

ActivateWizard Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthActivateWizardC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthActivateWizardX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthActivateWizardA"}
{ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthActivateWizardS"}

Resumes or suspends an Office Assistant wizard session.

Note This method is used as part of the process begun with the **StartWizard** method.

Syntax

expression.**ActivateWizard**(*WizardID*, *Act*, *Animation*)

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

WizardID Required **Long**. A number that uniquely identifies the Office Assistant wizard session, as returned by the **StartWizard** method.

Act Required **Variant**. Specifies the way the Office Assistant wizard session changes. Can be one of the following constants: **msoWizardActTypes**: **msoWizardActActive**, **msoWizardActInactive**, **msoWizardActResume**, or **msoWizardActSuspend**.

Animation Optional **Variant**. The type of animation that the Office Assistant performs when it resumes or suspends the wizard session.

ActivateWizard Method Example

This example suspends the Office Assistant wizard session begun with the **StartWizard** method. The variable `helpForWiz` was set to the return value of the **StartWizard** method.

```
Assistant.ActivateWizard WizardID:=helpForWiz, _  
    Act:=msoWizardActSuspend, Animation:=msoAnimationGoodbye
```

Close Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthCloseC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthCloseX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthCloseA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthCloseS"}

Closes the active modeless balloon.

Syntax

expression.**Close**

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

Close Method Example

This example displays a balloon that contains instructions for selecting a printer. After the user clicks the OK button on the balloon, the `ProcessPrinter` procedure is run and the balloon is closed.

```
Sub shar()  
Set bln = Assistant.NewBalloon  
With bln  
    .Heading = "Instructions for Choosing a Printer."  
    .Text = "Click OK when you've chosen a printer."  
    .Labels(1).Text = "Choose File, 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  
End Sub  
  
Sub ProcessPrinter(bln As Balloon, ibtn As Long, _  
    iPriv As Long)  
    Assistant.Animation = msoAnimationSearching  
    ' Insert printer-specific code  
    bln.Close  
End Sub
```

EndWizard Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthEndWizardC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthEndWizardX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthEndWizardA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthEndWizardS"}

Closes the Office Assistant window and releases the variable that uniquely identifies the wizard session.

Note You use this method to complete the process begun with the [StartWizard](#) method.

Syntax

expression.**EndWizard**(*WizardID*, *varfSuccess*, *Animation*)

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

WizardID Required **Long**. A number that uniquely identifies the Office Assistant wizard session, as returned by the **StartWizard** method.

varfSuccess Required **Boolean**. When the method returns, this argument is **True** if the user completes the wizard successfully.

Animation Optional **Variant**. The type of animation that the Office Assistant performs when the wizard session ends.

EndWizard Method Example

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

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

StartWizard Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthStartWizardC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthStartWizardX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthStartWizardA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthStartWizardS"}

Starts the Office Assistant as part of a process to provide additional explanation for existing custom wizard panels. Returns a number that uniquely identifies the Office Assistant wizard session.

Syntax

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

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

On Required **Boolean**. The wizard's local state. When the method returns, this argument is **True** if the user requests help for the wizard from the standard balloon that asks the user whether he or she wants help.

Callback Required **String**. The name of the callback procedure to be run.

PrivateX Required **Long**. The unique identifier of the Office Assistant balloon panel that initiates the callback procedure.

Animation Optional **Variant**. The type of animation that the Office Assistant performs when it appears. The default value is **msoAnimationGetWizardy**.

CustomTeaser Optional **Variant**. **True** to have the standard balloon that asks the user whether he or she wants help be replaced with a custom balloon.

Top, Left, Bottom, Right Optional **Variant**. The position of the corners (in points and relative to the screen) of the custom wizard panel that the Office Assistant will avoid when the user starts it or switches to it.

Remarks

It isn't necessary to use the **Visible** property to display the Office Assistant if you use the **StartWizard** method.

StartWizard Method Example

This example starts the Office Assistant as part of a process to provide additional explanation for existing custom wizards. If the user clicks the selection in the balloon that requests additional help, the callback procedure will be run. The variable `helpForWiz` is set to the return value of the **StartWizard** method.

```
helpForWiz = Assistant.StartWizard(on:=True, _  
    PrivateX:=23, Callback:="myCallback", _  
    Animation:=msoAnimationGetAttentionMajor, _  
    CustomTeaser:=False, Top:=100, Left:=200, _  
    Bottom:=50, Right:=200)
```


AssistWithAlerts Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproAssistWithAlertsC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproAssistWithAlertsX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproAssistWithAlertsA"}
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproAssistWithAlertsS"}

True if application alerts are delivered by the Office Assistant balloon whenever the Office Assistant is visible. Read/write **Boolean**.

Remarks

The **AssistWithAlerts** property corresponds to the **Display alerts** option under **Assistant capabilities** on the **Options** tab in the **Office Assistant** dialog box.

If this property is **False**, the application displays alerts in dialog boxes without the Office Assistant.

AssistWithAlerts Property Example

This example sets the Office Assistant to be displayed whenever an application alert is generated.

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

AssistWithHelp Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproAssistWithHelpC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproAssistWithHelpX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproAssistWithHelpA"}
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproAssistWithHelpS"}

True if the Office Assistant appears whenever 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 **Assistant capabilities** 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.

AssistWithHelp Property Example

This example displays the Office Assistant whenever the user presses the F1 key to display Help.

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

Callback Property

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproCallbackC"} {ewc HLP95EN.DLL, DYNALINK,  
"Example": "ofproCallbackX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproCallbackA"} {ewc  
HLP95EN.DLL, DYNALINK, "Specifics": "ofproCallbacks"}
```

Sets the name of the procedure to be run whenever a modeless balloon is displayed. Read/write **String**.

Note If you specify a macro that exists behind an Excel worksheet, you must include the worksheet in the reference, as in "Sheet1.myCallback" instead of "myCallback".

Callback Property Example

This example displays a balloon that contains a selection of three printers. After the user clicks the **OK** button on the balloon, the `ProcessPrinter` procedure is run and the balloon is closed.

```
Sub shar()  
Set bln = Assistant.NewBalloon  
With bln  
    .Heading = "Select a Printer."  
    .Text = "Click OK when you've selected a printer."  
    .Labels(1).Text = "Network Printer"  
    .Labels(2).Text = "Local Printer"  
    .Labels(3).Text = "Local Color Printer"  
    .BalloonType = msoBalloonTypeButtons  
    .Mode = msoModeModeless  
    .Callback = "ProcessPrinter"  
    .Button = msoButtonSetOK  
    .Show  
End With  
End Sub
```

(Every procedure specified in the **Callback** property is passed three arguments: the balloon that activated the procedure, the return value of the button the user pressed, and an integer that uniquely identifies the balloon that called the procedure.)

