoConnectorAnd msoConnectorOr

Remarks

A connector specifies whether two similar search criteria will be combined to form one property test (as with **msoConnectorAnd**) or treated independently (as with **msoConnectorOr**).

This example displays a message that describes how the search criteria will be evaluated in a file search.

```
With Application.FileSearch.PropertyTests(1)
If .Connector = msoConnectorAnd Then
    MsgBox "All search criteria will be combined."
Else
    MsgBox "Criteria will be treated independently"
End If
End With
```

ConnectString Property

Returns or sets a **String** that represents the connection to the specified mail merge data source. Read/write.

expression.ConnectString

-

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

This example checks if the connection string contains the characters ODSOOutlook and displays a message accordingly.

```
Sub VerifyCorrectDataSource()
   Dim appOffice As OfficeDataSourceObject
   Set appOffice = Application.OfficeDataSourceObject
   appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName
      "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
   If InStr(appOffice.ConnectString, "ODSOOutlook") > 0 Then
      MsgBox "Your Outlook address book is used as the data source
   Else
      MsgBox "Your Outlook address book is not used as the data sc
End If
```

End Sub

- <u>Show All</u>

Context Property

-

Returns or sets a string that determines where a <u>command bar</u> will be saved. The string is defined and interpreted by the application. Read/write **String**.

Remarks

You can set the **Context** property only for custom command bars. This property will fail if the application doesn't recognize the context string, or if the application doesn't support changing context strings programmatically.

This example displays a message box containing the context string for the command bar named "Custom". This example works in Microsoft Word and other applications that support the **Context** property.

```
Set myBar = CommandBars _
.Add(Name:="Custom", Position:=msoBarTop, _
Temporary:=True)
With myBar
.Controls.Add Type:=msoControlButton, ID:=2
.Visible = True
End With
MsgBox (myBar.Context)
```

Controls Property

-

Returns a **<u>CommandBarControls</u>** object that represents all the controls on a command bar or pop-up control. Read-only.

For information about returning a single member of a collection, see **<u>Returning</u>** <u>**an Object from a Collection**</u>.

This example adds a combo box control to the command bar named "Custom" and fills the list with two items. The example also sets the number of line items, the width of the combo box, and an empty default for the combo box.

```
Set myControl = CommandBars("Custom").Controls _
.Add(Type:=msoControlComboBox, Before:=1)
With myControl
.AddItem Text:="First Item", Index:=1
.AddItem Text:="Second Item", Index:=2
.DropDownLines = 3
.DropDownWidth = 75
.ListHeaderCount = 0
End With
```

Count Property

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

Returns or sets a **Long** indicating the number of items in the specified collection. Read/write **Long** for the **BalloonCheckboxes** and **BalloonLabels** objects; readonly **Long** for all other objects in the Applies To list.

expression.Count

_

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

Remarks

For the **CommandBars** collection, the count includes only menu bars, toolbars, and shortcut menus.

For the **Scripts** collection, the count returned is the number of script blocks in the specified document. In Microsoft Word, Scripts.Count returns the total number of inline and floating script anchors combined.
This example uses the **Count** property to display the number of command bars in the **CommandBars** collection.

This example uses the **Count** property to display the number of check boxes in the Office Assistant balloon.

```
With Assistant.NewBalloon
    .CheckBoxes(1).Text = "First Choice"
    .CheckBoxes(2).Text = "Second Choice"
    .Text = "You have the following " _
    & .CheckBoxes.Count & " choices."
    .Show
End With
```

This example displays the number of custom document properties in the active document.

```
MsgBox ("There are " & _
ActiveDocument.CustomDocumentProperties.Count & _
" custom document properties in the " & _
"active document.")
```

Creator Property

Returns the four-character code for the application in which the specified object was created. Macintosh only. Read-only **Long**.

expression.Creator

-

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

DataSource Property

Returns or sets a **String** that represents the name of the attached data source. Read/write.

expression. DataSource

-

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

The following example sets the name of the data source if the name is blank.

```
Sub SetAndReturnDataSourceName()
    Dim appOffice As OfficeDataSourceObject
    Set appOffice = Application.OfficeDataSourceObject
    With appOffice
    .Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName;" &
        "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employee
        If .DataSource = "" Then
        .DataSource = "Northwind"
        MsgBox .DataSource
        End If
    End With
End Sub
```

Description Property

Description property as it applies to the COMAddIn object.

Returns or sets a descriptive **String** value for the specified **COMAddIn** object. Read/write.

expression.**Description**

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

• Description property as it applies to the **FileDialogFilter** object.

Returns the description of each **Filter** object as a **String** value. The description is the text that is displayed in the file dialog box. Read-only.

expression. Description

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

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

The following example displays the description text of the Microsoft Accessibility COM add-in for drawing.

```
MsgBox "The description of this " & _
    "COMAddIn is """ & Application.COMAddIns. _
    Item("msodraa9.ShapeSelect"). _
    Description & """
```

• As it applies to the **FileDialogFilter** object.

The following example iterates through the default filters of the SaveAs dialog box and displays the description of each filter that includes a Microsoft Excel file. The **Extensions** property is used to find the appropriate filter objects.

```
Sub Main()
    'Declare a variable as a FileDialogFilters collection.
    Dim fdfs As FileDialogFilters
    'Declare a variable as a FileDialogFilter object.
    Dim fdf As FileDialogFilter
    'Set the FileDialogFilters collection variable to
    'the FileDialogFilters collection of the SaveAs dialog box.
    Set fdfs = Application.FileDialog(msoFileDialogSaveAs).Filters
    'Iterate through the description and extensions of each
    'default filter in the SaveAs dialog box.
    For Each fdf In fdfs
        'Display the description of filters that include
        'Microsoft Excel files.
        If InStr(1, fdf.Extensions, "xls", vbTextCompare) > 0 Then
            MsgBox "Filter description: " & fdf. Description
        End If
    Next fdf
```

End Sub

DescriptionText Property

-

Returns or sets the description for a <u>command bar control</u>. The description is not displayed to the user, but it can be useful for documenting the behavior of the control for other developers. Read/write **String**.

Remarks

This property is used for Balloon Help on the Macintosh.

This example adds a control to a custom command bar, including a description of the control's behavior.

```
Set myBar = CommandBars.Add("Custom", msoBarTop, , True)
myBar.Visible = True
Set myControl = myBar.Controls _
    .Add(Type:=msoControlButton, ID:= _
    CommandBars("Standard").Controls("Paste").ID)
With myControl
    .DescriptionText = "Pastes the contents of the Clipboard"
    .Caption = "Paste"
End With
```

DialogType Property

Returns an **MsoFileDialogType** constant representing the type of file dialog box that the **FileDialog** object is set to display. Read-only.

MsoFileDialogType can be one of these MsoFileDialogType constants. msoFileDialogFilePicker msoFileDialogFolderPicker msoFileDialogOpen msoFileDialogSaveAs

expression.DialogType

_

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

The following example takes a **FileDialog** object of an unknown type and runs the **Execute** method if it is a SaveAs dialog box or an Open dialog box.

Sub DisplayAndExecuteFileDialog(ByRef fd As FileDialog)

```
'Use a With...End With block to reference the FileDialog object.
With fd
'If the user presses the action button...
If .Show = -1 Then
'Use the DialogType property to determine whether to
'use the Execute method.
Select Case .DialogType
Case msoFileDialogOpen, msoFileDialogSaveAs: .Execut
'Do nothing otherwise.
Case Else
End Select
'If the user presses Cancel...
Else
End If
End With
```

End Sub

DisableAskAQuestionDropdown Property

True if the Answer Wizard dropdown menu is enabled. Read/write **Boolean**.

expression.DisableAskAQuestionDropdown

-

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

The following example toggles the **DisableAskAQuestionDropdown** property.

```
Sub ToggleQuestionDropdown()
With Application.CommandBars
If .DisableAskAQuestionDropdown = True Then
.DisableAskAQuestionDropdown = False
Else
.DisableAskAQuestionDropdown = True
End If
End With
End Sub
```

DisableCustomize Property

True if toolbar customization is disabled. Read/write **Boolean**.

expression. DisableCustomize

-

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

The following example toggles the **DisableCustomize** property.

```
Sub ToggleCustomize()
With Application.CommandBars
If .DisableCustomize = True Then
.DisableCustomize = False
Else
.DisableCustomize = True
End If
End With
End Sub
```

DisplayFonts Property

-

True if the font names in the **Font** box are displayed in their actual fonts. Read/write **Boolean**.

This example sets three options for all command bars in Microsoft Office, including custom command bars and the controls on them.

```
With CommandBars
.LargeButtons = True
.DisplayFonts = True
.AdaptiveMenus = True
End With
```

DisplayKeysInTooltips Property

-

True if shortcut keys are displayed in the ToolTips for each <u>command bar</u> <u>control</u>. Read/write **Boolean**.

Remarks

To display shortcut keys in ToolTips, you must also set the **<u>DisplayTooltips</u>** property to **True**.

This example sets options for all command bars in Microsoft Office.

```
With CommandBars
    .LargeButtons = True
    .DisplayTooltips = True
    .DisplayKeysInTooltips = True
    .MenuAnimationStyle = msoMenuAnimationUnfold
End With
```

DisplayTooltips Property

-

True if ScreenTips are displayed whenever the user positions the pointer over <u>command bar controls</u>. Read/write **Boolean**.

Remarks

Setting the **DisplayTooltips** property in a container application immediately affects every command bar in every running Microsoft Office application, and in every Office application opened after the property is set.

This example displays large controls and ToolTips on all command bars.

Set allBars = CommandBars

allBars.LargeButtons = True
allBars.DisplayTooltips = True

DropDownLines Property

-

Returns or sets the number of lines in a command bar <u>combo box control</u>. The combo box control must be a custom control and it must be a drop-down list box or a combo box. Read/write **Long**.

Note An error occurs if you attempt to set this property for a combo box control that's an edit box or a built-in combo box control.

Remarks

If this property is set to 0 (zero), the number of lines in the control will be based on the number of items in the list.

This example adds a combo box control containing two items to the command bar named "Custom". The example also sets the number of line items, the width of the combo box, and an empty default for the combo box.

```
Set myBar = CommandBars("Custom")
Set myControl = myBar.Controls.Add(Type:=msoControlComboBox, Id:=1)
With myControl
    .AddItem Text:="First Item", Index:=1
    .AddItem "Second Item", 2
    .DropDownLines = 3
    .DropDownWidth = 75
    .ListHeaderCount = 0
End With
```

DropDownWidth Property

-

Returns or sets the width (in pixels) of the list for the specified command bar <u>combo box control</u>. Read/write **Long**.

Note An error occurs if you attempt to set this property for a built-in control.

Remarks

If this property is set to -1, the width of the list is based on the length of the longest item in the combo box list. If this property is set to 0, the width of the list is based on the width of the control.
This example adds a combo box control containing two items to the command bar named "Custom". The example also sets the number of line items, the width of the combo box, and an empty default for the combo box.

```
Set myBar = CommandBars("Custom")
Set myControl = myBar.Controls.Add(Type:=msoControlComboBox, Id:=1)
With myControl
    .AddItem "First Item", 1
    .AddItem "Second Item", 2
    .DropDownLines = 3
    .DropDownWidth = 75
    .ListHeaderCount = 0
End With
```

- <u>Show All</u>

Enabled Property

-

True if the specified <u>command bar</u> or <u>command bar control</u> is enabled. Read/write **Boolean**.

Remarks

For command bars, setting this property to **True** causes the name of the command bar to appear in the list of available command bars.

For built-in controls, if you set the **Enabled** property to **True**, the application determines its state, but setting it to **False** will force it to be disabled.

This example adjusts the command bars according to the user level specified by user. If user is "Level 1," the command bar named "VB Custom Bar" is displayed. If user is any other value, the built-in Visual Basic command bar is reset to its default state and the command bar named "VB Custom Bar" is disabled.

```
Set myBar = CommandBars _
    .Add(Name:="VB Custom Bar", Position:=msoBarTop, _
    Temporary:=True)
With myBar
    .Controls.Add Type:=msoControlButton, ID:=2
    .Visible = True
End With
If user = "Level 1" Then
    myBar.Visible = True
Else
    CommandBars("Visual Basic").Reset
    myBar.Enabled = False
End If
```

This example adds two command bar buttons to the command bar named "Custom". The first control is disabled; the second control is enabled by default.

```
Set myBar = CommandBars("Custom")
With myBar
    .Controls.Add Type:=msoControlButton, Id:=3
    .Controls(1).Enabled = False
    .Controls.Add Type:=msoControlButton, Id:=3
End With
myBar.Visible = True
```

ExpireDate Property

Returns a **Variant** representing the date on which the digital signature that corresponds to the **Signature** object will expire. Read-only.

expression. ExpireDate

-

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

The following example prompts the user to select a digital signature with which to sign the active document in Microsoft Word. To use this example, open a document in Word and call this function. The function will test to make sure that the digital signature that the user selects will not expire in less than 12 months. If it will expire, the certificate isn't attached.

```
Function AddSignature() As Boolean
    On Error GoTo Error Handler
    Dim sig As Signature
    'Display the dialog box that lets the
    'user select a digital signature.
    'If the user selects a signature, then
    'it is added to the Signatures
    'collection. If the user doesn't, then
    'an error is returned.
    Set sig = ActiveDocument.Signatures.Add
    'Test to make sure that the new Signature object
    'doesn't expire too soon. This expression calculates
    'the number of months until the Signature object expires.
    If DateDiff("m", sig.SignDate, sig.ExpireDate) < 12 Then</pre>
        MsgBox "This Certificate will expire in less than 1 year." &
        "Please use a newer certificate."
        AddSignature = False
        sig.Delete
    Else
        AddSignature = True
    End If
    'Commit all signatures in the SignatureSet collection to the dis
    ActiveDocument.Signatures.Commit
    Exit Function
Error Handler:
    AddSignature = False
    MsgBox "Action cancelled."
```

```
End Function
```

Extended Property

-

Sets or returns attributes added to the <SCRIPT> tag, with the exception of the LANGUAGE and ID attributes. Read/write **String**.

Remarks

Attributes are separated by spaces, the same as in HTML. You cannot pass the LANGUAGE attribute or the ID attribute by using the **Extended** property.

The Microsoft Office host application doesn't provide any means of checking the syntax of passed attributes.

If you pass the LANGUAGE attribute in the **Extended** property, the <SCRIPT> tag receives two language settings, which causes a conflict.

If you pass an ID attribute in the **Extended** property and no ID has been set through either the ID parameter of the **Add** method or the **Id** property of the **Script** object, the ID is exported correctly.

This example checks the **Extended** property to ensure that no additional attributes have been added to the first script in worksheet one in the active workbook.

Extensions Property

Returns a **String** value containing the extensions that determine which files are displayed in a file dialog box for each **Filter** object. Read-only.

expression. Extensions

-

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

The following example iterates through the default filters of the SaveAs dialog box and displays the description of each filter that includes a Microsoft Excel file. The **Extensions** property is used to find the appropriate filter objects.

```
Sub Main()
    'Declare a variable as a FileDialogFilters collection.
    Dim fdfs As FileDialogFilters
    'Declare a variable as a FileDialogFilter object.
    Dim fdf As FileDialogFilter
    'Set the FileDialogFilters collection variable to
    'the FileDialogFilters collection of the SaveAs dialog box.
    Set fdfs = Application.FileDialog(msoFileDialogSaveAs).Filters
    'Iterate through the description and extensions of each
    'default filter in the SaveAs dialog box.
    For Each fdf In fdfs
        'Display the description of filters that include
        'Microsoft Excel files.
        If InStr(1, fdf.Extensions, "xls", vbTextCompare) > 0 Then
            MsgBox "Description of filter: " & fdf.Description
        End If
    Next fdf
End Sub
```

- <u>Show All</u>

FaceId Property

-

Returns or sets the Id number for the face of a command bar <u>button control</u>. Read/write **Long**.

Remarks

The **FaceId** property dictates the look, not the function, of a command bar button. The **Id** property of the **CommandBarControl** object determines the function of the button.

The value of the **FaceId** property for a command bar button with a custom face is 0 (zero).

This example adds a command bar button to a custom command bar. Clicking this button is equivalent to clicking the **Open** command on the **File** menu because the ID number is 23, yet the button has the same button face as the built-in **Charting** button.

```
Set newBar = CommandBars.Add(Name:="Custom2", _
    Position:=msoBarTop, Temporary:=True)
newBar.Visible = True
Set con = newBar.Controls.Add(Type:=msoControlButton, Id:=23)
con.FaceId = 17
```

FeatureTips Property

-

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

True if the Office Assistant provides information about using application features more effectively. Read/write **Boolean**.

Remarks

The **FeatureTips** property corresponds to the **Using features more effectively** check box on the **Options** tab in the **Assistant** dialog box.

This example allows the Office Assistant to provide information about using application features more effectively.

Assistant.**FeatureTips** = True

FileName Property

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

Assistant object: Returns or sets the path and file name for the active Office Assistant. Read/write **String**.