```
Sub ProcessPrinter(bln As Balloon, ibtn As Long, _  
    iPriv As Long)  
    Assistant.Animation = msoAnimationPrinting  
    Select Case ibtn  
    Case 1  
        ' Insert printer-specific code  
        bln.Close  
    Case 2  
        ' Insert printer-specific code  
        bln.Close  
    Case 3  
        ' Insert printer-specific code  
        bln.Close  
    End Select  
End Sub
```

Private Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproPrivateC"} {ewc HLP95EN.DLL, DYNALINK,
"Example":"ofproPrivateX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproPrivateA"} {ewc
HLP95EN.DLL, DYNALINK, "Specifics":"ofproPrivateS"}

Returns or sets an integer that uniquely identifies the Office Assistant balloon that initiates the callback procedure. Read/write **Long**.

Remarks

It may be helpful to use this property if you use the same callback procedure for a variety of circumstances.

Private Property Example

This example identifies the Office Assistant balloon by setting the **Private** property to 129. The callback procedure, `myCallback`, is used in a variety of circumstances, and the integer 129 will be used to identify this circumstance.

```
With myBalloon
    .Heading = "Select a region"
    .Private = 129
    .Callback = "myCallback"
    .Show
End With
```


SetAvoidRectangle Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthSetAvoidRectangleC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthSetAvoidRectangleX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthSetAvoidRectangleA"}
{ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthSetAvoidRectangleS"}

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

Syntax

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 be excluded from when it's displayed.

Remarks

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

SetAvoidRectangle Method Example

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 (measured in pixels).

```
With myBalloon
    .SetAvoidRectangle 300, 250, 700, 500
    .Show
End With
```

ResetTips Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthResetTipsC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthResetTipsX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthResetTipsA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthResetTipsS"}

Resets the application tips that will appear in the Office Assistant balloon.

Syntax

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.

ResetTips Method Example

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

MoveWhenInTheWay Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproMoveWhenInTheWayC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproMoveWhenInTheWayX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproMoveWhenInTheWayA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproMoveWhenInTheWayS"}

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**.

Remarks

The **MoveWhenInTheWay** property corresponds to the **Move when in the way** option under **Assistant capabilities** on the **Options** tab in the **Office Assistant** dialog box.

MoveWhenInTheWay Property Example

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
```

NewBalloon Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproNewBalloonC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproNewBalloonX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproNewBalloonA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproNewBalloonS"}

Creates a new Office Assistant balloon. Returns a **Balloon** object.

NewBalloon Property Example

This example creates a balloon with a heading, text, and three region choices, and then displays it.

```
With Assistant.NewBalloon
    .Button = msoButtonSetOK
    .Heading = "Regional Sales Data"
    .Text = "Select a region"
    For i = 1 To 3
        .CheckBoxes(i).Text = "Region " & i
    Next
    .Show
End With
```


Reduced Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproReducedC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproReducedX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproReducedA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproReducedS"}

True if the Office Assistant window appears in its smaller size. Read/write **Boolean**.

Reduced Property Example

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
```

SearchWhenProgramming Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproSearchWhenProgrammingC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproSearchWhenProgrammingX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproSearchWhenProgrammingA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproSearchWhenProgrammingS"}

True if the Office Assistant displays application help as well as programming help while the user is working in Visual Basic. Read/write **Boolean**.

Remarks

The **SearchWhenProgramming** property corresponds to the **Search for both product and programming help when programming** option under **Assistant capabilities** on the **Options** tab in the **Office Assistant** dialog box.

SearchWhenProgramming Property Example

This example enables you to search both application and programming help while you're working in Visual Basic.

```
Assistant.SearchWhenProgramming = True
```

Show Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthShowC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthShowX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthShowA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthShowS"}

Balloon object: Displays the specified balloon object. Returns an **msoBalloonType** constant that indicates which balloon the user clicks.

FileFind object: Displays the **Find File** dialog and the current search criteria. Macintosh only.

Syntax

expression.**Show**

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

Remarks

For the **Balloon** object, you can use the return value of the **Show** method to display a user's button selection. The **Show** method returns one of the following **msoBalloonType** constants:

msoBalloonButtonAbort
msoBalloonButtonCancel
msoBalloonButtonIgnore
msoBalloonButtonNo
msoBalloonButtonOK
msoBalloonButtonRetry
msoBalloonButtonSnooze
msoBalloonButtonYes

msoBalloonButtonBack
msoBalloonButtonClose
msoBalloonButtonNext
msoBalloonButtonNull
msoBalloonButtonOptions
msoBalloonButtonSearch
msoBalloonButtonTips
msoBalloonButtonYesToAll

Show Method Example

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, corresponding to the label the user clicks. In the example, a simple 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
```

Sounds Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproSoundsC"} {ewc HLP95EN.DLL, DYNALINK,
"Example":"ofproSoundsX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproSoundsA"} {ewc
HLP95EN.DLL, DYNALINK, "Specifics":"ofproSoundsS"}

True if the Office Assistant produces the sounds that correspond to animations. Read/write **Boolean**.

Remarks

If a sound card is not installed, this property has no effect.

Sounds Property Example

This example displays the Office Assistant, animates it, and generates sound.

```
With Assistant
    .Visible = True
    .Sounds = True
    .Animation = msoAnimationBeginSpeaking
End With
```


TipOfDay Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproTipOfDayC"} {ewc HLP95EN.DLL, DYNALINK,
"Example":"ofproTipOfDayX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproTipOfDayA"} {ewc
HLP95EN.DLL, DYNALINK, "Specifics":"ofproTipOfDayS"}

True if the Office Assistant displays a special tip each time the Office application is opened.

Read/write **Boolean**.

Remarks

The **TipOfDay** property corresponds to the **Show the Tip of the Day at startup** option under **Other tip options** on the **Options** tab in the **Office Assistant** dialog box.

TipOfDay Property Example

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
```

Animation Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproAnimationC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproAnimationX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproAnimationA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproAnimationS"}

Returns or sets the animation action for the Office Assistant. When this property is applied to the **Assistant** object, the Assistant is animated immediately (if the Assistant is visible). When this property is applied to the **Balloon** object, the Assistant is animated only while the balloon is displayed. Read/write **Long**.

Can be one of the following **MsoAnimationType** constants:

msoAnimationAppear	msoAnimationIdle
msoAnimationBeginSpeaking	msoAnimationListensToComputer
msoAnimationCharacterSuccessMajor	msoAnimationLookDown
msoAnimationCheckingSomething	msoAnimationLookDownLeft
msoAnimationDisappear	msoAnimationLookDownRight
msoAnimationEmptyTrash	msoAnimationLookLeft
msoAnimationGestureDown	msoAnimationLookRight
msoAnimationGestureLeft	msoAnimationLookUp
msoAnimationGestureRight	msoAnimationLookUpLeft
msoAnimationGestureUp	msoAnimationLookUpRight
msoAnimationGetArtsy	msoAnimationPrinting
msoAnimationGetAttentionMajor	msoAnimationSaving
msoAnimationGetAttentionMinor	msoAnimationSearching
msoAnimationGetTechy	msoAnimationSendingMail
msoAnimationGetWizardy	msoAnimationThinking
msoAnimationGoodbye	msoAnimationWorkingAtSomething
msoAnimationGreeting	msoAnimationWritingNotingSomething

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 any obvious animation. However, all **MsoAnimationType** constants are valid for all Assistants.

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	msoAnimationSearching
msoAnimationGetArtsy	msoAnimationThinking
msoAnimationGetTechy	msoAnimationWorkingAtSomething
msoAnimationSaving	msoAnimationWritingNotingSomething

Animation Property Example

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

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

FileName Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproFileNameC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofproFileNameX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproFileNameA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproFileNameS"}

Assistant object: Returns or sets the name of the file 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**.

Remarks

In Windows, the file name extension for Assistant files is .act, and the files are installed in the \Program Files\Microsoft Office\Office\Actors folder in Windows 95, in the \Windows\MsApps\Actors folder in Windows NT, and in the Microsoft:Assistants folder on the Macintosh.

The following file names correspond to the Assistants in Office 97.

Character	File name
Office Logo	Logo.act
PowerPup	Powerpup.act
The Genius	Genius.act
Hoverbot	Hoverbot.act
Scribble	Scribble.act
The Dot	Dot.act
Clippit	Clippit.act
Mother Nature	MNature.act
Will	Will.act

FileName Property Example

This example uses the **FileName** property to customize balloon text for specific Office Assistants. The variable `btext` can be used for the value of the **Text** property in a Print Wizard balloon.

```
Select Case Assistant.FileName
Case "will.act"
    btext = "To Print, or Not to Print. That is the question."
Case "genius.act"
    btext = "The sum of the checkboxes equals the document to print."
Case Else
    btext = "Choose the document you want to print."
End Select
```

This example searches for all files located in the My Documents folder that begin with "cmd." 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
```

BalloonType Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproBalloonTypeC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofproBalloonTypeX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproBalloonTypeA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproBalloonTypeS"}

Returns or sets the type of balloon the Office Assistant uses. Can be one of the following **MsoBalloonType** constants: **msoBalloonTypeButtons**, **msoBalloonTypeBullets**, or **msoBalloonTypeNumbers**. When you create a new balloon with the **NewBalloon** method, this property is initially set to **msoBalloonTypeButtons**. Read/write **Long**.

BalloonType Property Example

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
```


Button Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproButtonC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofproButtonX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproButtonA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproButtonS"}

Returns or sets the type of button displayed at the bottom of the Office Assistant balloon. When you create a new balloon with the **NewBalloon** method, this property is initially set to **msoButtonSetOK**.
Read/write **Long**.

Can be one of the following **MsoButtonSetType** constants:

msoButtonSetAbortRetryIgnore	msoButtonSetOkCancel
msoButtonSetBackClose	msoButtonSetRetryCancel
msoButtonSetBackNextClose	msoButtonSetSearchClose
msoButtonSetBackNextSnooze	msoButtonSetTipsOptionsClose
msoButtonSetCancel	msoButtonSetYesAllNoCancel
msoButtonSetNextClose	msoButtonSetYesNoCancel
msoButtonSetNone	msoButtonSetYesNo
msoButtonSetOK	

Button Property Example

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 a region"
    For i = 1 To 3
        .CheckBoxes(i).Text = "Region " & i
    Next
    .Button = msoButtonSetOkCancel
    .Show
End With
```

Checkboxes Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproCheckboxesC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofproCheckboxesX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproCheckboxesA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproCheckboxesS"}

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 [Returning an Object from a Collection](#).

Checkboxes Property Example

This example creates a balloon with a heading, text, and three region choices. When the user selects a check box and then clicks **OK** in the balloon, the appropriate procedure is run.

```
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
Select Case True
    Case .CheckBoxes(1).Checked
        runregion1
    Case .CheckBoxes(2).Checked
        runregion2
    Case .CheckBoxes(3).Checked
        runregion3
End Select
End With
```

Checked Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproCheckedC"} {ewc HLP95EN.DLL, DYNALINK,
"Example":"ofproCheckedX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproCheckedA"} {ewc
HLP95EN.DLL, DYNALINK, "Specifics":"ofproCheckedS"}

True if the specified check box in the Office Assistant balloon is selected. Read/write **Boolean**.

Checked Property Example

This example creates a balloon with a heading, text, and three region choices. When the user selects a check box and then clicks **OK** in the balloon, the appropriate procedure is run.

```
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
Select Case True
    Case .CheckBoxes(1).Checked
        runregion1
    Case .CheckBoxes(2).Checked
        runregion2
    Case .CheckBoxes(3).Checked
        runregion3
End Select
End With
```

FeatureTips Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproFeatureTipsC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproFeatureTipsX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproFeatureTipsA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproFeatureTipsS"}

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.

FeatureTips Property Example

This example sets the Office Assistant to provide information about using application features more effectively.

```
Assistant.FeatureTips = True
```


GuessHelp Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproGuessHelpC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproGuessHelpX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproGuessHelpA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproGuessHelpS"}

True if the Office Assistant displays a list of relevant Help topics based on the context immediately before the user clicks the Assistant window or presses F1. Read/write **Boolean**.

Remarks

The **GuessHelp** property corresponds to the **Guess help topics** option under **Assistant capabilities** on the **Options** tab in the **Office Assistant** dialog box.

GuessHelp Property Example

This example sets the Office Assistant to guess at Help topics.

```
Assistant.GuessHelp = True
```

Heading Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproHeadingC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproHeadingX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproHeadingA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproHeadingS"}

Returns or sets the heading that appears in the Office Assistant balloon. Read/write **String**.