FileSearch object: Returns or sets the name of the file to look for during a file search. The name of the file may include the * (asterisk) or ? (question mark) wildcards. Use the question mark wildcard to match any single character. For example, type **gr?y** to match both "gray" and "grey." Use the asterisk wildcard to match any number of characters. For example, type ***.txt** to find all files that have the .TXT extension. Read/write **String**.

expression.FileName

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

This example searches for all files located in the My Documents folder that begin with "cmd" and have a file name extension. The example displays the name and location of each found file.

```
Set fs = Application.FileSearch
With fs
.LookIn = "C:\My Documents"
.FileName = "cmd*.*"
If .Execute > 0 Then
    MsgBox "There were " & .FoundFiles.Count & _
        " file(s) found."
    For i = 1 To .FoundFiles.Count
        MsgBox .FoundFiles(i)
        Next i
    Else
        MsgBox "There were no files found."
    End If
End With
```

Files Property

-

Returns an **AnswerWizardFiles** collection that represents the list of files available to the current AnswerWizard. Read-only.

For information about returning a single member of a collection, see <u>Returning</u> <u>an Object from a Collection</u>.

This example resets the file list for the current AnswerWizard and then displays both the file count and the file names in a message box.

```
Dim customAnswerWizard As AnswerWizard
Dim strFileList As String
Dim intCounter As Integer
Dim intNumFiles As Integer
Set customAnswerWizard = Application.AnswerWizard
intCounter = 1
customAnswerWizard.ResetFileList
strFileList = ""
intNumFiles = customAnswerWizard.Files.Count
For intCounter = 1 To (intNumFiles)
    strFileList = strFileList & _
    customAnswerWizard.Files.Item(intCounter) & Chr(13)
Next
MsgBox "There are " & customAnswerWizard.Files.Count & _
    " files avaialble through this AnswerWizard: " & _
    Chr(13) & strFileList
```

- <u>Show All</u>

FileType Property

Returns or sets the type of file to look for during a file search. Read/write **MsoFileType**.

MsoFileType can be one of these MsoFileType constants.

msoFileTypeAllFiles msoFileTypeBinders msoFileTypeCalendarItem msoFileTypeContactItem msoFileTypeCustom **msoFileTypeDatabases** msoFileTypeDataConnectionFiles **msoFileTypeDesignerFiles** msoFileTypeDocumentImagingFiles msoFileTypeExcelWorkbooks msoFileTypeJournalItem **msoFileTypeMailItem** msoFileTypeNoteItem **msoFileTypeOfficeFiles** msoFileTypeOutlookItems msoFileTypePhotoDrawFiles msoFileTypePowerPointPresentations **msoFileTypeProjectFiles msoFileTypePublisherFiles msoFileTypeTaskItem msoFileTypeTemplates** msoFileTypeVisioFiles **msoFileTypeWebPages msoFileTypeWordDocuments**

expression.FileType

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

Remarks

The constant **msoFileTypeOfficeFiles** includes all files with any of the following extensions: *.doc, *.xls, *.ppt, *.pps, *.obd, *.mdb, *.mpd, *.dot, *.xlt, *.pot, *.obt, *.htm, or *.html.

This example searches for all Binder files located in the My Documents folder. The example displays a message box that contains the name and location of each file that's found.

FileTypes Property

Returns a **<u>FileTypes</u>** collection.

expression.FileTypes

-

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

The following example searches for all HTML and Microsoft Excel files on the C:\ drive.

```
Sub SearchForFiles()
    'Declare a variable to act as a generic counter.
    Dim lngCount As Long
    'Use a With...End With block to reference the
    'FileSearch object.
   With Application.FileSearch
        'Clear all the parameters of the previous searches.
        'This method doesn't clear the LookIn property or
        'the SearchFolders collection.
        .NewSearch
        'Setting the FileType property clears the
        'FileTypes collection and sets the first
        'item in the collection to the file type
        'defined by the FileType property.
        .FileType = msoFileTypeWebPages
        'Add a second item to the FileTypes collection.
        .FileTypes.Add msoFileTypeExcelWorkbooks
        'Display the number of FileTypes objects in the collection.
        MsgBox "You are about to search for " & .FileTypes.Count & _
            " file types."
        'Set up the search to look in all subfolders on the C:\ driv
        LookIn = "C: \
        .SearchSubFolders = True
        'Execute the search and test to see if any files
        'were found.
        If .Execute <> 0 Then
            'Display the number of files found.
            MsgBox "Files found: " & .FoundFiles.Count
            'Loop through the list of found files and
            'display the path of each one in a message box.
            For lngCount = 1 To .FoundFiles.Count
```

FilterIndex Property

Returns or sets a **Long** indicating the default file filter of a file dialog box. The default filter determines which types of files are displayed when the file dialog box is first opened. Read/write.

expression.FilterIndex

-

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

Remarks

If you try to set this property to a number greater than the number of filters, the last available filter will be selected.

The following example displays a File Picker dialog box using the **FileDialog** object and displays each selected file in a message box. This example also demonstrates how to add a new file filter and how to make it the default filter.

```
Sub Main()
    'Declare a variable as a FileDialog object.
    Dim fd As FileDialog
    'Create a FileDialog object as a File Picker dialog box.
    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    'Declare a variable to contain the path
    'of each selected item. Even though the path is a String,
    'the variable must be a Variant because For Each...Next
    'routines only work with Variants and Objects.
    Dim vrtSelectedItem As Variant
    'Use a With...End With block to reference the FileDialog object.
   With fd
        'Add a filter that includes GIF and JPEG images and make it
        .Filters.Add "Images", "*.gif; *.jpg; *.jpeg", 2
        'Sets the initial file filter to number 2.
        .FilterIndex = 2
        'Use the Show method to display the File Picker dialog box a
        'If the user presses the action button...
        If .Show = -1 Then
            'Step through each string in the FileDialogSelectedItems
            For Each vrtSelectedItem In .SelectedItems
                'vrtSelectedItem is a String that contains the path
                'You can use any file I/O functions that you want to
                'This example simply displays the path in a message
                MsgBox "Selected item's path: " & vrtSelectedItem
            Next vrtSelectedItem
        'If the user presses Cancel...
        Else
        End If
    End With
```

```
'Set the object variable to Nothing.
Set fd = Nothing
```

End Sub
Filters Property

Returns a **<u>FileDialogFilters</u>** collection.

expression.Filters

-

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

The following example displays a File Picker dialog box using the **FileDialog** object and displays each selected file in a message box. The example also adds a new file filter called "Images."

```
Sub Main()
    'Declare a variable as a FileDialog object.
    Dim fd As FileDialog
    'Create a FileDialog object as a File Picker dialog.
    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    'Declare a variable to contain the path
    'of each selected item. Even though the path is a String,
    'the variable must be a Variant because For Each...Next
    'routines only work with Variants and Objects.
    Dim vrtSelectedItem As Variant
    'Use a With...End With block to reference the FileDialog object.
   With fd
        'Add a filter that includes GIF and JPEG images and make it
        .Filters.Add "Images", "*.gif; *.jpg; *.jpeg", 1
        'Use the Show method to display the File Picker dialog box a
        'If the user presses the action button...
        If .Show = -1 Then
            'Step through each string in the FileDialogSelectedItems
            For Each vrtSelectedItem In .SelectedItems
                'vrtSelectedItem is a String that contains the path
                'You can use any file I/O functions that you want to
                'This example simply displays the path in a message
                MsgBox "Selected item's path: " & vrtSelectedItem
            Next vrtSelectedItem
        'If the user presses Cancel...
        Else
        End If
    End With
    'Set the object variable to Nothing.
```

```
Set fd = Nothing
```

End Sub

FixedWidthFont Property

-

Sets or returns the fixed-width font setting in the host application. Read/write **String**.

Remarks

When you set the **FixedWidthFont** property, the host application does not check the value for validity.

This example sets the fixed-width font and fixed-width font size for the English/Western European/Other Latin Script character set in the active application.

```
Application.DefaultWebOptions. _
Fonts(msoCharacterSetEnglishWesternEuropeanOtherLatinScript) _
.FixedWidthFont = "System"
Application.DefaultWebOptions. _
Fonts(msoCharacterSetEnglishWesternEuropeanOtherLatinScript) _
.FixedWidthFontSize = 12
```

FixedWidthFontSize Property

-

Sets or returns the fixed-width font size setting in the host application, in points. Read/write **Single**.

Remarks

When you set the **FixedWidthFontSize** property, the host application does not check the value for validity. If you enter an invalid value, such as a nonnumber, the host application sets the size to 0 points. You can enter half-point sizes; if you enter other fractional point sizes, they are rounded up or down to the nearest half-point.

This example sets the fixed-width font and fixed-width font size for the English/Western European/Other Latin Script character set in the active application.

```
Application.DefaultWebOptions. _
Fonts(msoCharacterSetEnglishWesternEuropeanOtherLatinScript) _
.FixedWidthFont = "System"
Application.DefaultWebOptions. _
Fonts(msoCharacterSetEnglishWesternEuropeanOtherLatinScript) _
.FixedWidthFontSize = 12
```

FoundFiles Property

-

Returns a **FoundFiles** object that contains the names of all the files found during a search. Read-only.

This example steps through the list of files found during a search and displays the path for each file.

With Application.FileSearch
For i = 1 To .FoundFiles.Count
 MsgBox .FoundFiles(i)
Next I
End With

GuessHelp Property

-

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

True if the Office Assistant balloon presents a list of Help topics based on keywords the user selects before clicking the Assistant window or pressing F1. Read/write **Boolean**.

Remarks

The **GuessHelp** property corresponds to the **Guess help topics** option under **Use the Office Assistant** on the **Options** tab in the **Office Assistant** dialog box.

This example allows the Office Assistant to guess at Help topics.

Assistant.**GuessHelp** = True

Guid Property

-

Returns the globally unique class identifier (GUID) for the specified **COMAddIn** object. Read-only **String**.

The following example displays the ProgID and GUID for COM add-in one in a message box.

```
MsgBox "My ProgID is " & _
Application.COMAddIns(1).ProgID & _
" and my GUID is " & _
Application.COMAddIns(1).Guid
```

Heading Property

-

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

Returns or sets the heading that appears in the Office Assistant balloon. Read/write **String**.

Remarks

You can specify a graphic to be displayed in the balloon heading by using the following syntax: {*type location sizing_factor*}, where *type* is bmp (bitmap) or wmf (Windows metafile), *location* is the resource ID or the path and file name, and *sizing_factor* specifies the width of the wmf (omitted for bmp).

The balloon heading also supports underlined text and text that has one of the 16 system palette colors applied to it. To display underlined text in a heading, use the syntax {*ul*} or {*ul* 1}; use {*ul* 0} to turn underlining off. To change the color of heading text, precede the text string with the character sequence {*cf number*}, where *number* is one of the system color numbers listed in the following table.

System color number	Color
0	Black
1	Dark red
2	Dark green
3	Dark yellow
4	Dark blue
5	Dark magenta
6	Dark cyan
7	Light gray
248	Medium gray
249	Red
250	Green
251	Yellow
252	Blue
253	Magenta
254	Cyan
255	White

If you specify a number other than one of the preceding system color numbers, the heading text is black.

This example displays a balloon with a heading, text, and three region choices.

```
With Assistant.NewBalloon
    .Button = msoButtonSetOkCancel
    .Heading = "Regional Sales Data"
    .Text = "Select a region"
    For i = 1 To 3
        .CheckBoxes(i).Text = "Region " & i
        Next
        .Show
End With
```

- <u>Show All</u>

Height Property

Returns or sets the height of a <u>command bar control</u> or <u>command bar</u>. Read/write **Long**.

expression.Height

-

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

This example adds a custom control to the command bar named Custom. The example sets the height of the custom control to twice the height of the command bar and sets the control's width to 50 pixels. Notice how the command bar automatically resizes itself to accommodate the control.

```
Set myBar = CommandBars("Custom")
barHeight = myBar.Height
Set myControl = myBar.Controls _
    .Add(Type:=msoControlButton, _
    Id:= CommandBars("Standard").Controls("Save").Id, _
    Temporary:=True)
With myControl
    .Height = barHeight * 2
    .Width = 50
End With
myBar.Visible = True
```

- <u>Show All</u>

HelpContextId Property

-

Returns or sets the Help context Id number for the Help topic attached to the <u>command bar control</u>. Read/write **Long**.

Remarks

To use this property, you must also set the **<u>HelpFile</u>** property. Help topics respond to Shift+F1.

This example adds a custom command bar with a combo box that tracks stock data. The example also specifies the Help topic to be displayed for the combo box when the user presses SHIFT+F1.

```
Set myBar = CommandBars _
    .Add(Name:="Custom", Position:=msoBarTop, _
    Temporary:=True)
With myBar
    .Controls.Add Type:=msoControlComboBox, ID:=1
    .Visible = True
End With
With CommandBars("Custom").Controls(1)
    .AddItem "Get Stock Quote", 1
    .AddItem "View Chart", 2
    .AddItem "View Fundamentals", 3
    .AddItem "View News", 4
    .Caption = "Stock Data"
    .DescriptionText = "View Data For Stock"
    .HelpFile = "C:\corphelp\custom.hlp"
    .HelpContextID = 47
End With
```

- <u>Show All</u>

HelpFile Property

-

Returns or sets the file name for the Help topic attached to the <u>command bar</u> <u>control</u>. Read/write **String**.

Remarks

To use this property, you must also set the <u>HelpContextID</u> property. Help topics respond to the user pressing SHIFT+F1.

This example adds a custom command bar with a combo box that tracks stock data. The example also specifies the Help topic to be displayed for the combo box when the user presses SHIFT+F1.

```
Set myBar = CommandBars _
    .Add(Name:="Custom", Position:=msoBarTop, _
    Temporary:=True)
With myBar
    .Controls.Add Type:=msoControlComboBox, ID:=1
    .Visible = True
End With
With CommandBars("Custom").Controls(1)
    .AddItem "Get Stock Quote", 1
    .AddItem "View Chart", 2
    .AddItem "View Fundamentals", 3
    .AddItem "View News", 4
    .Caption = "Stock Data"
    .DescriptionText = "View Data For Stock"
    .HelpFile = "C:\corphelp\custom.hlp"
    .HelpContextID = 47
End With
```

HighPriorityTips Property

-

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

True if the Office Assistant displays high-priority tips. Read/write **Boolean**.

Remarks

The **HighPriorityTips** property corresponds to the **Only show high priority tips** option under **Show tips about** on the **Options** tab in the **Office Assistant** dialog box.

This example sets the Office Assistant to display high-priority tips.

Assistant.**HighPriorityTips** = True

HTMLProjectItems Property

-

Returns the **HTMLProjectItems** collection that is included in the specified HTML project. Read-only **HTMLProjectItems**.

This example returns the number of items in the **HTMLProjectItems** collection in the HTML project in the active workbook.

intCount = ActiveWorkbook.HTMLProject.HTMLProjectItems.Count

HyperlinkType Property

-

Sets or returns the type of hyperlink associated with the specified command bar button. Read/write **MsoCommandBarButtonHyperlinkType**.

MsoCommandBarButtonHyperlinkType can be one of these MsoCommandBarButtonHyperlinkType constants. msoCommandBarButtonHyperlinkInsertPicture msoCommandBarButtonHyperlinkNone msoCommandBarButtonHyperlinkOpen
This example checks the **HyperlinkType** property for the specified command bar button on the command bar named "Custom.". If **HyperlinkType** is set to **msoCommandBarButtonHyperlinkNone**, the example sets the property to **msoCommandBarButtonHyperlinkOpen** and sets the URL to www.microsoft.com.

```
Set myBar = CommandBars _
    .Add(Name:="Custom", Position:=msoBarTop, _
    Temporary:=True)
Set myButton = myBar.Controls.Add(Type:=msoControlButton)
With myButton
    .FaceId = 277
    .HyperlinkType = msoCommandBarButtonHyperlinkNone
End With
If myButton.HyperlinkType <> _
    msoCommandBarButtonHyperlinkOpen Then
    myButton.HyperlinkType = _
        msoCommandBarButtonHyperlinkOpen
    myButton.TooltipText = "www.microsoft.com"
End If
```

Icon Property

_

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

Returns or sets the type of icon that appears in the upper-left portion of the Office Assistant balloon. Read/write **MsoIconType**.

MsoIconType can be one of these MsoIconType constants. msoIconAlert msoIconAlertCritical msoIconAlertInfo msoIconAlertQuery msoIconAlertWarning msoIconNone msoIconTip

This example creates a balloon with an "Alert" icon that instructs the user to select a printer.

```
With Assistant.NewBalloon
    .Heading = "Select A Printer"
    .Text = "You must select a printer before printing."
    .Icon = msoIconAlert
    .CheckBoxes(1).Text = "Local printer"
    .CheckBoxes(2).Text = "Network printer"
    .Show
End With
```

- <u>Show All</u>

Id Property

Id property as it applies to the CommandBarButton,
 CommandBarComboBox, and CommandBarControl objects.

Returns the ID for a built-in <u>command bar control</u>. Read-only **Long**.

expression.Id

-

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

Remarks

A control's ID determines the built-in action for that control. The value of the **Id** property for all custom controls is 1.

▶ <u>Id property as it applies to the **Script** object.</u>

Sets or returns the ID of a **Script** object. Read/write **String**.

expression.Id

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

Remarks

The ID returned is the ID attribute of the <SCRIPT> tag in HTML. If there's no ID attribute specified in the <SCRIPT> tag, the **Id** property returns an empty string.

Id specifies an SGML identifier used for naming elements. Valid identifiers include any string that begins with a letter and is composed of alphanumeric characters; the string can also include the underscore character (_).

The ID must be unique within the HTML document.

```
As it applies to the CommandBarButton, CommandBarComboBox, and
CommandBarControl objects.
```

This example changes the button face of the first control on the command bar named "Custom2" if the button's ID value is less than 25.

```
Set ctrl = CommandBars("Custom").Controls(1)
With ctrl
    If .Id < 25 Then
        .FaceId = 17
        .Tag = "Changed control"
    End If
End With</pre>
```

The following example changes the caption of every control on the toolbar named "Standard" to the current value of the **Id** property for that control.

```
For Each ctl In CommandBars("Standard").Controls
    ctl.Caption = CStr(ctl.Id)
Next ctl
```

• As it applies to the **Script** object.

This example sets the **Id** property of the first script in worksheet one of the active workbook to a new value.

```
ActiveWorkbook.Worksheets(1).Scripts(1).Id = "UpdatedScriptName"
```

Index Property

Returns a **Long** representing the index number for an object in the collection. Read-only.

expression.Index

-

Remarks

The position of the first command bar control is 1. Separators are not counted in the **CommandBarControls** collection.

This example searches the command bar named "Custom2" for a control with an Id value of 23. If such a control is found and the index number of the control is greater than 5, the control will be positioned as the first control on the command bar.

```
Set myBar = CommandBars("Custom2")
Set ctrl1 = myBar.FindControl(Id:=23)
If ctrl1.Index > 5 Then
        ctrl1.Move before:=1
End If
```

InitialFileName Property

Set or returns a **String** representing the path and/or file name that is initially displayed in a file dialog box. Read/write.

expression.InitialFileName

-

Remarks

You can use the '*' and '?' wildcard characters when specifying the file name but not when specifying the path. The '*' represents any number of consecutive characters and the '?' represents a single character. For example,

```
.InitialFileName = "c:\c*s.txt" will return both "charts.txt" and
"checkregister.txt."
```

If you specify a path and no file name, then all files that are allowed by the file filter will appear in the dialog box.

If you specify a file that exists in the initial folder, then only that file will appear in the dialog box.

If you specify a file name that doesn't exist in the initial folder, then the dialog box will contain no files. The type of file that you specify in the **InitialFileName** property will override the file filter settings.

If you specify an invalid path, the last-used path is used. A message will warn users when an invalid path is used.

Setting this property to a string longer than 256 characters will cause a run-time error.

The following example displays a File Picker dialog box using the **FileDialog** object and displays each selected file in a message box.

```
Sub Main()
    'Declare a variable as a FileDialog object
    Dim fd As FileDialog
    'Create a FileDialog object as a File Picker dialog box.
    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    'Declare a variable to contain the path
    'of each selected item. Even though the path is a String,
    'the variable must be a Variant because For Each...Next
    'routines only work with Variants and Objects.
    Dim vrtSelectedItem As Variant
    'Use a With...End With block to reference the FileDialog object.
   With fd
        'Set the initial path to the C:\ drive.
        .InitialFileName = "C:\"
        'Use the Show method to display the File Picker dialog box a
        'If the user presses the action button...
        If .Show = -1 Then
            'Step through each string in the FileDialogSelectedItems
            For Each vrtSelectedItem In .SelectedItems
                'vrtSelectedItem is a String that contains the path
                'You can use any file I/O functions that you want to
                'This example simply displays the path in a message
                MsgBox "Selected item's path: " & vrtSelectedItem
            Next vrtSelectedItem
        'If the user presses Cancel...
        Else
        End If
    End With
    'Set the object variable to Nothing.
    Set fd = Nothing
```

End Sub

- <u>Show All</u>

InitialView Property

Returns or sets an **MsoFileDialogView** constant representing the initial presentation of files and folders in a file dialog box. Read/write.

MsoFileDialogView can be one of these MsoFileDialogView constants.

msoFileDialogViewDetails

msoFileDialogViewLargeIcons

msoFileDialogViewList

-

msoFileDialogViewPreview

msoFileDialogViewProperties

msoFileDialogViewSmallIcons

msoFileDialogViewThumbnail This constant is only available in conjunction with Microsoft Windows 2000 or Microsoft Windows Millennium Edition, or later.

msoFileDialogViewWebView Not available. If you select this constant, the default view will be used.

expression.InitialView

The following example displays a File Picker dialog box in details view using the **FileDialog** object and displays each selected file in a message box.

```
Sub Main()
    'Declare a variable as a FileDialog object.
    Dim fd As FileDialog
    'Create a FileDialog object as a File Picker dialog box.
    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    'Declare a variable to contain the path
    'of each selected item. Even though the path is a String,
    'the variable must be a Variant because For Each...Next
    'routines only work with Variants and Objects.
    Dim vrtSelectedItem As Variant
    'Use a With...End With block to reference the FileDialog object.
   With fd
        'Set the initial view to the details view.
        .InitialView = msoFileDialogViewDetails
        'Use the Show method to display the File Picker dialog box a
        'If the user presses the action button...
        If .Show = -1 Then
            'Step through each string in the FileDialogSelectedItems
            For Each vrtSelectedItem In .SelectedItems
                'vrtSelectedItem is a String that contains the path
                'You can use any file I/O functions that you want to
                'This example simply displays the path in a message
                MsgBox "Selected item's path: " & vrtSelectedItem
            Next vrtSelectedItem
        'If the user presses Cancel...
        Else
        End If
    End With
    'Set the object variable to Nothing.
    Set fd = Nothing
```

End Sub

Introduction Property

Sets or returns the introductory text that is included with a document that is sent using the **MsoEnvelope** object. The introductory text is included at the top of the document in the e-mail. Read/write **String**.

expression.Introduction

-

The following example sends the active Microsoft Word document as an e-mail to the e-mail address that you pass to the subroutine.

```
Sub SendMail(ByVal strRecipient As String)
    'Use a With...End With block to reference the MsoEnvelope object
    With Application.ActiveDocument.MailEnvelope
        'Add some introductory text before the body of the e-mail.
        .Introduction = "Please read this and send me your comments.
        'Return a MailItem object that you can use to send the docum
        With .Item
            'All of the mail item settings are saved with the docume
            'When you add a recipient to the Recipients collection
            'or change other properties these settings will persist.
            .Recipients.Add strRecipient
            .Subject = "Here is the document."
            'The body of this message will be
            'the content of the active document.
            .Send
        End With
    End With
End Sub
```

IsCertificateExpired Property

True if the digital certificate that corresponds to the **Signature** object has expired. Read-only **Boolean**.

expression. Is Certificate Expired

-

The following example prompts the user to select a digital signature with which to sign the active document in Microsoft Word. To use this example, open a document in Word and pass this function the name of a certificate issuer and the name of a certificate signer that match the **Issued By** and **Issued To** fields of a digital certificate in the **Digital Certificates** dialog box. This example will test to make sure that the digital signature that the user selects meets certain criteria, such as not having expired, before the new signature is committed to the disk.

```
Function AddSignature(ByVal strIssuer As String,
    strSigner As String) As Boolean
    On Error GoTo Error_Handler
    Dim sig As Signature
    'Display the dialog box that lets the
    'user select a digital signature.
    'If the user selects a signature, then
    'it is added to the Signatures
    'collection. If the user doesn't, then
    'an error is returned.
    Set sig = ActiveDocument.Signatures.Add
    'Test several properties before commiting the Signature object t
    If sig.Issuer = strIssuer And
        sig.Signer = strSigner And _
        sig.IsCertificateExpired = False And _
        sig.IsCertificateRevoked = False And _
        siq.IsValid = True Then
        MsgBox "Signed"
        AddSignature = True
    'Otherwise, remove the Signature object from the SignatureSet co
    Else
        sig.Delete
        MsgBox "Not signed"
        AddSignature = False
    End If
    'Commit all signatures in the SignatureSet collection to the dis
    ActiveDocument.Signatures.Commit
```

Exit Function

```
Error_Handler:
AddSignature = False
MsgBox "Action cancelled."
End Function
```

IsCertificateRevoked Property

True if the digital certificate that corresponds to the **Signature** object has been revoked by the issuer of the certificate. Read-only **Boolean**.

expression. Is Certificate Revoked

_

The following example prompts the user to select a digital signature with which to sign the active document in Microsoft Word. To use this example, open a document in Word and pass this function the name of a certificate issuer and the name of a certificate signer that match the **Issued By** and **Issued To** fields of a digital certificate in the **Digital Certificates** dialog box. This example will test to make sure that the digital signature that the user selects meets certain criteria, such as not having expired, before the new signature is committed to the disk.

```
Function AddSignature(ByVal strIssuer As String,
    strSigner As String) As Boolean
    On Error GoTo Error_Handler
    Dim sig As Signature
    'Display the dialog box that lets the
    'user select a digital signature.
    'If the user selects a signature, then
    'it is added to the Signatures
    'collection. If the user doesn't, then
    'an error is returned.
    Set sig = ActiveDocument.Signatures.Add
    'Test several properties before commiting the Signature object t
    If sig.Issuer = strIssuer And
        sig.Signer = strSigner And _
        sig.IsCertificateExpired = False And _
        sig.IsCertificateRevoked = False And _
        siq.IsValid = True Then
        MsgBox "Signed"
        AddSignature = True
    'Otherwise, remove the Signature object from the SignatureSet co
    Else
        sig.Delete
        MsgBox "Not signed"
        AddSignature = False
    End If
    'Commit all signatures in the SignatureSet collection to the dis
    ActiveDocument.Signatures.Commit
```

Exit Function

```
Error_Handler:
AddSignature = False
MsgBox "Action cancelled."
End Function
```

IsOpen Property

-

True if the specified HTML project item is open in the Microsoft Script Editor. Read-only **Boolean**.

This example determines whether the specified HTML project item is open. and if it is, the example then loads script from the specified file.

```
If ActiveWorkbook.HTMLProject.HTMLProjectItems _
    .Item(1).IsOpen Then
    ActiveWorkbook.HTMLProject.HTMLProjectItems _
    .Item(1).LoadFromFile ("C:\MyScript.txt")
Else
    MsgBox "The HTMLProjectItem is not open."
End If
```

IsPriorityDropped Property

-

True if the control is currently dropped from the menu or toolbar based on usage statistics and layout space. (Note that this is not the same as the control's visibility, as set by the **Visible** property.) A control with **Visible** set to **True**, will not be immediately visible on a **Personalized Menu** or **Toolbar** if **IsPriorityDropped** is **True**. Read-only **Boolean**.

Remarks

To determine when to set **IsPriorityDropped** to **True** for a specific menu item, Microsoft Office maintains a total count of the number of times the menu item was used and a record of the number of different application sessions in which the user has used another menu item in the same menu as this menu item, without using the specific menu item. When this value reaches certain threshold values, the count is decremented. When the count reaches zero, **IsPriorityDropped** is set to **True**. Programmers cannot set the session value, the threshold value, or the **IsPriorityDropped** property. Programmers can, however, use the **AdaptiveMenus** property to disable adaptive menus for specific menus in an application.

To determine when to set **IsPriorityDropped** to **True** for a specific toolbar control, Office maintains a list of the order in which all the controls on that toolbar were last executed. A toolbar will always show as many controls as it has space to show, in the order of most recently used to least recently used. Controls with **Priority** set to 1 will always be shown and the toolbar will wrap rows, if necessary, to show these controls. Programmers can use the **Priority** property to ensure that specific toolbar controls are always shown, or to reposition toolbars so that they have enough space to display all of their controls.

You can use the following table to predict the number of sessions for which a menu item on a Personalized Menu will remain visible before the menu item's **IsPriorityDropped** property is set to **True**.

Number of uses of the command bar control	Number of sessions of the application
0, 1	3
2	6
3	9
4, 5	12
6-8	17
9–13	23
14–24	29
25 or more	31

This example checks the **IsPriorityDropped** property for the first control on the command bar named "Custom." If **IsPriorityDropped** is **True**, the example sets the **AdaptiveMenus** property to **False**, restoring the dropped command bar.

If CommandBars("Custom").Controls(1).IsPriorityDropped Then CommandBars.AdaptiveMenus = True End If

Issuer Property

Returns a **String** representing the name of the issuer of the digital certificate that corresponds to the **Signature** object. Read-only.

expression. Issuer

-

The following example prompts the user to select a digital signature with which to sign the active document in Microsoft Word. To use this example, open a document in Word and pass this function the name of a certificate issuer and the name of a certificate signer that match the **Issued By** and **Issued To** fields of a digital certificate in the **Digital Certificates** dialog box. This example will test to make sure that the digital signature that the user selects meets certain criteria, such as not having expired, before the new signature is committed to the disk.

```
Function AddSignature(ByVal strIssuer As String,
    strSigner As String) As Boolean
    On Error GoTo Error_Handler
    Dim sig As Signature
    'Display the dialog box that lets the
    'user select a digital signature.
    'If the user selects a signature, then
    'it is added to the Signatures
    'collection. If the user doesn't, then
    'an error is returned.
    Set sig = ActiveDocument.Signatures.Add
    'Test several properties before commiting the Signature object t
    If sig.Issuer = strIssuer And
        sig.Signer = strSigner And
        sig.IsCertificateExpired = False And _
        sig.IsCertificateRevoked = False And _
        siq.IsValid = True Then
        MsgBox "Signed"
        AddSignature = True
    'Otherwise, remove the Signature object from the SignatureSet co
    Else
        sig.Delete
        MsgBox "Not signed"
        AddSignature = False
    End If
    'Commit all signatures in the SignatureSet collection to the dis
    ActiveDocument.Signatures.Commit
```

Exit Function

```
Error_Handler:
AddSignature = False
MsgBox "Action cancelled."
End Function
```

IsValid Property

True if the digital certificate that corresponds to the **Signature** object is a valid certificate. A certificate may be invalid for several reasons ranging from its having expired to changes in the document that contains it. Read-only **Boolean**.

expression.IsValid

_
The following example prompts the user to select a digital signature with which to sign the active document in Microsoft Word. To use this example, open a document in Word and pass this function the name of a certificate issuer and the name of a certificate signer that match the **Issued By** and **Issued To** fields of a digital certificate in the **Digital Certificates** dialog box. This example will test to make sure that the digital signature that the user selects meets certain criteria, such as not having expired, before the new signature is committed to the disk.

```
Function AddSignature(ByVal strIssuer As String,
    strSigner As String) As Boolean
    On Error GoTo Error_Handler
    Dim sig As Signature
    'Display the dialog box that lets the
    'user select a digital signature.
    'If the user selects a signature, then
    'it is added to the Signatures
    'collection. If the user doesn't, then
    'an error is returned.
    Set sig = ActiveDocument.Signatures.Add
    'Test several properties before commiting the Signature object t
    If sig.Issuer = strIssuer And
        sig.Signer = strSigner And _
        sig.IsCertificateExpired = False And _
        sig.IsCertificateRevoked = False And _
        siq.IsValid = True Then
        MsgBox "Signed"
        AddSignature = True
    'Otherwise, remove the Signature object from the SignatureSet co
    Else
        sig.Delete
        MsgBox "Not signed"
        AddSignature = False
    End If
    'Commit all signatures in the SignatureSet collection to the dis
    ActiveDocument.Signatures.Commit
```

Exit Function

```
Error_Handler:
AddSignature = False
MsgBox "Action cancelled."
End Function
```

- <u>Show All</u>

Item Property

▶ Item property as it applies to the **FileTypes** collection.

Returns a value that indicates which file type will be searched for by the **Execute** method of the **FileSearch** object. Read-only **MsoFileType**.

MsoFileType can be one of these MsoFileType constants. **msoFileTypeAllFiles msoFileTypeBinders** msoFileTypeCalendarItem msoFileTypeContactItem **msoFileTypeDatabases** msoFileTypeDataConnectionFiles **msoFileTypeDesignerFiles** msoFileTypeDocumentImagingFiles msoFileTypeExcelWorkbooks msoFileTypeJournalItem **msoFileTypeMailItem** msoFileTypeNoteItem **msoFileTypeOfficeFiles msoFileTypeOutlookItems** msoFileTypePhotoDrawFiles msoFileTypePowerPointPresentations **msoFileTypeProjectFiles** msoFileTypePublisherFiles msoFileTypeTaskItem **msoFileTypeTemplates** msoFileTypeVisioFiles **msoFileTypeWebPages**

msoFileTypeWordDocuments

expression.Item(Index)

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

Index Required **Long**. The index number of the object to be returned.

▶ Item property as it applies to the **CommandBars** object.

Returns a **CommandBar** object from the **CommandBars** collection.

expression.Item(Index)

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

Index Required **Variant**. The name or index number of the object to be returned.

▶ Item property as it applies to the **CommandBarControls** object.

Returns a **CommandBarControl** object from the **CommandBarControls** collection.

expression.**Item**(*Index*)

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

Index Required **Variant**. The name or index number of the object to be returned.

▶ Item property as it applies to the **DocumentProperties** object.

Returns a **DocumentProperty** object from the **DocumentProperties** collection.

expression.Item(Index)

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

Index Required **Variant**. The name or index number of the document property

returned.