Remarks

You can specify a graphic to display by using the following syntax: *{type location sizing_factor}*, where *type* is bmp (bitmap), wmf (Windows metafile), or pict (Macintosh PICT file); *location* is the resource id or the path and file name; and *sizing_factor* specifies the width of the wmf or pict (omitted for bmp).

Heading Property Example

This example creates a balloon with a heading, text, and three region choices, and then displays it.

```
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
```

Help Method

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofmthHelpC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofmthHelpX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofmthHelpA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofmthHelpS"}

Displays the Office Assistant and the built-in Assistant balloon.

Syntax

expression.**Help**

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

Help Method Example

This example displays the built-in Assistant balloon when the user selects the "I need more information" checkbox.

```
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
```

AssistWithWizards Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproAssistWithWizardsC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproAssistWithWizardsX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproAssistWithWizardsA"}
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproAssistWithWizardsS"}

True if the Office Assistant provides Help information about wizards. Read/write **Boolean**.

Remarks

The **AssistWithWizards** property corresponds to the **Help with wizards** option under **Assistant capabilities** on the **Options** tab in the **Office Assistant** dialog box.

AssistWithWizards Property Example

This example sets the Office Assistant to provide Help information about wizards.

```
Assistant.AssistWithWizards = True
```


HighPriorityTips Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproHighPriorityTipsC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproHighPriorityTipsX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproHighPriorityTipsA"}
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproHighPriorityTipsS"}

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 **Other tip options** on the **Options** tab in the **Office Assistant** dialog box.

HighPriorityTips Property Example

This example sets the Office Assistant to display high-priority tips.

```
Assistant.HighPriorityTips = True
```

Icon Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofprolconC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofprolconX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofprolconA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofprolconS"}

Returns or sets the type of icon that appears in the upper-left portion of the Office Assistant balloon. Can be one of the following **MsolconType** constants: **msolconAlert**, **msolconNone**, or **msolconTip**. Read/write **String**.

Icon Property Example

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
```

KeyboardShortcutTips Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproKeyboardShortcutTipsC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproKeyboardShortcutTipsX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproKeyboardShortcutTipsA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproKeyboardShortcutTipsS"}

True if the Office Assistant provides Help information about keyboard shortcuts. Read/write **Boolean**.

Remarks

The **KeyboardShortcutTips** property corresponds to the **Keyboard shortcuts** option under **Show tips about** on the **Options** tab in the **Office Assistant** dialog box.

KeyboardShortcutTips Property Example

This example sets the Office Assistant to provide Help information about keyboard shortcuts.

```
Assistant.KeyboardShortcutTips = True
```

Labels Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproLabelsC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproLabelsX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproLabelsA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproLabelsS"}

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 [Returning an Object from a Collection](#).

Labels Property Example

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, corresponding to the label the user clicks. In the example, a simple 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
```


BalloonError Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproBalloonErrorC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofproBalloonErrorX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproBalloonErrorA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproBalloonErrorS"}

Returns a value that indicates the last recorded balloon error. Read-only **String**.

Can be one of the following **MsoBalloonErrorType** constants.

Constant	Description
msoBalloonErrorBadPictureRef	The balloon contains a bitmap that couldn't be displayed because the file doesn't exist or because the bitmap isn't a valid .BMP or .WMF file (or PICT file on the Macintosh).
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 cannot be dismissed.
msoBalloonErrorButtonModeless	The balloon you attempted to display is modeless, contains no buttons, and has no procedure assigned to the Callback property. The balloon won't be shown because a procedure is required to evaluate the button clicked in the balloon.
msoBalloonErrorNone	The property succeeded in displaying the balloon; no error was encountered.
msoBalloonErrorBadCharacter	The balloon contains an ASCII control character other than CR or LF and greater than 32.
msoBalloonErrorOutOfMemory	The balloon won't appear because there is insufficient memory.
msoBalloonErrorTooBig	The balloon is too big to appear on the screen.
msoBalloonErrorOther	The balloon won't appear because some other error occurred, such as another modal balloon is already active.

BalloonError Property Example

This example creates a balloon that generates an error. The error is caused because the balloon is created without a way to dismiss it: the button type was set to **msoButtonSetNone** and the default balloon mode is **msoModeModal**, resulting in a buttonless 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
```

Mode Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproModeC"} {ewc HLP95EN.DLL, DYNALINK,
"Example":"ofproModeX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproModeA"} {ewc
HLP95EN.DLL, DYNALINK, "Specifics":"ofproModeS"}

Returns or sets the type of balloon displayed. Can be one of the following **MsoModeType** constants: **msoModeAutoDown**, **msoModeModal**, or **msoModeModeless**. When you create a new balloon with the **NewBalloon** method, this property is initially set to **msoModeModal**. Read/write **Long**.

Remarks

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 he or she can return to working in the application. If the property is set to **msoModeAutoDown**, the balloon is instantly dismissed when the user clicks anywhere on the screen.

If the **Mode** property for a balloon is set to **msoModeModeless**, a value for the **Callback** property is required. The **Close** method can only be used if the property is set to **msoModeModeless**.

Mode Property Example

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.

```
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

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproMouseTipsC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproMouseTipsX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproMouseTipsA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproMouseTipsS"}

True if the Office Assistant provides suggestions for using the mouse more effectively. Read/write.

Boolean.

Remarks

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.

MouseTips Property Example

This example sets the Office Assistant to provide suggestions for using the mouse more effectively.

```
Assistant.MouseTips = True
```

Assistant Object

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofobjCommandBarC "} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofobjAssistantX":1} {ewc HLP95EN.DLL, DYNALINK, "Properties":"ofobjAssistantP "} {ewc HLP95EN.DLL, DYNALINK, "Methods":"ofobjAssistantM "} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofobjAssistantS" }
```

Assistant

L

Balloon

L

BalloonCheckboxes (BalloonCheckbox)

L

BalloonLabels (BalloonLabel)

Represents the Microsoft Office Assistant.

Using the Assistant Object

Use the **Assistant** property to return the **Assistant** object. There's no collection for the **Assistant** object; only one **Assistant** object can be active at a time. Use the **Visible** property to display the Assistant.

Remarks

The default Assistant is Clippit. To select a different Assistant programatically, use the **FileName** property.

The following example displays a previously selected Assistant and animates it with the associated sound. If your computer doesn't have a sound card installed, this example won't generate an error, but the sound won't be heard.