Item property as it applies to the BalloonCheckboxes and BalloonLabels objects.

Returns a **BalloonCheckBox** or **BalloonLabel** object.

expression.Item(Index)

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

Index Required **Long**. The index number of the check box or label to be returned.

▶ Item property as it applies to the **MsoEnvelope** object.

Returns a **MailItem** object that can be used to send the document as an e-mail.

expression.Item

expression Required. An expression that returns an **MsoEnvelope** object.

▶ <u>Item property as it applies to the **PropertyTests** object.</u>

Returns a **PropertyTest** object from the **PropertyTests** collection.

expression.Item(Index)

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

Index Required **Long**. The index number of the property test to be returned.

▶ Item property as it applies to the **ScopeFolders** and **SearchFolders** objects.

Returns a **ScopeFolder** object that represents a subfolder of the parent object.

expression.Item(Index)

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

Index Required **Long**. Determines which subfolder to return.

• Item property as it applies to the **SearchScopes** object.

Returns a **SearchScope** object that corresponds to an area in which to perform a file search, such as local drives or Microsoft Outlook folders.

expression.Item(Index)

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

Index Required **Long**. Determines which **SearchScope** object to return.

▶ Item property as it applies to the **SignatureSet** object.

Returns a **Signature** object that corresponds to one of the digital signatures with which the document is currently signed.

expression.Item(iSig)

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

iSig Required **Long**. Determines which **Signature** object to return.

▶ Item property as it applies to the **AnswerWizardFiles** and **FoundFiles** objects.

Returns a file name string from an **AnswerWizardFiles** collection, or a file name from the list of file names represented by the **FoundFiles** object, respectively. Read-only **String**.

expression.Item(Index)

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

Index Required **Long**. The index number of the Answer Wizard file name string, or the file name, to be returned.

Item property as it applies to the Assistant, BalloonCheckbox, BalloonLabel, and FileDialog objects.

Returns the text associated with an object. Read-only **String**.

expression.Item

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

▶ Item property as it applies to the **WebPageFonts** object.

Returns a **WebPageFont** object from the **WebPageFonts** collection for a particular value of **MsoCharacterSet**.

expression.Item(Index)

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

Index Required <u>MsoCharacterSet</u>. The specified character set.

MsoCharacterSet can be one of these MsoCharacterSet constants.

msoCharacterSetArabic msoCharacterSetCyrillic msoCharacterSetEnglishWesternEuropeanOtherLatinScript msoCharacterSetGreek msoCharacterSetHebrew msoCharacterSetJapanese msoCharacterSetJapanese msoCharacterSetKorean msoCharacterSetKorean msoCharacterSetThaiingualUnicode msoCharacterSetThai msoCharacterSetThai msoCharacterSetTraditionalChinese msoCharacterSetVietnamese

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

Item is the default member of the object or collection. The following two statements both assign a **CommandBar** object to cmdBar.

```
Set cmdBar = CommandBars.Item("Standard")
Set cmdBar = CommandBars("Standard")
```

• As it applies to the **BalloonCheckboxes** and **BalloonLabels** objects.

Item is the default member of the object or collection. The following two statements both assign to lblText the text of the first label in the **Balloon** object assigned to myBalloon.

```
lblText = myBalloon.Labels(1).Item
lblText = myBalloon.Labels(1)
```

• As it applies to the **AnswerWizardFiles** and **FoundFiles** objects.

This example resets the file list for the current Answer Wizard and displays both the file count and the file names in a message box, using the **Item** property to return the file names.

```
Dim customAnswerWizard As AnswerWizard
Dim strFileList As String
Dim intCounter As Integer
Dim intNumFiles As Integer
Set customAnswerWizard = Application.AnswerWizard
intCounter = 1
customAnswerWizard.ResetFileList
strFileList = ""
intNumFiles = customAnswerWizard.Files.Count
For intCounter = 1 To (intNumFiles)
    strFileList = strFileList & _
    customAnswerWizard.Files.Item(intCounter) & Chr(13)
Next
MsgBox "There are " & customAnswerWizard.Files.Count & _
    " files avaialble through this AnswerWizard: " & _
    Chr(13) & strFileList
```

• As it applies to the WebPageFonts object.

The following example uses the **Item** property to set myFont to the **WebPageFont** object for the English/Western European/Other Latin Script character set in the active application.

```
Dim myFont As WebPageFont
Set myFont = _
Application.DefaultWebOptions.Fonts. _
Item(msoCharacterSetEnglishWesternEuropeanOtherLatinScript)
```

▶ <u>As it applies to the **MsoEnvelope** object.</u>

The following example sends the active Microsoft Word document as an e-mail to the e-mail address that you pass to the subroutine.

```
Sub SendMail(ByVal strRecipient As String)
    'Use a With...End With block to reference the msoEnvelope object
    With Application.ActiveDocument.MailEnvelope
        'Add some introductory text before the body of the e-mail me
        .Introduction = "Please read this and send me your comments.
        'Return a MailItem object that you can use to send the docum
        With .Item
            'All of the mail item settings are saved with the docume
            'When you add a recipient to the Recipients collection
            'or change other properties these settings will persist.
            .Recipients.Add strRecipient
            .Subject = "Here is the document."
            'The body of this message will be
            'the content of the active document.
            .Send
        End With
    End With
End Sub
```

KeyboardShortcutTips Property

-

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

True if the Office Assistant displays Help about keyboard shortcuts. Read/write **Boolean**.

Remarks

The **KeyboardShortcutTips** property corresponds to the **Keyboard shortcuts** option in the **Show tips about** section on the **Options** tab in the **Office Assistant** dialog box.

This example sets the Office Assistant to provide Help information about keyboard shortcuts.

Assistant.KeyboardShortcutTips = True

Labels Property

-

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

Returns a **BalloonLabels** collection that represents the button labels, number labels, and bullet labels contained in the specified Office Assistant balloon. Read-only.

For information about returning a single member of a collection, see <u>Returning</u> <u>an Object from a Collection</u>.

This example creates a balloon containing three choices. The variable x is set to the return value of the **Show** method, which will be 1, 2 or 3, depending on the label the user clicks. In the example, a message box displays the value of the variable x, but you can pass the value to another procedure or you can use the value in a **Select Case** statement.

```
Set b = Assistant.NewBalloon
With b
    .Heading = "This is my heading"
    .Text = "Select one of these things:"
    .Labels(1).Text = "Choice One"
    .Labels(2).Text = "Choice Two"
    .Labels(3).Text = "Choice Three"
    x = .Show
End With
MsgBox x
```

Language Property

-

Specifies or returns the scripting language of the active script. Read/write **MsoScriptLanguage**.

MsoScriptLanguage can be one of these MsoScriptLanguage constants. msoScriptLanguageASP msoScriptLanguageJava msoScriptLanguageOther msoScriptLanguageVisualBasic

Remarks

The **MsoScriptLanguage** constants used with the **Language** property are also used in the **Language** parameter in the **Add** method of the **Scripts** collection.

This example checks the **Language** property to ensure that the first script in worksheet one in the active workbook is written in VBScript.

```
If ActiveWorkbook.Worksheets(1).Scripts(1).Language <> _
    msoScriptLanguageVisualBasic Then
    MsgBox "Language is not set to VBScript."
End If
```

LanguageID Property

Returns the locale identifier (LCID) for the install language, the user interface language, or the Help language. Read-only **Long**.

expression.LanguageID(Id)

-

expression Required. An expression that returns a LanguageSettings object.

Id Required MsoAppLanguageID.

MsoAppLanguageID constant	Meaning
	The language mode that the application is using. This setting applies only to Microsoft Excel and Microsoft Access. This setting affects languages that can be displayed and edited, available language-specific features, number styles, currency settings, and so forth.
	If none of the languages supported by Excel and Access are used, the host application will not be configured to support right-to-left and East Asian languages. The supported languages are as follows:
	Arabic
msoLanguageIDExeMode	Farsi
	Hebrew
	Japanese
	Korean

	Simplified Chinese
	Traditional Chinese
	Urdu
	Yiddish
msoLanguageIDHelp	The language used for online Help. The language settings used by Microsoft Office to set up defaults. For example, Microsoft Word
msoLanguageIDInstall	toolbars and the default types of bullets and numbers on a language-by-language basis.
msoLanguageIDUI	The language used by the host application's user interface.
msoLanguageIDUIPrevious	The language setting for the user interface when a given computer was last rebooted. A program or add-in can use this to determine whether the user interface language has changed.

This Microsoft Excel example checks the **LanguageID** property settings for the user interface and execution mode to verify that they are set to the same LCID. The example returns an error if there is a discrepancy.

If Application.LanguageSettings.LanguageID(msoLanguageIDExeMode) _
 <> Application.LanguageSettings.LanguageID(msoLanguageIDUI) _
 Then MsgBox "The user interface language and execution " & _
 "mode are different."

LanguagePreferredForEditing Property

Returns **True** if the value for the **msoLanguageID** constant has been identified in the Windows registry as a preferred language for editing. Read-only **Boolean**.

expression.LanguagePreferredForEditing(lid)

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

lid Required <u>MsoLanguageID</u>.

MsoLanguageID can be one of these MsoLanguageID constants. msoLanguageIDAfrikaans msoLanguageIDAlbanian msoLanguageIDAmharic **msoLanguageIDArabic** msoLanguageIDArabicAlgeria msoLanguageIDArabicBahrain msoLanguageIDArabicEgypt msoLanguageIDArabicIraq msoLanguageIDArabicJordan msoLanguageIDArabicKuwait msoLanguageIDArabicLebanon msoLanguageIDArabicLibya msoLanguageIDArabicMorocco msoLanguageIDArabicOman msoLanguageIDArabicQatar msoLanguageIDArabicSyria msoLanguageIDArabicTunisia

msoLanguageIDArabicUAE msoLanguageIDArabicYemen **msoLanguageIDArmenian** msoLanguageIDAssamese msoLanguageIDAzeriCyrillic msoLanguageIDAzeriLatin **msoLanguageIDBasque** msoLanguageIDBelgianDutch msoLanguageIDBelgianFrench msoLanguageIDBengali msoLanguageIDBrazilianPortuguese msoLanguageIDBulgarian **msoLanguageIDBurmese** msoLanguageIDByelorussian msoLanguageIDCatalan msoLanguageIDCherokee msoLanguageIDChineseHongKong msoLanguageIDChineseMacao msoLanguageIDChineseSingapore **msoLanguageIDCroatian** msoLanguageIDCzech msoLanguageIDDanish msoLanguageIDDutch **msoLanguageIDEnglishAUS** msoLanguageIDEnglishBelize msoLanguageIDEnglishCanadian msoLanguageIDEnglishCaribbean msoLanguageIDEnglishIreland msoLanguageIDEnglishJamaica msoLanguageIDEnglishNewZealand msoLanguageIDEnglishPhilippines msoLanguageIDEnglishSouthAfrica msoLanguageIDEnglishTrinidad

msoLanguageIDEnglishUK **msoLanguageIDEnglishUS** msoLanguageIDEnglishZimbabwe msoLanguageIDEstonian msoLanguageIDFaeroese msoLanguageIDFarsi **msoLanguageIDFinnish** msoLanguageIDFrench msoLanguageIDFrenchCameroon msoLanguageIDFrenchCanadian msoLanguageIDFrenchCotedIvoire msoLanguageIDFrenchLuxembourg msoLanguageIDFrenchMali msoLanguageIDFrenchMonaco msoLanguageIDFrenchReunion msoLanguageIDFrenchSenegal msoLanguageIDFrenchWestIndies msoLanguageIDFrenchZaire msoLanguageIDFrisianNetherlands msoLanguageIDGaelicIreland msoLanguageIDGaelicScotland msoLanguageIDGalician msoLanguageIDGeorgian msoLanguageIDGerman msoLanguageIDGermanAustria msoLanguageIDGermanLiechtenstein msoLanguageIDGermanLuxembourg msoLanguageIDGreek msoLanguageIDGujarati **msoLanguageIDHebrew** msoLanguageIDHindi msoLanguageIDHungarian msoLanguageIDIcelandic

msoLanguageIDIndonesian msoLanguageIDInuktitut **msoLanguageIDItalian** msoLanguageIDJapanese msoLanguageIDKannada msoLanguageIDKashmiri **msoLanguageIDKazakh** msoLanguageIDKhmer **msoLanguageIDKirghiz** msoLanguageIDKonkani msoLanguageIDKorean msoLanguageIDLao **msoLanguageIDLatvian** msoLanguageIDLithuanian msoLanguageIDMacedonian msoLanguageIDMalayalam msoLanguageIDMalayBruneiDarussalam msoLanguageIDMalaysian msoLanguageIDMaltese msoLanguageIDManipuri msoLanguageIDMarathi msoLanguageIDMexicanSpanish **msoLanguageIDMixed** msoLanguageIDMongolian msoLanguageIDNepali msoLanguageIDNone msoLanguageIDNoProofing msoLanguageIDNorwegianBokmol msoLanguageIDNorwegianNynorsk msoLanguageIDOriya msoLanguageIDOromo msoLanguageIDPolish msoLanguageIDPortuguese

msoLanguageIDPunjabi msoLanguageIDRhaetoRomanic msoLanguageIDRomanian msoLanguageIDRomanianMoldova **msoLanguageIDRussian** msoLanguageIDRussianMoldova msoLanguageIDSamiLappish msoLanguageIDSanskrit msoLanguageIDSerbianCyrillic msoLanguageIDSerbianLatin msoLanguageIDSesotho msoLanguageIDSimplifiedChinese msoLanguageIDSindhi msoLanguageIDSlovak msoLanguageIDSlovenian **msoLanguageIDSorbian msoLanguageIDSpanish** msoLanguageIDSpanishArgentina msoLanguageIDSpanishBolivia msoLanguageIDSpanishChile msoLanguageIDSpanishColombia msoLanguageIDSpanishCostaRica msoLanguageIDSpanishDominicanRepublic msoLanguageIDSpanishEcuador msoLanguageIDSpanishElSalvador msoLanguageIDSpanishGuatemala msoLanguageIDSpanishHonduras msoLanguageIDSpanishModernSort msoLanguageIDSpanishNicaragua msoLanguageIDSpanishPanama msoLanguageIDSpanishParaguay msoLanguageIDSpanishPeru msoLanguageIDSpanishPuertoRico

msoLanguageIDSpanishUruguay msoLanguageIDSpanishVenezuela **msoLanguageIDSutu** msoLanguageIDSwahili msoLanguageIDSwedish msoLanguageIDSwedishFinland msoLanguageIDSwissFrench msoLanguageIDSwissGerman msoLanguageIDSwissItalian **msoLanguageIDTajik msoLanguageIDTamil msoLanguageIDTatar msoLanguageIDTelugu msoLanguageIDThai msoLanguageIDTibetan** msoLanguageIDTigrignaEritrea msoLanguageIDTigrignaEthiopic msoLanguageIDTraditionalChinese msoLanguageIDTsonga **msoLanguageIDTswana** msoLanguageIDTurkish msoLanguageIDTurkmen msoLanguageIDUkrainian msoLanguageIDUrdu msoLanguageIDUzbekCyrillic msoLanguageIDUzbekLatin msoLanguageIDVenda msoLanguageIDVietnamese **msoLanguageIDWelsh msoLanguageIDXhosa** msoLanguageIDZulu

Remarks

You must test all valid **msoLanguageID** values to enumerate the set of preferred languages.

This example displays a message if U.S. English is a preferred editing language.

```
If Application.LanguageSettings. _
LanguagePreferredForEditing(msoLanguageIDEnglishUS) Then
MsgBox "One of the preferred editing languages is US English."
End If
```

LargeButtons Property

-

True if the toolbar buttons displayed are larger than normal size. Read/write **Boolean**.

This example toggles the display size of toolbar buttons on all command bars.

```
Set allBars = CommandBars
If allBars.LargeButtons Then
        allBars.LargeButtons = False
Else
        allBars.LargeButtons = True
End If
```

LastModified Property

Returns or sets a constant that represents the amount of time since the specified file was last modified and saved. The default value is **msoLastModifiedAnyTime**. Read/write **MsoLastModified**.

MsoLastModified can be one of these MsoLastModified constants.

msoLastModifiedAnyTime msoLastModifiedLastMonth msoLastModifiedLastWeek msoLastModifiedThisMonth msoLastModifiedThisWeek msoLastModifiedToday msoLastModifiedYesterday

This example sets options for a file search. The files this search returns were modified yesterday and are located in the C:\My Documents folder or in one of its subfolders.

```
Set fs = Application.FileSearch
With fs
    .LookIn = "C:\My Documents"
    .SearchSubFolders = True
    .LastModified = msoLastModifiedYesterday
End With
```

- <u>Show All</u>

Left Property

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

▶ Left property as it applies to the **Assistant** and **CommandBar** objects.

Sets or returns the horizontal position of the Office Assistant window (in points), or the distance (in pixels) of the <u>command bar</u>, from the left edge of the specified object relative to the screen. Read/write **Long**.

expression.Left

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

Left property as it applies to the CommandBarButton, CommandBarComboBox, CommandBarControl, and CommandBarPopup objects.