```
With Assistant
    .Visible = True
    .Sounds = True
    .Animation = msoAnimationBeginSpeaking
End With
```

Balloon Object

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofobjCommandBarC "} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofobjBalloonX": 1} {ewc HLP95EN.DLL, DYNALINK, "Properties": "ofobjBalloonP "} {ewc HLP95EN.DLL, DYNALINK, "Methods": "ofobjBalloonM "} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofobjBalloonS "}
```

Assistant

L

Balloon

L

BalloonCheckboxes (BalloonCheckbox)

L

BalloonLabels (BalloonLabel)

Represents the balloon in which the Office Assistant displays headings and text 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's no collection for the **Balloon** object; only one balloon can be visible at a time. However, it's possible to define several balloons and call any one of them when needed. For more information, see "Defining and Reusing Balloons" later in this topic.

Use the **Show** method to make the specified balloon visible. Use the **Callback** property to run procedures based on selections from modeless balloons (balloons that remain visible while a user works in the application). Use the **Close** 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 = msoButtonSetOkCancel
    .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 balloons by assigning them to object variables and calling them when needed throughout your procedure. This example defines `balloon1`, `balloon2`, and `balloon3` as separate balloons, and it displays the balloons at various points in the procedure.

```
Set balloon1 = Assistant.NewBalloon
balloon1.Heading = "First balloon"
```



```
Set balloon2 = Assistant.NewBalloon
balloon2.Heading = "Second balloon"
Set balloon3 = Assistant.NewBalloon
balloon3.Heading = "Third balloon"
balloon1.Show
balloon3.Show
balloon2.Show
```

You can also combine balloon object variables in an array and index into the array.

BalloonCheckboxes Collection Object

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofobjBalloonCheckboxesC "} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofobjBalloonCheckboxesX": 1} {ewc HLP95EN.DLL, DYNALINK, "Properties": "ofobjBalloonCheckboxesP "}
{ewc HLP95EN.DLL, DYNALINK, "Methods": "ofobjBalloonCheckboxesM "} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofobjBalloonCheckboxesS "}

Assistant

Balloon
Balloon

Balloon
Balloon
Balloon

Balloon
Balloon
Balloon

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. 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 writes the checkbox text of all visible checkboxes, in an existing balloon, to the debug window.

```
For Each box In .CheckBoxes  
    If box.Text <> "" Then  
        Debug.Print box.Text  
    End If  
Next
```

You cannot add check boxes to or remove check boxes from the **BalloonCheckboxes** collection.

Remarks

Balloon check boxes display the user's choices until he or she dismisses the balloon. Balloon labels record the user's choice as soon as he or she clicks the button beside the label.

BalloonCheckbox Object

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofobjBalloonCheckboxC "} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofobjBalloonCheckboxX":1} {ewc HLP95EN.DLL, DYNALINK, "Properties":"ofobjBalloonCheckboxP "}
{ewc HLP95EN.DLL, DYNALINK, "Methods":"ofobjBalloonCheckboxM "} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofobjBalloonCheckboxS "}

Balloon

Balloon
Balloon

Balloon
Balloon
Balloon

Balloon
Balloon
Balloon

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. When the user selects a check box and then clicks **OK**, the appropriate procedure is run.

```
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
    Select Case True
        Case .CheckBoxes(1).Checked
            runregion1
        Case .CheckBoxes(2).Checked
            runregion2
        Case .CheckBoxes(3).Checked
            runregion3
    End Select
End With
```

Remarks

Balloon check boxes display the user's choices until the user dismisses the balloon. Balloon labels record the user's choice as soon as the user clicks the button beside the label.

BalloonLabels Collection Object

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofobjBalloonLabelsC "} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofobjBalloonLabelsX": 1} {ewc HLP95EN.DLL, DYNALINK, "Properties": "ofobjBalloonLabelsP "} {ewc HLP95EN.DLL, DYNALINK, "Methods": "ofobjBalloonLabelsM "} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofobjBalloonLabelsS "}

Balloon

Balloon
Balloon

Balloon
Balloon
Balloon

Balloon
Balloon
Balloon

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. There can be up to five labels in one balloon; each label appears when a value is assigned to its **Text** property.

The following 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, corresponding to the label the user clicks. The example 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
```

Remarks

Balloon check boxes display the user's choices until the user dismisses the balloon. Balloon labels record the user's choice as soon as the user clicks the button beside the label.

BalloonLabel Object

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofobjBalloonLabelC "} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofobjBalloonLabelX":1} {ewc HLP95EN.DLL, DYNALINK, "Properties":"ofobjBalloonLabelP "} {ewc HLP95EN.DLL, DYNALINK, "Methods":"ofobjBalloonLabelM "} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofobjBalloonLabelS "}
```

Balloon

Balloon
Balloon

Balloon
Balloon
Balloon

Balloon
Balloon
Balloon

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 assigned 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 = "Over 30."
    .Labels(3).Text = "None of your business."
    .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. Balloon labels record the user's choice as soon as he or she clicks the button beside the label.

Add Method

Adds an object to a collection. Select one of the following collections to see a detailed description of the **Add** method for that collection.

CommandBarControls

CommandBars

DocumentProperties

PropertyTests

Add Method (CommandBarControls Collection)

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthAddCommandBarControlsObjC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthAddCommandBarControlsObjX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthAddCommandBarControlsObjA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthAddCommandBarControlsObjS"}

Creates a new command bar control and adds it to the collection of controls on the specified command bar. Returns a **CommandBarButton**, **CommandBarComboBox**, or **CommandBarPopup** object.

Syntax

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 (CommandBarControls Collection) Example

This example adds a button control with a charting face to the command bar named "Custom," and then it assigns the control the procedure named "Analyze."

```
Set custBar = CommandBars("Custom")
custBar.Visible = True
Set graphBtn = custBar.Controls.Add(Type:=msoControlButton)
graphBtn.FaceId = 17
graphBtn.OnAction = "Analyze"
```


Add Method (CommandBars Collection)

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthAddCommandBarsObjC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthAddCommandBarsObjX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthAddCommandBarsObjA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthAddCommandBarsObjS"}

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

Syntax

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, Word assigns a default name to the command bar (such as "Custom 1").

Position Optional **Variant**. The position of the new command bar. Can be one of the following **MsoBarPosition** constants.