Set or returns the horizontal position of the specified <u>command bar control</u> (in pixels) relative to the left edge of the screen. Returns the distance from the left side of the docking area. Read-only **Long**.

expression.Left

expression Required. An expression that returns one of the above objects.
```
• As it applies to the Assistant and CommandBar objects.
```

This example displays the Office Assistant and moves it to the specified position within the application window.

```
With Assistant
.Visible = True
.Left = 300
.Top = 300
End With
```

This example moves the command bar named Custom from its docked position along the top of the window to the left edge of the window.

```
Set myBar = CommandBars("Custom")
With myBar
    .Position = 1
    .RowIndex = 2
    .Left = 0
End With
```

LinkSource Property

-

Returns or sets the source of a linked custom document property. Read/write **String**.

Remarks

This property applies only to custom document properties; you cannot use it with built-in document properties.

The source of the specified link is defined by the container application.

Setting the **LinkSource** property sets the **LinkToContent** property to **True**.

This example displays the linked status of a custom document property. For the example to work, dp must be a valid **DocumentProperty** object.

```
Sub DisplayLinkStatus(dp As DocumentProperty)
Dim stat As String, tf As String
If dp.LinkToContent Then
    tf = ""
Else
    tf = "not "
End If
stat = "This property is " & tf & "linked"
If dp.LinkToContent Then
    stat = stat + Chr(13) & "The link source is " & dp.LinkSourc
End If
MsgBox stat
End Sub
```

LinkToContent Property

-

True if the value of the custom document property is linked to the content of the container document. **False** if the value is static. Read/write **Boolean**.

Remarks

This property applies only to custom document properties. For built-in document properties, the value of this property is **False**.

Use the **LinkSource** property to set the source for the specified linked property. Setting the **LinkSource** property sets the **LinkToContent** property to **True**.

This example displays the linked status of the custom document property. For the example to work, dp must be a valid **DocumentProperty** object.

```
Sub DisplayLinkStatus(dp As DocumentProperty)
Dim stat As String, tf As String
If dp.LinkToContent Then
    tf = ""
Else
    tf = "not "
End If
stat = "This property is " & tf & "linked"
If dp.LinkToContent Then
    stat = stat + Chr(13) & "The link source is " & dp.LinkSourc
End If
MsgBox stat
End Sub
```

- <u>Show All</u>

List Property

Returns or sets an item in the command bar <u>combo box control</u>. Read/write **String**.

Note This property is read-only for built-in combo box controls.

expression.List(Index)

-

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

Index Required **Long**. The list item to be set.

This example checks the fourth list item in the combo box control whose caption is "Stock Data" on the command bar named "Custom." If the item isn't "View News," the example displays a message advising the user that the combo box may be damaged and asks the user to reinstall the application.

```
Set myBar = CommandBars
    .Add(Name:="Custom", Position:=msoBarTop, _
    Temporary:=True)
With myBar
    .Controls.Add Type:=msoControlComboBox, ID:=1
    .Visible = True
End With
With CommandBars("Custom").Controls(1)
    .AddItem "Get Stock Quote", 1
    .AddItem "View Chart", 2
    .AddItem "View Fundamentals", 3
    .AddItem "View News", 4
    .Caption = "Stock Data"
    .DescriptionText = "View Data For Stock"
End With
If CommandBars("Custom").Controls(1).List(4) _
     <> "View News" Then
MsgBox ("Stock Data appears to be damaged." & _
     " Please reinstall application.")
End If
```

- <u>Show All</u>

ListCount Property

-

Returns the number of list items in a command bar <u>combo box control</u>. Readonly **Long**.

This example checks the number of items in the combo box on the command bar named "Custom." If there aren't three items in the list that the procedure produces, the example displays a message advising the user that the combo box may be damaged and asks the user to reinstall the application.

```
Set myBar = CommandBars
    .Add(Name:="Custom", Position:=msoBarTop, _
    Temporary:=True)
With myBar
    .Controls.Add Type:=msoControlComboBox, ID:=1
    .Visible = True
End With
With CommandBars("Custom").Controls(1)
    .AddItem "Get Stock Quote", 1
    .AddItem "View Chart", 2
    .AddItem "View Fundamentals", 3
    .Caption = "Stock Data"
    .DescriptionText = "View Data For Stock"
End With
If CommandBars("Custom").Controls(1).ListCount _
     <> 4 Then
MsgBox ("ComboBox appears to be damaged." & _
     " Please reinstall.")
End If
```

- <u>Show All</u>

ListHeaderCount Property

-

Returns or sets the number of list items in a command bar <u>combo box control</u> that appears above the separator line. Read/write **Long**.

Note This property is read-only for built-in combo box controls.

Remarks

A **ListHeaderCount** property value of -1 indicates that there's no separator line in the combo box control.

This example adds a combo box control to the command bar named "Custom" and then adds two items to the combo box. The example uses the **ListHeaderCount** property to display a separator line between First Item and Second Item in the combo box. The example also sets the number of line items, the width of the combo box, and an empty default for the combo box.

```
Set myBar = CommandBars("Custom")
Set myControl = myBar.Controls.Add(Type:=msoControlComboBox)
With myControl
    .AddItem Text:="First Item", Index:=1
    .AddItem Text:="Second Item", Index:=2
    .DropDownLines = 3
    .DropDownWidth = 75
    .ListHeaderCount = 1
End With
```

- <u>Show All</u>

ListIndex Property

-

Returns or sets the index number of the selected item in the list portion of the command bar <u>combo box control</u>. If nothing is selected in the list, this property returns zero. Read/write **Long**.

Note This property fails when applied to controls other than list controls.

Remarks

Setting the **ListIndex** property causes the specified control to select the given item and execute the appropriate action in the application.

This example uses the **ListIndex** property to determine the correct subroutine to run, based on the selection in the combo box on the command bar named "My Custom Bar." Because the procedure uses **ListIndex**, the text in the combo box can be anything.

```
Sub processSelection()
Dim userChoice As Long
userChoice = CommandBars("My Custom Bar").Controls(1).ListIndex
Select Case userChoice
    Case 1
        chartcourse
        Case 2
            displaygraph
        Case Else
            MsgBox ("Invalid choice. Please choose again.")
        End Select
End Sub
```

Location Property

-

Returns the location of the script anchor in the specified HTML document. Read-only **MsoScriptLocation**.

MsoScriptLocation can be one of these MsoScriptLocation constants. msoScriptLocationInBody msoScriptLocationInHead

Remarks

Script tags in an HTML document can appear anywhere between the <HTML> tags in the document. In Microsoft Word, Excel, or PowerPoint, only the script anchors located between the <BODY> tags are visible. Additional HTML script that appears before or after the <BODY> tags is stored in but isn't visible to the user.

The **Scripts** collection contains all of the script anchors that appear in the document, whether inside or outside of the main body of the document. Using the **Location** argument of the **Add** method, you can insert script anchors within the <HEAD> and <BODY> tags in the HTML document. You can also use the **Location** property to determine where a particular script anchor is stored within the document.

This example checks the **Location** property of the first script in worksheet one in the active workbook and displays the location in a message box.

```
If ActiveWorkbook.Worksheets(1).Scripts(1).Location = 1 Then
    MsgBox "The script in located in the header."
End If
If ActiveWorkbook.Worksheets(1).Scripts(1).Location = 2 Then
    MsgBox "The script in located in the body of the worksheet."
End If
```

LookIn Property

-

Returns or sets the folder to be searched during the specified file search. Read/write **String**.

This example searches the My Documents folders for all files that begin with "Cmd" and displays the name and location of each file that's found.

```
Set fs = Application.FileSearch
With fs
.LookIn = "C:\My Documents"
.FileName = "cmd*.*"
If .Execute > 0 Then
    MsgBox "There were " & .FoundFiles.Count & _
    " file(s) found."
    For i = 1 To .FoundFiles.Count
        MsgBox .FoundFiles(i)
        Next i
    Else
        MsgBox "There were no files found."
    End If
End With
```

Mask Property

Returns an **IPictureDisp** object representing the mask image of a **CommandBarButton** object. The mask image determines what parts of the button image are transparent.

expression.Mask

-

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

Remarks

When you create an image that you plan on using as a mask image, all of the areas that you want to be transparent should be white, and all of the areas that you want to show should be black.

Always set the mask after you have set the picture for a **CommandBarButton** object.

The following example sets the image and mask of the first

CommandBarButton that the code returns. To make this work, create a mask image and a button image and sustitute the paths in the sample with the paths to your images.

```
Sub ChangeButtonImage()
   Dim picPicture As IPictureDisp
   Dim picMask As IPictureDisp
   Set picPicture = stdole.StdFunctions.LoadPicture( _
       "c:\images\picture.bmp")
   "c:\images\mask.bmp")
   'Reference the first button on the first command bar
   'using a With...End With block.
   With Application.CommandBars.FindControl(msoControlButton)
       'Change the button image.
       .Picture = picButton
       'Use the second image to define the area of the
       'button that should be transparent.
       .Mask = picMask
   End With
End Sub
```

The following example gets the image and mask of the first **CommandBarButton** that the code returns and outputs each of them to a file. To make this work, specify a path for the output files.

```
Sub GetButtonImageAndMask()
   Dim picPicture As IPictureDisp
   Dim picMask As IPictureDisp
   With Application.CommandBars.FindControl(msoControlButton)
      'Get the button image and mask of the this CommandBarButton
      Set picPicture = .Picture
      Set picMask = .Mask
End With
   'Save the button image and mask in a folder.
   stdole.SavePicture picPicture, "c:\temp\image.bmp"
```

stdole.SavePicture picMask, "c:\temp\mask.bmp"
End Sub

MatchAllWordForms Property

-

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

True if the file search is expanded to include all forms of the specified word contained in the body of the file, or in the file's properties. Read/write **Boolean**.

Remarks

This property is available only if the file Mswds_en.lex has been installed and registered. Note that this file isn't installed as part of a Typical setup.

This example returns all files that contain the word "run," "running," "runs," or "ran" in the body of the file, or in the properties of the file. The **TextOrProperty** property sets the word to be matched, and limits the search to either the body of the file or the file properties.

```
With Application.FileSearch
    .NewSearch
    .LookIn = "C:\My Documents"
    .SearchSubFolders = True
    .TextOrProperty = "run"
    .MatchAllWordForms = True
    .FileType = msoFileTypeAllFiles
End With
```

MatchTextExactly Property

-

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

True if the specified file search will find only files whose body text or file properties contain the exact word or phrase that you've specified. Read/write **Boolean**.

This example searches the C:\My Documents folder and returns all files that contain the word "Run" either in the body text or in the file properties.

```
With Application.FileSearch
    .NewSearch
    .LookIn = "C:\My Documents"
    .TextOrProperty = "Run"
    .MatchTextExactly = True
    .FileType = msoFileTypeAllFiles
End With
```

- <u>Show All</u>
MenuAnimationStyle Property

Returns or sets the way a <u>command bar</u> is animated. Read/write <u>MsoMenuAnimation</u>.

-

MsoMenuAnimation can be one of these MsoMenuAnimation constants. msoMenuAnimationNone msoMenuAnimationRandom msoMenuAnimationSlide msoMenuAnimationUnfold

This example sets options for all command bars in Microsoft Office.

```
With CommandBars
    .LargeButtons = True
    .DisplayTooltips = True
    .DisplayKeysInTooltips = True
    .MenuAnimationStyle = msoMenuAnimationUnfold
End With
```

Mode Property

-

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

Returns or sets the modal behavior of the Office Assistant balloon. When you create a **Balloon** object, this property is initially set to **msoModeModal**. Read/write **MsoModeType**.

MsoModeType can be one of these MsoModeType constants. msoModeAutoDown msoModeModal msoModeModeless

If the **Mode** property for a balloon is set to **msoModeModeless**, the user can work in the application while the balloon is visible. If the property is set to **msoModeModal**, the user must dismiss the balloon before continuing to work in the application. If the property is set to **msoModeAutoDown**, the balloon is dismissed when the user clicks anywhere on the screen.

If the **Mode** property for a balloon is set to **msoModeModeless** and a value for the **Callback** property is not supplied, an error occurs. The **Close** method can only be used in the procedure specified by the **Callback** property when the **Mode** property is set to **msoModeModeless**.

This example creates a balloon with an alert icon that instructs the user to select a printer. Because the balloon is modeless, the user has access to printer commands while the balloon is visible. When the user clicks the **OK** button, the procedure specified in the **Callback** property is run.

```
With Assistant.NewBalloon
    .Heading = "Select A Printer"
    .Text = "You must select a printer before printing."
    .Icon = msoIconAlert
    .CheckBoxes(1).Text = "Local printer"
    .CheckBoxes(2).Text = "Network printer"
    .Mode = msoModeModeless
    .Callback = "ProcessPrinter"
    .Show
End With
```

MouseTips Property

-

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

True if the Office Assistant provides suggestions for using the mouse effectively. Read/write **Boolean**.

The **MouseTips** property corresponds to the **Using the mouse more effectively** option under **Show tips about** on the **Options** tab in the **Office Assistant** dialog box.

This example sets the Office Assistant to provide suggestions for using the mouse effectively.

Assistant.MouseTips = True

MoveWhenInTheWay Property

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

True if the Office Assistant window automatically moves when it's in the way of the user's work area. For example, the Assistant will move if it's in the way of dragging or dropping or in the way of keystroke entries. Read/write **Boolean**.

The default value is **True**. The **MoveWhenInTheWay** property corresponds to the **Move when in the way** option in the Use the **Office Assistant** section on the **Options** tab in the **Office Assistant** dialog box.

This example displays the Office Assistant in a specific location and it sets several options before making the Assistant visible and active.

```
With Assistant
   .On = True
   .Visible = True
   .Left = 400
   If Not MoveWhenInTheWay Then MoveWhenInTheWay = True
   .Animation = msoAnimationGetAttentionMajor
End With
```

- <u>Show All</u>

Name Property

-

Returns or sets the name of the specified object. Read/write **String** for the **CommandBar** and **DocumentProperty** objects; read-only **String** for all other objects.

The local name of a built-in <u>command bar</u> is displayed in the title bar (when the command bar isn't docked) and in the list of available command bars — wherever that list is displayed in the container application.

For a built-in command bar, the **Name** property returns the command bar's U.S. English name. Use the **NameLocal** property to return the localized name.

If you change the value of the **LocalName** property for a custom command bar, the value of **Name** changes as well, and vice versa.

This example searches the collection of command bars for the command bar named "Custom." If this command bar is found, the example makes it visible.

```
foundFlag = False
For Each bar In CommandBars
    If bar.Name = "Custom" Then
        foundFlag = True
        bar.Visible = True
    End If
Next
If Not foundFlag Then
        MsgBox "'Custom' bar isn't in collection."
Else
        MsgBox "'Custom' bar is now visible."
End If
```

This example displays the name, type, and value of a document property. You must pass a valid **DocumentProperty** object to the procedure.

```
Sub DisplayPropertyInfo(dp As DocumentProperty)
    MsgBox "value = " & dp.Value & Chr(13) & _
        "type = " & dp.Type & Chr(13) & _
        "name = " & dp.Name
End Sub
```

- <u>Show All</u>

NameLocal Property

-

Returns the name of a built-in <u>command bar</u> as it's displayed in the language version of the container application, or returns or sets the name of a custom command bar. Read/write **String**.

Note If you attempt to set this property for a built-in command bar, an error occurs.

The local name of a built-in command bar is displayed in the title bar (when the command bar isn't docked) and in the list of available command bars, wherever that list is displayed in the container application.

If you change the value of the **LocalName** property for a custom command bar, the value of **Name** changes as well, and vice versa.

This example displays the name and localized name of the first command bar in the container application.

With CommandBars(1)

```
MsgBox "The name of the command bar is " & .Name
MsgBox "The localized name of the command bar is " & .NameLocal
End With
```

NewBalloon Property

-

Some of the content in this topic may not be applicable to some languages. Creates an Office Assistant balloon. Returns a **Balloon** object. Read-only.

This example creates a balloon with a heading, text, and three region choices, and then displays it.

```
With Assistant.NewBalloon
    .Heading = "Regional Sales Data"
    .Text = "Select one or more regions"
    For i = 1 To 3
        .CheckBoxes(i).Text = "Region " & i
        Next
        .Show
End With
```

Object Property

-

Sets or returns the object that is the basis for the specified **COMAddIn** object. Read/write **Object**.

This property is primarily used for enabling one **COMAddIn** to communicate with another **COMAddIn**.

The following example returns the object associated with the COM add in msodraa9.ShapeSelect.

```
Dim objBaseObject As Object
Set objBaseObject = _
Application.COMAddIns.Item("msodraa9.ShapeSelect"). _
Object
```

- <u>Show All</u>

OLEMenuGroup Property

Returns or sets the menu group that the specified command bar <u>pop-up control</u> belongs to when the menu groups of the OLE server are merged with the menu groups of an OLE client (that is, when an object of the container application type is embedded in another application). Read/write <u>MsoOLEMenuGroup</u>.

MsoOLEMenuGroup can be one of these MsoOLEMenuGroup constants.

msoOLEMenuGroupContainer msoOLEMenuGroupEdit msoOLEMenuGroupFile msoOLEMenuGroupHelp msoOLEMenuGroupNone msoOLEMenuGroupObject msoOLEMenuGroupWindow

Note This property is read-only for built-in controls.