Constant	Description
msoBarLeft , msoBarTop , msoBarRight , msoBarBottom	Indicate 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	Indicates that the new command bar will replace the system menu bar on the Macintosh

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 (CommandBars Collection) Example

This example adds a top-level command bar named "Custom" that won't be saved when the session ends. The example also adds a built-in spelling-checker button to the command bar.

```
Set mybar = CommandBars _  
    .Add(Name:="Custom", Position:=msoBarTop, _  
        Temporary:=True)  
With mybar  
    .Controls.Add Type:=msoControlButton, Id:=2  
    .Visible = True  
End With
```

Add Method (DocumentProperties Collection)

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthAddDocumentPropertiesObjC"} {ewc HLP95EN.DLL, DYNALINK,  
"Example":"ofmthAddDocumentPropertiesObjX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies  
To":"ofmthAddDocumentPropertiesObjA"} {ewc HLP95EN.DLL, DYNALINK,  
"Specifics":"ofmthAddDocumentPropertiesObjS"}
```

Creates a new custom document property. You can use this method only with the collection of custom document properties.

Syntax

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 Required **Long**. 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 possible; otherwise, an error occurs. If **LinkToContent** is **True**, the **Value** argument is ignored and the new document property has a default value until linked property values are updated by the container application (usually when the document is saved).

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

Add Method (DocumentProperties Collection) Example

This example adds a new custom document property and names it "Complete." You must pass the custom **DocumentProperties** collection to the procedure.

```
Sub AddCustomProperty(dp As DocumentProperties)
    dp.Add name:="Complete", linkToContent:=False, _
        type:=msoPropertyTypeBoolean, value:=False
End Sub
```

Add Method (PropertyTests Collection)

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthAddPropertyTestsObjC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthAddPropertyTestsObjX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthAddPropertyTestsObjA"}

Adds a **PropertyTest** object to the **PropertyTests** collection.

Syntax

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

expression 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 **Advanced Find** dialog box.

Condition Required **Variant**. The condition of the search criteria. Can be one of the following **MsoCondition** constants:

msoConditionAnyNumberBetween	msoConditionInTheNext
msoConditionAnytime	msoConditionIsExactly
msoConditionAnytimeBetween	msoConditionIsNo
msoConditionAtLeast	msoConditionIsNot
msoConditionAtMost	msoConditionIsYes
msoConditionBeginsWith	msoConditionLastMonth
msoConditionDoesNotEqual	msoConditionLastWeek
msoConditionEndsWith	msoConditionLessThan
msoConditionEquals	msoConditionMoreThan
msoConditionFileTypeAllFiles	msoConditionNextMonth
msoConditionFileTypeBinders	msoConditionNextWeek
msoConditionFileTypeDatabases	msoConditionOn
msoConditionFileTypeExcelWorkbooks	msoConditionOnorAfter
msoConditionFileTypeOfficeFiles	msoConditionOnorBefore
msoConditionFileTypePowerPointPresentations	msoConditionThisMonth
msoConditionFileTypeTemplates	msoConditionThisWeek
msoConditionFileTypeWordDocuments	msoConditionToday
msoConditionIncludesNearEachOther	msoConditionTomorrow
msoConditionIncludesPhrase	msoConditionYesterday
msoConditionInTheLast	

Value Optional **Variant**. The value of the search criterion to test for.

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

Connector Optional **Variant**. Specifies the way two search criteria are combined. Can be either of the following **msoConnector** constants: **msoConnectorAnd** or **msoConnectorOr**.

Add Method (PropertyTests Collection) Example

This example adds two property tests to the search criteria. The first test is that the found files must be Word documents, and the second test is that the found files must have been modified between January 1, 1996 and June 30, 1996.

```
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:="1/1/96", SecondValue:="6/30/96", _
        Connector:=msoConnectorAnd
End With
```

Copy Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthCopyC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthCopyX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthCopyA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthCopyS"}

Copies the specified command bar control to an existing command bar.

Syntax

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 same command bar (the command bar it's already on).

Before Optional **Variant**. A number that indicates the position for the new control on the specified 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.

Copy Method Example

This example copies the first control from the command bar named "Standard" to the command bar named "Custom" and positions it as the first control there too. The example assigns a new parameter to the control and sets the focus to the new button.

```
Set myCustomBar = CommandBars("Custom")
Set myControl = CommandBars("Standard").Controls(1)
With myControl
    .Copy Bar:=myCustomBar, Before:=1
    .Parameter = "2"
    .SetFocus
End With
```


Count Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproCountC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproCountX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproCountA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproCountS"}

Returns the number of items in the specified collection. Read-only **Long**.

Remarks

For the **CommandBars** collection, the count includes only menu bars, toolbars, and shortcut menus. Menus and submenus aren't included.

Count Property Example

This example uses the **Count** property to display the number of command bars in the collection.

```
MsgBox "There are " & CommandBars.Count & " bars in your 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 uses the **Count** property to display the number of labels in the Office Assistant balloon.

```
With Assistant.NewBalloon
    .Heading = "Check Your Age Group."
    .Labels(1).Text = "20 to 29."
    .Labels(2).Text = "30 to 39."
    .Labels(3).Text = "40 or over"
    .Text = "Which of the following " &
    & .Labels.Count & " choices apply to you?"
    .Show
End With
```

This example displays the number of document properties in the **DocumentProperties** collection that was passed to the procedure.

```
Sub CountDocumentProperties(dp As DocumentProperties)
    MsgBox "There are " & dp.Count & " properties in the collection."
End Sub
```

Delete Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthDeleteC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthDeleteX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthDeleteA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthDeleteS"}

Deletes the specified object from the collection it's contained in.

Syntax 1

expression.Delete

Syntax 2

expression.Delete(*Temporary*)

Syntax 3

expression.Delete(*BstrQueryName*)

expression Required. An expression that returns a **CommandBar** or **DocumentProperty** object (Syntax 1), a **CommandBarControl** object (Syntax 2), or a **FileFind** object (Syntax 3).

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

BstrQueryName Required **String**. A string containing up to 31 characters that indicates the name of the saved search criterion to be deleted.

Remarks

You cannot delete a built-in document property.

Delete Method Example

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
If Not foundFlag Then
    MsgBox "No command bars have been deleted."
Else
    MsgBox delBars & " custom bar(s) deleted."
End If
```

This example deletes a custom document property. You must pass a **DocumentProperty** object to the procedure.

```
Sub DeleteCustomDocumentProperty(dp As DocumentProperty)
    dp.Delete
End Sub
```

Execute Method

Runs a procedure associated with a command bar control or begins searching for files. Select one of the following objects to see a detailed description of the **Execute** method for that object.

[**CommandBarButton**](#)

[**CommandBarControl**](#)

[**CommandBarPopup**](#)

[**CommandBarComboBox**](#)

[**FileSearch**](#)

[**FileFind**](#)

Execute Method (Command Bar Controls)

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthExecuteCommandBarControlObjC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthExecuteCommandBarControlObjX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthExecuteCommandBarControlObjA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthExecuteCommandBarControlObjS"}

Runs the macro or built-in command assigned to the specified command bar control. For custom controls, use the **OnAction** property to specify the macro to run.

Syntax

expression.**Execute**

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

Execute Method (Command Bar Controls) Example

This example checks the value of the combo box control on the custom command bar named "My Command Bar." If the value is "First," the example runs the macro specified by the **OnAction** property of the command bar control.

```
Set mycontrol = CommandBars("My Custom Bar").Controls(1)
With mycontrol
    If .List(.ListIndex) = "First" Then
        mycontrol.Execute
    End If
End With
```

Execute Method (FileSearch and FileFind Objects)

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthExecuteFileSearchObjC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthExecuteFileSearchObjX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthExecuteFileSearchObjA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthExecuteFileSearchObjS"}
```

FileSearch object: Begins the search for the specified files.

FileFind object: Begins the search for the specified files and updates the **FindFileResults** collection. Macintosh only.

Syntax 1

expression.**Execute**(*SortBy*, *SortOrder*, *AlwaysAccurate*)

Syntax 2

expression.**Execute**

expression Required. An expression that returns a **FileSearch** object (Syntax 1) or a **FileFind** object (Syntax 2).

SortBy Optional **Variant**. The method used to sort the returned files. Can be one of the following **MsoSortBy** constants: **msoSortbyFileName**, **msoSortbyFileType**, **msoSortbyLastModified**, or **msoSortbySize**.

SortOrder Optional **Variant**. The order in which the returned files are to be sorted. Can be either of the following **MsoSortOrder** constants: **msoSortOrderAscending** or **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 (FileSearch and FileFind Objects) Example

This example searches for all files that begin with "cmd" in the My Documents folder and displays the location and name of each file that's found. The example sorts the list of 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
```

SetFocus Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthSetFocusC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthSetFocusX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthSetFocusA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthSetFocusS"}

Moves the keyboard focus to the specified command bar control so that it can receive keyboard input. The kind of keyboard input you can direct to the control depends on what type of control it is.

Note If the control is disabled or isn't visible from the current state, this method will fail.

Syntax

expression.**SetFocus**

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

SetFocus Method Example

This example copies the fourth control from the My Custom Bar custom command bar and pastes it onto the same command bar as the first control. The example also assigns a new parameter to the control and sets the focus to the new button.

```
Set mycontrol = CommandBars("My Custom Bar") _  
    .Controls(4)  
With mycontrol  
    .Copy Before:=1  
    .Parameter = "2"  
    .SetFocus  
End With
```

Text Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproTextC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofproTextX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproTextA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproTextS"}

CommandBarComboBox object: Returns or sets the text in the display or edit portion of the command bar combo box control. Read/write **String**.

BalloonLabel or **BalloonCheckbox** object: Returns or sets the text displayed next to the specified 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**.

FileFind object: Returns or sets the string to look for, up to 80 characters, in a the body of a document during a file search. Macintosh only. Read-write **String**.

Remarks

For the **Balloon**, **BalloonLabel**, and **BalloonCheckbox** objects, you can specify a graphic to display by using the following syntax: *{type location sizing_factor}*, where *type* is bmp (bitmap), wmf (Windows metafile), or pict (Macintosh PICT file); *location* is the resource id or the path and file name; and *sizing_factor* specifies the width of the wmf or pict (omitted for bmp).

Text Property Example

This example checks the number of items in the combo box control on the command bar named "Custom." If there are more than three items in the list, the example clears the list and adds a new first item to it. The new default item for this control is "Floyd."

```
Set myBar = CommandBars("Custom")
Set myControl = myBar.Controls(2)
With myControl
    If .ListCount > 3 Then
        .Clear
        .AddItem "Floyd", 1
        .Text = "Floyd"
    End If
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 a label for each check 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 with a heading, text, and three region choices. The example uses the **Text** property to provide balloon-related instructions to the user.

```
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
```

Top Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproTopC"}
{ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproTopA"}

{ewc HLP95EN.DLL, DYNALINK, "Example": "ofproTopX": 1}
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproTopS"}

CommandBar or **CommandBarControl** object: Returns or sets the distance (in pixels) from the top edge of the specified command bar or command bar control 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** for **CommandBar**, read-only **Long** for **CommandBarControl**.

Assistant object: Sets or returns the vertical position of the Office Assistant window (in points), relative to the screen. Read/write **Integer**.

Top Property Example

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
```

This example moves the Office Assistant to the specified coordinate.

```
Assistant.Top = 500
```

HelpContextId Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproHelpContextIdC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofproHelpContextIdX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproHelpContextIdA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproHelpContextIdS"}

Returns or sets the Help context ID number for the Help topic attached to the command bar control.
Read/write **Long**.

Remarks

To use this property, you must also set the **HelpFile** property.

HelpContextId Property Example

This example adds a custom report manager button to the **File** menu and specifies a Help topic to be displayed as that button's context-sensitive Help.

```
Set fileMenu = CommandBars _  
    .FindControl(Type:=msoControlPopup, Id:=fileID)  
Set fileMenuDropdown = fileMenu.CommandBar  
Set newButton = fileMenuDropdown.Controls _  
    .Add(Type:=msoControlButton, Before:=3)  
With newButton  
    .Caption = "Open Report"  
    .DescriptionText = "Opens a quarterly report form"  
    .OnAction = "ReportManager"  
    .HelpFile = "C:\corphelp\custom.hlp"  
    .HelpContextID = 47  
End With
```

HelpFile Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproHelpFileC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproHelpFileX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproHelpFileA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproHelpFileS"}

Returns or sets the Help file name for the Help topic attached to the command bar control. Read/write **String**.

Remarks

To use this property, you must also set the **HelpContextID** property.

HelpFile Property Example

This example adds a custom report manager button to the **File** menu and specifies a Help topic to be displayed as that button's context-sensitive Help.