This property is intended to allow add-in applications to specify how their command bar controls will be represented in the Office application. If either the container or the server does not implement command bars, normal OLE menu merging will occur: the menu bar will be merged, as well as all the toolbars from the server, and none of the toolbars from the container. This property is relevant only for pop-up controls on the menu bar because menus are merged on the basis of their menu group category.

If both of the merging applications implement command bars, command bar controls are merged according to the **OLEUsage** property.

This example checks the **OLEMenuGroup** property of a new custom pop-up control on the command bar named "Custom" and sets the property to **msoOLEMenuGroupNone**.

Set myControl = CommandBars("Custom").Controls _
.Add(Type:=msoControlPopup,Temporary:=False)
myControl.OLEMenuGroup = msoOLEMenuGroupNone

- <u>Show All</u>

OLEUsage Property

-

Returns or sets the OLE client and OLE server roles in which a <u>command bar</u> <u>control</u> will be used when two Microsoft Office applications are merged. Read/write <u>MsoControlOLEUsage</u>.

MsoControlOLEUsage can be one of these MsoControlOLEUsage constants. msoControlOLEUsageBoth msoControlOLEUsageClient msoControlOLEUsageNeither msoControlOLEUsageServer

This property is intended to allow you to specify how individual add-in applications' command bar controls will be represented in one Office application when it is merged with another Office application. If both the client and server implement command bars, the command bar controls are embedded in the client control by control. Custom controls marked as client-only (or neither client nor server) are dropped from the server, and controls marked as server-only (or neither server nor client) are dropped from the client. The remaining controls are merged.

If one of the merging applications isn't an Office application, normal OLE menu merging is used, which is controlled by the **OLEMenuGroup** property.

This example adds a new button to the command bar named Tools, and sets its **OLEUsage** property.

```
Set myControl = CommandBars("Tools").Controls _
.Add(Type:=msoControlButton,Temporary:=True)
myControl.OLEUsage = msoControlOLEUsageNeither
```

On Property

-

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

True if the Office Assistant is enabled. Read/write **Boolean**.

This example disables the Office Assistant, displays a message box that asks the user whether the Assistant should be enabled, and enables the Assistant if the user clicks **Yes**. If the users enables the Assistant, the Assistant appears and performs the animation **msoAnimationGreeting**.

```
Assistant.On = False
If MsgBox("Enable Office Assistant?", _
    vbYesNo, "Assistant is Off") = vbYes Then
    Assistant.On = True
    Assistant.Visible = True
    Assistant.Animation = _
        msoAnimationGetAttentionMajor
End If
```

- <u>Show All</u>

OnAction Property

-

Returns or sets the name of a Visual Basic procedure that will run when the user clicks or changes the value of a <u>command bar control</u>. Read/write **String**.

Note The container application determines whether the value is a valid macro name.
This example adds a command bar control to the command bar named "Custom". The procedure named "MySub" will run each time the control is clicked.

```
Set myBar = CommandBars("Custom")
Set myControl = myBar.Controls _
    .Add(Type:=msocontrolButton)
With myControl
    .FaceId = 2
    .OnAction = "MySub"
End With
myBar.Visible = True
```

This example adds a command bar control to the command bar named "Custom". The COM add in named "FinanceAddIn" will run each time the control is clicked.

```
Set myBar = CommandBars("Custom")
Set myControl = myBar.Controls _
   .Add(Type:=msocontrolButton)
With myControl
   .FaceId = 2
   .OnAction = "!<FinanceAddIn>"
End With
myBar.Visible = True
```

Parameter Property

-

Returns or sets a string that an application can use to execute a command. Read/write **String**.

If the specified parameter is set for a built-in control, the application can modify its default behavior if it can parse and use the new value. If the parameter is set for custom controls, it can be used to send information to Visual Basic procedures, or it can be used to hold information about the control (similar to a second **Tag** property value).

This example assigns a new parameter to a control and sets the focus to the new button.

```
Set myControl = CommandBars("Custom").Controls(4)
With myControl
    .Copy , 1
    .Parameter = "2"
    .SetFocus
End With
```

Parent Property

-

Returns the **Parent** object for the specified object. Read-only.

This example displays the name of the parent object for a document property. You must pass a valid **DocumentProperty** object to the procedure.

Sub DisplayParent(dp as DocumentProperty) MsgBox dp.**Parent**.Name End Sub

Path Property

Returns a **String** indicating the full path of a **ScopeFolder** object. Read-only.

expression.Path

-

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

The following example displays the root path of each directory in My Computer. To retrieve this information, the example first gets the **ScopeFolder** object at the root of My Computer. The path of this **ScopeFolder** will always be "*". As with all **ScopeFolder** objects, the root object contains a **ScopeFolders** collection. This example loops through this **ScopeFolders** collection and displays the path of each **ScopeFolder** object in it. The paths of these **ScopeFolder** objects will be "A:\", "C:\", etc.

```
Sub DisplayRootScopeFolders()
    'Declare variables that reference a
    'SearchScope and a ScopeFolder object.
    Dim ss As SearchScope
    Dim sf As ScopeFolder
    'Use a With...End With block to reference the
    'FileSearch object.
    With Application.FileSearch
        'Loop through the SearchScopes collection
        'and display all of the root ScopeFolders collections in
        'the My Computer scope.
        For Each ss In .SearchScopes
            Select Case ss.Type
                Case msoSearchInMyComputer
                     'Loop through each ScopeFolder collections in
                     'the ScopeFolders collection of the
                    'SearchScope object and display the path.
                    For Each sf In ss.ScopeFolder.ScopeFolders
                        MsgBox "Path: " & sf.Path
                    Next sf
                Case Else
            End Select
        Next ss
    End With
End Sub
```

Picture Property

Returns an **IPictureDisp** object representing the image of a **CommandBarButton** object.

expression.Picture

-

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

When you change the image on a button, you will also want to use the **Mask** property to set a mask image. The mask image determines which parts of the button image are transparent. Always set the mask after you have set the picture for a **CommandBarButton** object.

Note The images for the **View Microsoft** *Application* and **Insert** *Item* buttons on the **Standard** toolbar in the Visual Basic Editor cannot be changed.

The following example sets the image and mask of the first

CommandBarButton that the code returns. To make this work, create a mask image and a button image and substitute the paths in the sample with the paths to your images.

```
Sub ChangeButtonImage()
   Dim picPicture As IPictureDisp
   Dim picMask As IPictureDisp
   Set picPicture = stdole.StdFunctions.LoadPicture( _
       "c:\images\picture.bmp")
   "c:\images\mask.bmp")
   'Reference the first button on the first command bar
   'using a With...End With block.
   With Application.CommandBars.FindControl(msoControlButton)
       'Change the button image.
       .Picture = picButton
       'Use the second image to define the area of the
       'button that should be transparent.
       .Mask = picMask
   End With
End Sub
```

The following example gets the image and mask of the first **CommandBarButton** that the code returns and outputs each of them to a file. To make this work, specify a path for the output files.

```
Sub GetButtonImageAndMask()
   Dim picPicture As IPictureDisp
   Dim picMask As IPictureDisp
   With Application.CommandBars.FindControl(msoControlButton)
      'Get the button image and mask of the this CommandBarButton
      Set picPicture = .Picture
      Set picMask = .Mask
End With
   'Save the button image and mask in a folder.
   stdole.SavePicture picPicture, "c:\temp\image.bmp"
```

stdole.SavePicture picMask, "c:\temp\mask.bmp"
End Sub

- <u>Show All</u>

Position Property

-

Returns or sets the position of a <u>command bar</u>. Read/write <u>MsoBarPosition</u>.

MsoBarPosition can be one of these MsoBarPosition constants. msoBarBottom msoBarFloating msoBarLeft msoBarMenuBar msoBarPopup msoBarRight msoBarTop

This example steps through the collection of command bars, docking the custom command bars at the bottom of the application window and docking the built-in command bars at the top of the window.

```
For Each bar In CommandBars
   If bar.Visible = True Then
        If bar.BuiltIn Then
        bar.Position = msoBarTop
        Else
        bar.Position = msoBarBottom
        End If
   End If
   Next
```

- <u>Show All</u>

Priority Property

-

Returns or sets the priority of a <u>command bar control</u>. A control's priority determines whether the control can be dropped from a docked command bar if the command bar controls can't fit in a single row. Read/write **Long**.

Valid priority numbers are 0 (zero) through 7. A priority of 1 means that the control cannot be deleted from a toolbar. Other priority values are ignored.

The **Priority** property is not used by command bar controls that are menu items.

This example moves a control and assigns it a priority of 5 so that it will likely be dropped from the command bar if the controls don't all fit in one row.

```
Set allcontrols = CommandBars("Custom").Controls
For Each ctrl In allControls
    If ctrl.Type = msoControlComboBox Then
        With ctrl
        .Move Before:=7
        .Tag = "Selection box"
        .Priority = 5
        End With
        Exit For
End If
Next
```

Private Property

_

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

Returns or sets an integer that identifies the Office Assistant balloon that initiated the callback procedure. Read/write **Long**.

This property is helpful if you run the same callback procedure from more than one balloon.

This example identifies the Office Assistant balloon by setting the **Private** property to 129.

```
Set printerOrientation = Assistant.NewBalloon
With printerOrientation .Heading = " Print portrait or landscape
.Text = "Click OK when you've selected the " & _
        "printer orientation."
.Labels(1).Text = "Portrait"
.Labels(2).Text = "Landscape"
.BalloonType = msoBalloonTypeButtons
.Mode = msoModeModeless
.Button = msoButtonSetOK
.Private = 129
.Callback = "PortraitOrLandscape "
.Show
End With
```

ProgId Property

-

Returns the programmatic identifier (ProgID) for the specified **COMAddIn** object. Read-only **String**.

The following example displays the ProgID and GUID for COM add-in one in a message box.

```
MsgBox "My ProgID is " & _
Application.COMAddIns(1).ProgID & _
" and my GUID is " & _
Application.COMAddIns(1).Guid
```

PropertyTests Property

-

Returns the **PropertyTests** collection that represents all the search criteria for a file search. Read-only.

For information about returning a single member of a collection, see <u>Returning</u> <u>an Object from a Collection</u>.

This example displays all the search criteria for the first property test in the collection.

ProportionalFont Property

-

Sets or returns the proportional font setting in the host application. Read/write **String**.

When you set the **ProportionalFont** property, the host application does not check the value for validity.

This example sets the proportional font and proportional font size for the English/Western European/Other Latin Script character set in the active application.

```
Application.DefaultWebOptions. _
Fonts(msoCharacterSetEnglishWesternEuropeanOtherLatinScript) _
.ProportionalFont = "Tahoma"
Application.DefaultWebOptions. _
Fonts(msoCharacterSetEnglishWesternEuropeanOtherLatinScript) _
.ProportionalFontSize = 14.5
```

ProportionalFontSize Property

-

Sets or returns the proportional font size setting in the host application, in points. Read/write **Single**.

When you set the **ProportionalFontSize** property, the host application does not check the value for validity. If you enter an invalid value, such as a nonnumber, the host application sets the size to 0 points. You can enter half-point sizes; if you enter other fractional point sizes, they are rounded up or down to the nearest half-point.

This example sets the proportional font and proportional font size for the English/Western European/Other Latin Script character set in the active application.

```
Application.DefaultWebOptions. _
Fonts(msoCharacterSetEnglishWesternEuropeanOtherLatinScript) _
.ProportionalFont = "Tahoma"
Application.DefaultWebOptions. _
Fonts(msoCharacterSetEnglishWesternEuropeanOtherLatinScript) _
.ProportionalFontSize = 14.5
```

- <u>Show All</u>

Protection Property

-

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

Returns or sets the way a <u>command bar</u> is protected from user customization. Read/write <u>MsoBarProtection</u>.

MsoBarProtection can be one of these MsoBarProtection constants. msoBarNoChangeDock msoBarNoChangeVisible msoBarNoCustomize msoBarNoHorizontalDock msoBarNoMove msoBarNoProtection msoBarNoResize msoBarNoVerticalDock

Using the constant **msoBarNoCustomize** prevents users from accessing the **Add or Remove Buttons** menu (this menu enables users to customize a toolbar).
This example steps through the collection of command bars to find the command bar named "Forms." If this command bar is found, it's docking state is protected and it's made visible.

```
foundFlag = False
For i = 1 To CommandBars.Count
    If CommandBars(i).Name = "Forms" Then
        CommandBars(i).Protection = msoBarNoChangeDock
        CommandBars(i).Visible = True
        foundFlag = True
    End If
Next
If Not foundFlag Then
        MsgBox "'Forms' command bar is not in the collection."
End If
```

Reduced Property

-

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

True if the Office Assistant window appears in its smaller size. Read/write **Boolean**.

Remarks

This property is not used in Microsoft Office.

This example displays the Office Assistant in a specific location and it sets several options before making the Assistant visible.

```
With Assistant
    .Reduced = True
    .Left = 400
    .MoveWhenInTheWay = True
    .TipOfDay = True
    .Visible = True
    .Animation = msoAnimationGreeting
End With
```

RowCount Property

Returns a **Long** that represents the number of records in the specified data source. Read-only.

expression.RowCount

-

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

This example adds a new filter that removes all records with a blank Region field and then applies the filter to the active publication.

```
Sub OfficeFilters()
Dim appOffice As OfficeDataSourceObject
Dim appFilters As ODSOFilters
Set appOffice = Application.OfficeDataSourceObject
appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName
    "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
Set appFilters = appOffice.Filters
MsgBox appOffice.RowCount
appFilters.Add Column:="Region", Comparison:=msoFilterComparison
    Conjunction:=msoFilterConjunctionAnd, bstrCompareTo:="WA"
appOffice.ApplyFilter
MsgBox appOffice.RowCount
```

End Sub

- <u>Show All</u>

RowIndex Property

-

Returns or sets the docking order of a <u>command bar</u> in relation to other command bars in the same docking area. Can be an integer greater than zero, or either of the following **MsoBarRow** constants: **msoBarRowFirst** or **msoBarRowLast**. Read/write **Long**.

Remarks

Several command bars can share the same row index, and command bars with lower numbers are docked first. If two or more command bars share the same row index, the command bar most recently assigned will be displayed first in its group.

This example adjusts the position of the command bar named "Custom" by moving it to the left 110 pixels more than the default, and it makes this command bar the first to be docked by changing its row index to **msoBarRowFirst**.

```
Set myBar = CommandBars("Custom")
With myBar
    .RowIndex = msoBarRowFirst
    .Left = 140
End With
```

ScopeFolder Property

Returns a **<u>ScopeFolder</u>** object.

expression.ScopeFolder

-

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

The following example displays the root path of each directory in My Computer. To retrieve this information, the example first gets the **ScopeFolder** object at the root of My Computer. The path of this **ScopeFolder** will always be "*". As with all **ScopeFolder** objects, the root object contains a **ScopeFolders** collection. This example loops through this **ScopeFolders** collection and displays the path of each **ScopeFolder** object in it. The paths of these **ScopeFolder** objects will be "A:\", "C:\", etc.

```
Sub DisplayRootScopeFolders()
    'Declare variables that reference a
    'SearchScope and a ScopeFolder object.
    Dim ss As SearchScope
    Dim sf As ScopeFolder
    'Use a With...End With block to reference the
    'FileSearch object.
    With Application.FileSearch
        'Loop through the SearchScopes collection
        'and display all of the root ScopeFolders collections in
        'the My Computer scope.
        For Each ss In .SearchScopes
            Select Case ss.Type
                Case msoSearchInMyComputer
                     'Loop through each ScopeFolder object in
                     'the ScopeFolders collection of the
                    'SearchScope object and display the path.
                    For Each sf In ss.ScopeFolder.ScopeFolders
                        MsgBox "Path: " & sf.Path
                    Next sf
                Case Else
            End Select
        Next ss
    End With
End Sub
```

ScopeFolders Property

Returns a **<u>ScopeFolders</u>** collection. The items in this collection correspond to the subfolders of the parent **ScopeFolder** object.

expression.ScopeFolders

-

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

The following example displays the root path of each directory in My Computer. To retrieve this information, the example first gets the **ScopeFolder** object at the root of My Computer. The path of this **ScopeFolder** will always be "*". As with all **ScopeFolder** objects, the root object contains a **ScopeFolders** collection. This example loops through this **ScopeFolders** collection and displays the path of each **ScopeFolder** object in it. The paths of these **ScopeFolder** objects will be "A:\", "C:\", etc.

```
Sub DisplayRootScopeFolders()
    'Declare variables that reference a
    'SearchScope and a ScopeFolder object.
    Dim ss As SearchScope
    Dim sf As ScopeFolder
    'Use a With...End With block to reference the
    'FileSearch object.
    With Application.FileSearch
        'Loop through the SearchScopes collection
        'and display all of the root ScopeFolders collections in
        'the My Computer scope.
        For Each ss In .SearchScopes
            Select Case ss.Type
                Case msoSearchInMyComputer
                     'Loop through each ScopeFolder object in
                     'the ScopeFolders collection of the
                    'SearchScope object and display the path.
                    For Each sf In ss.ScopeFolder.ScopeFolders
                        MsgBox "Path: " & sf.Path
                    Next sf
                Case Else
            End Select
        Nextb ss
    End With
End Sub
```

ScriptText Property

-

Sets or returns the text contained in a block of script. Read/write **String**.

Remarks

The Microsoft Office host application doesn't check the syntax of the script. The **ScriptText** property is the default property for the **Script** object.

This example sets the text of the first script in worksheet one in the active workbook.

```
ActiveWorkbook.Worksheets(1).Scripts(1).ScriptText = _
    "MsgBox ""New ScriptText"""
```

SearchFolders Property

Returns a **<u>SearchFolders</u>** collection.

expression.SearchFolders

-

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

The following example displays the current number of **ScopeFolder** objects in the **SearchFolders** collection. See the **SearchFolders** collection topic for a more detailed example.

MsgBox "Number of ScopeFolder objects in the SearchFolders collectic Application.FileSearch.SearchFolders.Count

SearchScopes Property

Returns a **<u>SearchScopes</u>** collection.

expression.SearchScopes

-

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

The following example displays all of the currently available **SearchScope** objects in the **SearchScopes** collection.

```
Sub DisplayAvailableScopes()
    'Declare a variable that references a
    'SearchScope object.
    Dim ss As SearchScope
    'Use a With...End With block to reference the
    'FileSearch object.
   With Application.FileSearch
        'Loop through the SearchScopes collection
        For Each ss In .SearchScopes
            Select Case ss.Type
                Case msoSearchInMyComputer
                    MsgBox "My Computer is an available search scope
                Case msoSearchInMyNetworkPlaces
                    MsgBox "My Network Places is an available search
                Case msoSearchInOutlook
                    MsgBox "Outlook is an available search scope."
                Case msoSearchInCustom
                    MsgBox "A custom search scope is available."
                Case Else
                    MsqBox "Can't determine search scope."
            End Select
        Next ss
    End With
End Sub
```

SearchSubFolders Property

-

True if the search includes all the subfolders in the folder specified by the **LookIn** property. Read/write **Boolean**.

This example searches the My Documents folder and all of its subfolders for all files whose names begin with "Cmd." The example also displays the name and location of each file that's found.

```
Set fs = Application.FileSearch
With fs
.LookIn = "C:\My Documents"
.SearchSubFolders = True
.FileName = "cmd*"
If .Execute() > 0 Then
    MsgBox "There were " & .FoundFiles.Count & _
    " file(s) found."
    For i = 1 To .FoundFiles.Count
        MsgBox .FoundFiles(i)
        Next i
Else
        MsgBox "There were no files found."
End If
End With
```

SearchWhenProgramming Property

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

_

True if the Office Assistant displays application and programming Help while the user is working in Visual Basic. Read/write **Boolean**.

Remarks

The default value is **False**. **SearchWhenProgramming** property corresponds to the **Search for both product and programming help when programming** option in the Use the **Office Assistant** section on the **Options** tab in the **Office Assistant** dialog box.

This example allows the user to search both application and programming help while working in Visual Basic.

Assistant.**SearchWhenProgramming** = True

SecondValue Property

-

Returns an optional second value property test (as in a range) for the file search. Read-only **Variant**.

Remarks

This property is intended to be used to specify a range, and it can only be used with the **MsoCondition** constant **msoConditionAnyTimeBetween** or **msoConditionAnyNumberBetween**.

This example displays the second value of the search criteria (if it exists) in a dialog box. If the second value doesn't exist, the example displays another message.

```
With Application.FileSearch.PropertyTests(1)
If .SecondValue = "" Then
    MsgBox "You haven't specified a second value."
Else
    MsgBox "The second value you've set is: " _
    & .SecondValue
End If
End With
```

SelectedItems Property

Returns a **FileDialogSelectedItems** collection. This collection contains a list of the paths of the files that a user selected from a file dialog box displayed using the **Show** method of the **FileDialog** object.

expression.SelectedItems

-

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

The following example displays a File Picker dialog box using the **FileDialog** object and displays each selected file in a message box.

```
Sub Main()
    'Declare a variable as a FileDialog object.
    Dim fd As FileDialog
    'Create a FileDialog object as a File Picker dialog box.
    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    'Declare a variable to contain the path
    'of each selected item. Even though the path is a String,
    'the variable must be a Variant because For Each...Next
    'routines only work with Variants and Objects.
    Dim vrtSelectedItem As Variant
    'Use a With...End With block to reference the FileDialog object.
   With fd
        'Allow the user to select multiple files.
        .AllowMultiSelect = True
        'Use the Show method to display the File Picker dialog box a
        'If the user presses the action button...
        If .Show = -1 Then
            'Step through each string in the FileDialogSelectedItems
            For Each vrtSelectedItem In .SelectedItems
                'vrtSelectedItem is a String that contains the path
                'You can use any file I/O functions that you want to
                'This example simply displays the path in a message
                MsgBox "Selected item's path: " & vrtSelectedItem
            Next
        'If the user presses Cancel...
        Else
        End If
    End With
    'Set the object variable to Nothing.
    Set fd = Nothing
```

End Sub

Shape Property

-

Returns a **Shape** object or **InlineShape** object, depending on the Microsoft Office host application. Read-only **Object**.

Remarks

The **Shape** property returns a **Shape** object in Microsoft Excel and PowerPoint. In Word, the **Shape** property returns a **Shape** object if the script anchor is floating; if it's an inline anchor, however, this property returns an **InlineShape** object.
This example gets the shape associated with the first script in the **Scripts** collection and deletes it from worksheet one in the active workbook.

```
Dim objScriptShape As Object
Set objScriptShape = ______
ActiveWorkbook.Worksheets(1).Scripts(1).Shape
ObjScriptShape.Delete
```

- <u>Show All</u>

ShortcutText Property

-

Returns or sets the shortcut key text displayed next to a <u>button control</u> when the button appears on a menu, submenu, or shortcut menu. Read/write **String**.

Remarks

You can set this property only for command bar buttons that contain an **OnAction** macro.

This example displays the shortcut text for the **Open** command (**File** menu) on the Microsoft Excel Worksheet menu bar in a message box.

```
MsgBox (CommandBars("Worksheet Menu Bar"). _
Controls("File").Controls("New...).ShortcutText)
```

SignDate Property

Returns a **Variant** representing the date and time that the digital certificate corresponding to the **Signature** object was attached to the document. Read-only.

expression.SignDate

-

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

The following example prompts the user to select a digital signature with which to sign the active document in Microsoft Word. To use this example, open a document in Word and call this function. The function will test to make sure that the digital signature that the user selects will not expire in less than 12 months. If it will expire, the certificate isn't attached.

```
Function AddSignature() As Boolean
    On Error GoTo Error Handler
    Dim sig As Signature
    'Display the dialog box that lets the
    'user select a digital signature.
    'If the user selects a signature, then
    'it is added to the Signatures
    'collection. If the user doesn't, then
    'an error is returned.
    Set sig = ActiveDocument.Signatures.Add
    'Test to make sure that the new Signature object
    'doesn't expire too soon. This expression calculates
    'the number of months until the Signature object expires.
    If DateDiff("m", sig.SignDate, sig.ExpireDate) < 12 Then</pre>
        MsgBox "This certificate will expire in less than 1 year." &
        "Please use a newer certificate."
        AddSignature = False
        sig.Delete
    Else
        AddSignature = True
    End If
    'Commit all signatures in the SignatureSet collection to the dis
    ActiveDocument.Signatures.Commit
    Exit Function
Error Handler:
    AddSignature = False
    MsgBox "Action cancelled."
```

```
End Function
```

Signer Property

Returns a **String** representing the name of the person who attached the digital certificate that corresponds to the **Signature** object to the document. Read-only.

expression.Signer

-

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

The following example prompts the user to select a digital signature with which to sign the active document in Microsoft Word. To use this example, open a document in Word and pass this function the name of a certificate issuer and the name of a certificate signer that match the **Issued By** and **Issued To** fields of a digital certificate in the **Digital Certificates** dialog box. This example will test to make sure that the digital signature that the user selects meets certain criteria, such as not having expired, before the new signature is committed to the disk.

```
Function AddSignature(ByVal strIssuer As String,
    strSigner As String) As Boolean
    On Error GoTo Error_Handler
    Dim sig As Signature
    'Display the dialog box that lets the
    'user select a digital signature.
    'If the user selects a signature, then
    'it is added to the Signatures
    'collection. If the user doesn't, then
    'an error is returned.
    Set sig = ActiveDocument.Signatures.Add
    'Test several properties before commiting the Signature object t
    If sig.Issuer = strIssuer And
        sig.Signer = strSigner And _
        sig.IsCertificateExpired = False And _
        sig.IsCertificateRevoked = False And _
        siq.IsValid = True Then
        MsgBox "Signed"
        AddSignature = True
    'Otherwise, remove the Signature object from the SignatureSet co
    Else
        sig.Delete
        MsgBox "Not signed"
        AddSignature = False
    End If
    'Commit all signatures in the SignatureSet collection to the dis
    ActiveDocument.Signatures.Commit
```

Exit Function

```
Error_Handler:
AddSignature = False
MsgBox "Action cancelled."
End Function
```

Sounds Property

-

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

True if the Office Assistant produces the sounds that correspond to animations. Read/write **Boolean**.

Remarks

The default value is **True**. The **Sounds** property corresponds to the **Make sounds** option under **Use the Office Assistant** on the **Options** tab in the **Office Assistant** dialog box. If a sound card is not installed, this property has no effect.

This example displays and animates the Office Assistant and allows sound.

```
With Assistant

.Visible = True

.On = True

If Not Sounds Then Sounds = True

.Animation = msoAnimationGreeting

End With
```

- <u>Show All</u>

State Property

▶ <u>State property as it applies to the **CommandBarButton** object.</u>

Returns or sets the appearance of a command bar <u>button control</u>. Read/write <u>MsoButtonState</u>.

MsoButtonState can be one of these MsoButtonState constants.

msoButtonDown msoButtonMixed msoButtonUp

expression.State

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

▶ State property as it applies to the **HTMLProject** object.

Returns the current state of an **HTMLProject** object. Read-only **MsoHTMLProjectState**.

MsoHTMLProjectState can be one of these MsoHTMLProjectState constants. msoHTMLProjectStateDocumentLocked msoHTMLProjectStateDocumentProjectUnlocked msoHTMLProjectStateProjectLocked

expression.State

expression Required. An expression that returns an **HTMLProject** object.

• As it applies to the **CommandBarButton** object.

This example creates a command bar named Custom and adds two blank buttons to it. The example then sets the button on the left to **msoButtonUp** and sets the button on the right to **msoButtonDown**.

```
Set myBar = CommandBars _
.Add(Name:="Custom", Position:=msoBarTop, _
Temporary:=True)
With myBar
.Controls.Add Type:=msoControlButton, ID:=1
.Controls.Add Type:=msoControlButton, ID:=2
.Visible = True
End With
Set myControl1 = CommandBars("Custom").Controls(1)
myControl1.State = msoButtonUp
Set myControl2 = CommandBars("Custom").Controls(2)
myControl2.State = msoButtonDown
```

- <u>Show All</u>

Style Property

▶ <u>Style property as it applies to the **CommandBarButton** object.</u>

Returns or sets the way a command bar <u>button control</u> is displayed. Read/write <u>MsoButtonStyle</u>.

MsoButtonStyle can be one of these MsoButtonStyle constants.

msoButtonAutomatic msoButtonCaption msoButtonIcon msoButtonIconAndCaption msoButtonIconAndCaptionBelow msoButtonIconAndWrapCaption msoButtonIconAndWrapCaptionBelow msoButtonWrapCaption

▶ <u>Style property as it applies to the **CommandBarComboBox** object.</u>

Returns or sets the way a command bar <u>combo box control</u> is displayed. Can be either of the following **MsoComboStyle** constants: **msoComboLabel** or **msoComboNormal**. Read/write <u>MsoComboStyle</u>.

MsoComboStyle can be one of these MsoComboStyle constants. msoComboLabel msoComboNormal

This example creates a shortcut menu containing a button control and a combo box control and sets the style of each.

```
Set myBar = CommandBars _
.Add(Name:="Custom1", Position:=msoBarPopup, Temporary:=False)
With myBar
.Controls.Add Type:=msoControlButton, Id:=3
.Controls(1).Style = msoButtonCaption
.Controls.Add Type:=msoControlComboBox
With .Controls(2)
.Style = msoComboLabel
.AddItem "vanilla"
.AddItem "chocolate"
.AddItem "cookie dough"
End With
End With
End With
myBar.ShowPopup
```

Table Property

Returns a **String** that represents the name of the table within the data source file that contains the mail merge records. The returned value may be blank if the table name is unknown or not applicable to the current data source. Read-only.

expression.Table

-

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

This example sets the name of the table if the table name is currently blank.

```
Sub OfficeTest()
Dim appOffice As OfficeDataSourceObject
Set appOffice = Application.OfficeDataSourceObject
appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName
    "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
If appOffice.Table = "" Then
    appOffice.Table = "Employees"
End If
```

End Sub

- <u>Show All</u>

Tag Property

Returns or sets information about the <u>command bar control</u>, such as data that can be used as an argument in procedures, or information that identifies the control. Read/write **String**.

expression.Tag

-

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

Remarks

To avoid duplicate calls of the same class when tiggered with events, define the **Tag** property unique to the events. The following example demonstrates this concept with two modules.

```
Public WithEvents oBtn As CommandBarButton
Private Sub oBtn_click(ByVal ctrl As Office.CommandBarButton, Cancel
    MsgBox "Clicked " & ctrl.Caption
End Sub
Dim oBtns As New Collection
Sub Use_Tag()
    Dim oEvt As CBtnEvent
    Set oBtns = Nothing
    For i = 1 To 5
        Set oEvt = New CBtnEvent
        Set oEvt.oBtn = Application.CommandBars("Worksheet Menu Bar"
        With oEvt.oBtn
            .Caption = "Btn" & i
            .Style = msoButtonCaption
            .Tag = "Hello" & i
        End With
        oBtns.Add oEvt
    Next
```

End Sub

This example sets the tag for the button on the custom command bar to "Spelling Button" and displays the tag in a message box.

CommandBars("Custom").Controls(1).Tag = "Spelling Button"
MsgBox (CommandBars("Custom").Controls(1).Tag)

- <u>Show All</u>

Text Property

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

BalloonLabel or **BalloonCheckbox** object: Returns or sets the text displayed next to a check box or label in the Office Assistant balloon. Read/write **String**.

Balloon object: Returns or sets the text displayed after the heading but before the labels or check boxes in the Office Assistant balloon. Read/write **String**.

CommandBarComboBox object: Returns or sets the text in the display or edit portion of the command bar <u>combo box control</u>. Read/write **String**.

HTMLProjectItem object: Returns or sets the HTML text in the HTML editor. Read/write **String**.

expression.Text

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

Remarks

For the **Balloon**, **BalloonLabel**, and **BalloonCheckbox** objects, you can specify that a particular graphic be displayed by using the following syntax: {*type location sizing_factor*}, where *type* is bmp (bitmap) or wmf (Windows metafile); *location* is the resource ID or the path and file name; and *sizing_factor* denotes the width of the .wmf file (*sizing_factor* is omitted for .bmp files).

The **Balloon** object also supports underlined text and text that has one of the 16 system palette colors applied to it. To display underlined text, use the syntax {*u*} or {*u*l 1}; use {*u*l 0} to turn underlining off. To change the color of text, precede the text string with the character sequence {*cf number*}, where *number* is one of the system color numbers listed in the following table.

System color number	Color
0	Black
1	Dark red
2	Dark green
3	Dark yellow
4	Dark blue
5	Dark magenta
6	Dark cyan
7	Light gray
248	Medium gray
249	Red
250	Green
251	Yellow
252	Blue
253	Magenta
254	Cyan
255	White

If you specify a number other than one of the preceding system color numbers, the text in the Office Assistant balloon is black.

This example creates a new command bar named "Custom" and adds to it a combo box that contains four list items. The example then uses the **Text** property to set Item 3 as the default list item.

```
Set myBar = CommandBars ______.Add(Name:="Custom", Position:=msoBarTop, ______
Temporary:=True)
With myBar
    .Controls.Add Type:=msoControlComboBox, ID:=1
    .Visible = True
End With
Set testComboBox = CommandBars("Custom").Controls(1)
With testComboBox
    .AddItem "Item 1", 1
    .AddItem "Item 1", 1
    .AddItem "Item 2", 2
    .AddItem "Item 3", 3
    .AddItem "Item 4", 4
    .Text = "Item 3"
End With
```

This example creates a new Office Assistant balloon with a heading, text, and three region choices. The example uses the **Text** property to provide balloon-related instructions to the user and a label for each text box.

```
With Assistant.NewBalloon
   .Heading = "Regional Sales Data"
   .Text = "Select a region"
   For i = 1 To 3
        .CheckBoxes(i).Text = "Region " & i
        Next
        .Show
End With
```

This example creates a new Office Assistant balloon that contains underlined heading text, red text, and blue text that is also underlined.

```
With Assistant.NewBalloon
    .Heading = "Underlined {ul 1}Heading{ul 0}"
    .Text = "Some {cf 249}Red{cf 0} text and some " & _
    "underlined {cf 252}{ul 1}Blue{ul 0}{cf 0} text."
    .Show
End With
```

This example creates a new Office Assistant balloon that contains a Windows metafile.

```
With Assistant.NewBalloon
    .Heading = "Underlined {ul 1}Heading{ul 0}"
    .Text ="{WMF ""C:\Favorites\MyPicture.WMF""}"
    .Show
End With
```

TextOrProperty Property

-

Returns or sets the word or phrase to be searched for, in either the body of a file or the file's properties, during the file search. The word or phrase can include the * (asterisk) or ? (question mark) wildcard character. Read/write **String**.

Remarks

Use the question mark wildcard character to match any single character. For example, type **gr?y** to find all files that contain at least one instance of either "gray" or "grey."

Use the asterisk wildcard character to match any number of characters. For example, type **San*** to return all files that contain at least one word that begins with "San."

This example searches the C:\My Documents folder and all of its subfolders and returns all files whose body text or file properties contain any words that begin with "San." The **TextOrProperty** property sets the word to be searched for and limits the search to either the body of the file or the file properties.

```
With Application.FileSearch
    .NewSearch
    .LookIn = "C:\My Documents"
    .SearchSubFolders = True
    .TextOrProperty = "San*"
    .FileType = msoFileTypeAllFiles
End With
```

TipOfDay Property

-

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

True if the Office Assistant displays a special tip each time the Office application is opened. Read/write **Boolean**.

Remarks

The default value is **False**. **TipOfDay** property corresponds to the **Show the Tip of the Day at startup** option under **Show tips about** on the **Options** tab in the **Office Assistant** dialog box.
This example displays the Office Assistant in a specific location and it sets several options before making the Assistant visible.

```
With Assistant
    .On = True
    .Visible = True
    .Left = 400
    .MoveWhenInTheWay = True
    If Not TipOfDay Then TipOfDay = True
    .Animation = msoAnimationGreeting
End With
```

Title Property

Sets or returns the title of a file dialog box displayed using the **FileDialog** object. Read/write **String**.

expression.Title

-

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

The following example displays a File Picker dialog box using the **FileDialog** object and displays each selected file in a message box.

```
Sub main()
    'Declare a variable as a FileDialog object.
    Dim fd As FileDialog
    'Create a FileDialog object as a File Picker dialog box.
    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    'Declare a variable to contain the path
    'of each selected item. Even though the path is a String,
    'the variable must be a Variant because For Each...Next
    'routines only work with Variants and Objects.
    Dim vrtSelectedItem As Variant
    'Use a With...End With block to reference the FileDialog object.
   With fd
        'Change the title of the dialog
        .Title = "Archive"
        'Use the Show method to display the file picker dialog and r
        'If the user presses the action button...
        If .Show = -1 Then
            'Step through each string in the FileDialogSelectedItems
            For Each vrtSelectedItem In .SelectedItems
                'vrtSelectedItem is a String that contains the path
                'You can use any file I/O functions that you want to
                'This example simply displays the path in a message
                MsgBox "Selected item's path: " & vrtSelectedItem
            Next vrtSelectedItem
        'If the user presses Cancel...
        Else
        End If
    End With
    'Set the object variable to Nothing.
    Set fd = Nothing
```

End Sub

TooltipText Property

-

Returns or sets the text displayed in a command bar control's ScreenTip. Read/write **String**.

Remarks

By default, the value of the <u>Caption</u> property is used as the ScreenTip.

This example adds a ScreenTip to the last control on the active menu bar.

```
Set myMenuBar = CommandBars.ActiveMenuBar
Set lastCtrl = myMenuBar _
    .Controls(myMenuBar.Controls.Count)
lastCtrl.BeginGroup = True
lastCtrl.TooltipText = "Click for help on UI feature"
```

- <u>Show All</u>

Top Property

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

• Top property as it applies to the **Assistant** and **CommandBar** objects.

Sets or returns the distance (in points) from the top of the Office Assistant, or from the top edge of the specified <u>command bar</u>, to the top edge of the screen. For docked command bars, this property returns or sets the distance from the command bar to the top of the docking area. Read/write **Long**.

expression.Top

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

Top property as it applies to the CommandBarButton, CommandBarComboBox, CommandBarControl, and CommandBarPopup objects.

Returns the distance (in pixels) from the top edge of the specified <u>command bar</u> <u>control</u> to the top edge of the screen. Read-only **Long**.

expression.Top

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

```
As it applies to the Assistant and CommandBar objects.
```

This example moves the Office Assistant to another coordinate and sets its **Top** property for subsequent appearances.

This example positions the upper-left corner of the floating command bar named Custom 140 pixels from the left edge of the screen and 100 pixels from the top of the screen.

```
Set myBar = CommandBars("Custom")
myBar.Position = msoBarFloating
With myBar
    .Left = 140
    .Top = 100
End With
```

- <u>Show All</u>

Type Property

• Type property as it applies to the **CommandBar** object.

Returns the type of <u>command bar</u>. Read-only <u>MsoBarType</u>.

MsoBarType can be one of these MsoBarType constants. msoBarTypeMenuBar msoBarTypeNormal msoBarTypePopup

expression.Type

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

Type property as it applies to the CommandBarButton, CommandBarComboBox, CommandBarControl, and CommandBarPopup objects.

Returns the type of command bar control. Read-only MsoControlType.

MsoControlType can be one of these MsoControlType constants. msoControlActiveX msoControlButtoCompleteCombo msoControlButtonDropdown msoControlButtonPopup msoControlComboBox msoControlCustom msoControlDropdown msoControlEdit msoControlExpandingGrid msoControlGauge msoControlGenericDropdown msoControlGraphicCombo msoControlGraphicDropdown msoControlGraphicPopup **msoControlGrid** msoControlLabel **msoControlLabelEx msoControlOCXDropdown msoControlPane msoControlPopup** msoControlSpinner msoControlSplitButtonMRUPopup msoControlSplitButtonPopup **msoControlSplitDropdown** msoControlSplitExpandingGrid msoControlWorkPane

expression.**Type**

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

• Type property as it applies to the **SearchScope** object.

Returns a value that corresponds to the type of **SearchScope** object. The type indicates the area in which the **Execute** method of the **FileSearch** object will search for files. Read-only **MsoSearchIn**.

MsoSearchIn can be one of these MsoSearchIn constants. msoSearchInCustom msoSearchInMyComputer msoSearchInMyNetworkPlaces msoSearchInOutlook

expression.Type

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

• Type property as it applies to the **DocumentProperty** object.

Returns or sets the document property type. Read-only for built-in document properties; read/write for custom document properties.

expression.**Type**

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

Remarks

The return value will be a **MsoDocProperties** constant.

MsoDocProperties can be one of these MsoDocProperties constants. msoPropertyTypeBoolean msoPropertyTypeDate msoPropertyTypeFloat msoPropertyTypeNumber msoPropertyTypeString

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

This example finds the first control on the command bar named Custom. Using the **Type** property, the example determines whether the control is a button. If the control is a button, the example copies the face of the **Copy** button (on the **Standard** toolbar) and then pastes it onto the control.

```
Set oldCtrl = CommandBars("Custom").Controls(1)
If oldCtrl.Type = msoControlButton Then
   Set newCtrl = CommandBars.FindControl(Type:= _
        MsoControlButton, ID:= _
        CommandBars("Standard").Controls("Copy").ID)
   NewCtrl.CopyFace
   OldCtrl.PasteFace
End If
```

• <u>As it applies to the **DocumentProperty** object</u>.

This example displays the name, type, and value of a document property. You must pass a valid **DocumentProperty** object to the procedure.

```
Sub DisplayPropertyInfo(dp As DocumentProperty)
    MsgBox "value = " & dp.Value & Chr(13) & _
        "type = " & dp.Type & Chr(13) & _
        "name = " & dp.Name
End Sub
```

- <u>Show All</u>

Value Property

▶ <u>Value property as it applies to the **DocumentProperty** object.</u>

Returns or sets the value of a document property. Read/write Variant.

expression.Value

-

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

Remarks

If the container application doesn't define a value for one of the built-in document properties, reading the **Value** property for that document property causes an error.

• <u>Value property as it applies to the **PropertyTest** object.</u>

Returns the value of a property test for a file search. Read-only **Variant**.

expression.Value

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

```
• <u>As it applies to the DocumentProperty object</u>.
```

This example displays the name, type, and value of a document property. For the example to work, dp must be a valid **DocumentProperty** object.

```
Sub DisplayPropertyInfo(dp As DocumentProperty)
    MsgBox "value = " & dp.Value & Chr(13) & _
        "type = " & dp.Type & Chr(13) & _
        "name = " & dp.Name
End Sub
```

• As it applies to the **PropertyTest** object.

This example displays the value of the search criteria (if it exists) in a message box. If the second value doesn't exist, the example displays another message.

Visible Property

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

True if the specified object is visible. Read/write Boolean.

expression.Visible

-

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

Remarks

The **Visible** property for newly created custom command bars is **False** by default.

The **Enabled** property for a command bar must be set to **True** before the visible property is set to **True**.

This example steps through the collection of command bars to find the Forms command bar. If the Forms command bar is found, the example makes it visible and protects its docking state.

```
foundFlag = False
For Each cmdbar In CommandBars
    If cmdbar.Name = "Forms" Then
        cmdbar.Protection = msoBarNoChangeDock
        cmdbar.Visible = True
        foundFlag = True
        End If
Next
If Not foundFlag Then
        MsgBox "'Forms'command bar is not in the collection."
End If
```

This example makes the Office Assistant visible and sets its animation.

```
With Application.Assistant
   .Visible = True
   .Sounds = True
   .Animation = msoAnimationBeginSpeaking
End With
```

- <u>Show All</u>

Width Property

Returns or sets the width (in pixels) of the specified <u>command bar</u> or <u>command</u> <u>bar control</u>. Read/write **Integer**.

expression.Width

-

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

This example adds a custom control to the command bar named Custom. The example sets the height of the custom control to twice the height of the command bar and sets its width to 50 pixels. Notice how the command bar automatically resizes itself to accommodate the control.

```
Set myBar = CommandBars("Custom")
barHeight = myBar.Height
Set myControl = myBar.Controls _
    .Add(Type:=msoControlButton, _
    Id:= CommandBars("Standard").Controls("Save").Id, _
    Temporary:=True)
With myControl
    .Height = barHeight * 2
    .Width = 50
End With
myBar.Visible = True
```

Change Event

_

Occurs when the end user changes the selection in a command bar combo box.

Private Sub CommandBarComboBox_Change

(ByVal Ctrl As CommandBarComboBox)

Remarks

The **Change** event is recognized by the **CommandBarComboBox** object. To return the Change event for a particular **CommandBarComboBox** control, use the **WithEvents** keyword to declare a variable, and then set the variable to the **CommandBarComboBox** control. When the **Change** event is triggered, it executes the macro or code that you specified with the **OnAction** property of the control.

The following example creates a command bar with a **CommandBarComboBox** control containing four selections. The combo box responds to user interaction through the CommandBarComboBox_Change event.

```
Private ctlComboBoxHandler As New ComboBoxHandler
Sub AddComboBox()
```

```
Set HostApp = Application
Dim newBar As Office.CommandBar
Set newBar = HostApp.CommandBars.Add(Name:="Test CommandBar", Te
Dim newCombo As Office.CommandBarComboBox
Set newCombo = newBar.Controls.Add(msoControlComboBox)
With newCombo
    .AddItem "First Class", 1
    .AddItem "Business Class", 2
    .AddItem "Coach Class", 3
    .AddItem "Standby", 4
    .DropDownLines = 5
    .DropDownWidth = 75
    .ListHeaderCount = 0
End With
ctlComboBoxHandler.SyncBox newCombo
newBar.Visible = True
```

End Sub

The preceding example relies on the following code, which is stored in a class module in the VBA project.

```
Private WithEvents ComboBoxEvent As Office.CommandBarComboBox
Public Sub SyncBox(box As Office.CommandBarComboBox)
    Set ComboBoxEvent = box
    If Not box Is Nothing Then
        MsgBox "Synced " & box.Caption & " ComboBox events."
    End If
End Sub
Private Sub Class_Terminate()
    Set ComboBoxEvent = Nothing
```

End Sub

```
Private Sub ComboBoxEvent_Change(ByVal Ctrl As Office.CommandBarComb
    Dim stComboText As String
    stComboText = Ctrl.Text
        Select Case stComboText
        Case "First Class"
            FirstClass
        Case "Business Class"
            BusinessClass
        Case "Coach Class"
            CoachClass
        Case "Standby"
            Standby
    End Select
End Sub
Private Sub FirstClass()
    MsqBox "You selected First Class reservations"
End Sub
Private Sub BusinessClass()
    MsqBox "You selected Business Class reservations"
End Sub
Private Sub CoachClass()
    MsqBox "You selected Coach Class reservations"
End Sub
Private Sub Standby()
    MsgBox "You chose to fly standby"
End Sub
```

Click Event

-

Occurs when the user clicks a **CommandBarButton** object.

Private Sub CommandBarButton_Click

(ByVal Ctrl As CommandBarButton,

ByVal CancelDefault As Boolean)

The syntax for the Click event includes the two arguments described in the following table.

Argument	Description
	Required CommandBarButton.
Ctrl	Denotes the CommandBarButton
	control that initiated the event.
	Required Boolean . False if the default
	behavior associated with the
CancelDefault	CommandBarButton control occurs,
	unless it's canceled by another process
	or add-in.

Remarks

The Click event is recognized by the **CommandBarButton** object. To return the Click event for a particular **CommandBarButton** control, use the **WithEvents** keyword to declare a variable, and then set the variable to the control.

The following example creates a new command bar button on the File menu of the host application that enables the user to save a workbook as a commaseparated value file. (This example works in all applications, but the context of saving as CSV is applicable to Microsoft Excel.)

```
Private HostApp As Object
Sub createAndSynch()
    Dim iIndex As Integer
    Dim iCount As Integer
    Dim fBtnExists As Boolean
    Dim obCmdBtn As Object
    Dim btnSaveAsCSVHandler as new Class1
    On Error GoTo errHandler
    Set HostApp = Application
    Dim barHelp As Office.CommandBar
    Set barHelp = Application.CommandBars("File")
    fBtnExists = False
    iCount = barHelp.Controls.Count
    For iIndex = 1 To iCount
        If barHelp.Controls(iIndex).Caption = "Save As CSV (Comma De
    Next
    Dim btnSaveAsCSV As Office.CommandBarButton
    If fBtnExists Then
        Set btnSaveAsCSV = barHelp.Controls("Save As CSV (Comma Deli
    Else
        Set btnSaveAsCSV = barHelp.Controls.Add(msoControlButton)
        btnSaveAsCSV.Caption = "Save As CSV (Comma Delimited)"
    End If
    btnSaveAsCSV.Tag = "btn1"
    btnSaveAsCSVHandler.SyncButton btnSaveAsCSV
    Exit Sub
errHandler:
    ' Insert error handling code here
End Sub
```

EnvelopeHide Event

Occurs when the user interface (UI) that corresponds to the **MsoEnvelope** object is hidden.

Private Sub object_EnvelopHide()

-

object A variable which references an object of type **MsoEnvelope** declared with events in a class module.

The following example sets up event-handling routines for the **MsoEnvelope** object.

```
Public WithEvents env As MsoEnvelope
Private Sub Class_Initialize()
   Set env = Application.ActiveDocument.MailEnvelope
End Sub
Private Sub env_EnvelopeShow()
   MsgBox "The MsoEnvelope UI is showing."
End Sub
Private Sub env_EnvelopeHide()
   MsgBox "The MsoEnvelope UI is hidden."
End Sub
```
EnvelopeShow Event

Occurs when the user interface (UI) that corresponds to the **MsoEnvelope** object is displayed.

Private Sub object_EnvelopShow()

-

object A variable which references an object of type **MsoEnvelope** declared with events in a class module.

Example

The following example sets up event-handling routines for the **MsoEnvelope** object.

```
Public WithEvents env As MsoEnvelope
Private Sub Class_Initialize()
   Set env = Application.ActiveDocument.MailEnvelope
End Sub
Private Sub env_EnvelopeShow()
   MsgBox "The MsoEnvelope UI is showing."
End Sub
Private Sub env_EnvelopeHide()
   MsgBox "The MsoEnvelope UI is hidden."
End Sub
```

OnUpdate Event

-

Occurs when any change is made to a command bar.

Private Sub CommandBars_OnUpdate()

Remarks

The OnUpdate event is recognized by the **CommandBar** object and all command bar controls. The event is triggered by any change to a command bar or command bar control or any change to the state of a command bar or command bar control. These changes can occur due to a text or cell selection, for example. Since a large number of OnUpdate events can occur during normal usage, developers should exercise caution when using this event. It is strongly recommended that this event be used primarily for checking that a custom command bar has been added or removed by a **COMAddIn**.

Returning an Object from a Collection

The **Item** property returns a single object from a collection. The following example sets the cmdbar variable to a **CommandBar** object that represents the first command bar in the **CommandBars** collection.

```
Set cmdbar = CommandBars.Item(1)
```

The **Item** property is the default property for most collections, so you can write the same statement more concisely by omitting the **Item** keyword.

```
Set cmdbar = CommandBars(1)
```

For more information about a specific collection, see the Help topic for the collection or the **Item** property for the collection.