```
Set fileMenu = CommandBars _  
    .FindControl(Type:=msoControlPopup, Id:=fileId)  
Set fileMenuDropdown = fileMenu.CommandBar  
Set newbutton = fileMenuDropdown.Controls _  
    .Add(Type:=msoControlButton, Id:=customId, Before:=3)  
With newButton  
    .Caption = "Open Report"  
    .DescriptionText = "Opens a quarterly report form"  
    .OnAction = "ReportManager"  
    .HelpFile = "C:\corphelp\custom.hlp"  
    .HelpContextID = 47  
End With
```

LargeButtons Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproLargeButtonsC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproLargeButtonsX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproLargeButtonsA"}
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproLargeButtonsS"}

True if large toolbar buttons are displayed. Read/write **Boolean**.

LargeButtons Property Example

This example displays large toolbar buttons and ScreenTips on all command bars if the main menu bar that's active is named "Custom."

```
Set allBars = CommandBars
If allBars.ActiveMenuBar.Name = "Custom" Then
    allBars.LargeButtons = True
    allBars.ShowToolTips = True
End If
```

List Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproListC"}
{ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproListA"}

{ewc HLP95EN.DLL, DYNALINK, "Example":"ofproListX":1}
{ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproListS"}

Returns or sets the value of a list item in the command bar combo box control. Read/write **String**.

Note This property is read-only for built-in combo box controls.

Syntax

expression.**List**(*Index*)

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

Index Required **Long**. The list item to be set.

List Property Example

This example checks the value of the fourth list item in the combo box control named "Vegetables" on the command bar named "Custom." If the value is "Tomato," the code specified by the **OnAction** property of the command bar control is run.

```
Set myBar = CommandBars("Custom")
Set myControl = myBar.Controls("Vegetables")
If myControl.List(4) = "Tomatoes" Then myControl.Execute
```

ListCount Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproListCountC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofproListCountX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproListCountA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproListCountS"}

Returns the number of list items in the specified command bar combo box control. Read-only **Long**.

ListCount Property Example

This example checks the number of items in the combo box control on the command bar named "Custom." If there are more than three items in the list, the list is cleared and a new first item is added and displayed as the default for the combo box control.

```
Set myBar = CommandBars("Custom")
Set myControl = myBar.Controls(2)
With myControl
    If .ListCount > 3 Then
        .Clear
        .AddItem Text:="Floyd", Index:=1
        .ListIndex = 1
    End If
End With
```

ListHeaderCount Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproListHeaderCountC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproListHeaderCountX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproListHeaderCountA"}
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproListHeaderCountS"}

Returns or sets the number of list items in the command bar combo box control that appear 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.

ListHeaderCount Property Example

The following example adds a combo box control to the command bar named "Custom," and then it adds two items to the combo box list. 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
```

ListIndex Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproListIndexC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproListIndexX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproListIndexA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproListIndexS"}

Returns or sets the index number of the selected item in the list portion of the command bar combo box control. 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.

ListIndex Property Example

This example runs an existing procedure, based on the selection in the combo box. The text in the combo box can be anything, because the example uses either the **ListIndex** property or the index number of the text entry to determine which procedure to run.

```
With myControl
  Select Case .ListIndex
    Case 1
      chartcourse
    Case 2
      displaygraph
    Case Else
      MsgBox "invalid choice"
  End Select
End With
```

NameLocal Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproNameLocalC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproNameLocalX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproNameLocalA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproNameLocalS"}

Returns the name of a built-in **command bar** 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.

Remarks

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.

NameLocal Property Example

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
```

OLEMenuGroup Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproOLEMenuGroupC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproOLEMenuGroupX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproOLEMenuGroupA"}
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproOLEMenuGroupS"}

Returns or sets the menu group that the specified command bar **pop-up control** 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). Can be one of the following **MsoOLEMenuGroup** constants: **msoOleMenuGroupNone**, **msoOleMenuGroupFile**, **msoOleMenuGroupEdit**, **msoOleMenuGroupContainer**, **msoOleMenuGroupObject**, **msoOleMenuGroupWindow**, or **msoOleMenuGroupHelp**. Read/write **Long**.

Note This property is read-only for built-in controls.

Remarks

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.

OLEMenuGroup Property Example

This example checks the **OLEMenuGroup** property of a new custom popup control on the active menu bar, and sets it to **msoOLEMenuGroupNone**.

```
Set myControl = ActiveMenuBar.Controls _  
    .Add(Type:=msoControlPopup, Temporary:=False)  
myControl.OLEMenuGroup = msoOLEMenuGroupNone
```

OLEUsage Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproOLEUsageC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproOLEUsageX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproOLEUsageA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproOLEUsageS"}

Returns or sets the OLE client and OLE server roles in which a command bar control will be used when two Microsoft Office applications are merged. Can be one of the following **MsoControlOLEUsage** constants: **msoControlOLEUsageNeither**, **msoControlOLEUsageServer**, **msoControlOLEUsageClient**, or **msoControlOLEUsageBoth**. Read/write **Long**.

Remarks

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.

OLEUsage Property Example

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
```

OnAction Property

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproOnActionC"} {ewc HLP95EN.DLL, DYNALINK,  
"Example": "ofproOnActionX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproOnActionA"} {ewc  
HLP95EN.DLL, DYNALINK, "Specifics": "ofproOnActionS"}
```

Returns or sets the name of the Visual Basic macro that will be run when the user clicks or changes the value of a command bar control. Read/write **String**.

Note The container application determines whether the value is a valid macro name.

OnAction Property Example

This example adds a command bar control to the command bar named "Custom" and sets the macro named "MySub" to run whenever the button is clicked.

```
Set myBar = .CommandBars("Custom")
Set myControl = myBar.Controls _
    .Add(Type:=msocontrolButton)
With myControl
    .FaceId = 2
    .OnAction = "MySub"
End With
myBar.Visible = True
```

Parameter Property

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproParameterC"}          {ewc HLP95EN.DLL, DYNALINK,  
"Example":"ofproParameterX":1}          {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproParameterA"} {ewc  
HLP95EN.DLL, DYNALINK, "Specifics":"ofproParameterS"}
```

Returns or sets a string that an application can use to execute a command associated with a command bar control. Read/write **String**.

Remarks

If the specified parameter is set for a built-in control, the application can modify its default behavior if it's able to 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).

Parameter Property Example

This example copies the fourth control from the custom command bar named "Custom" and pastes it onto the same command bar as the new first control. The example assigns a new parameter and sets the focus to the new button.

```
Set myControl = CommandBars("Custom").Controls(4)
With myControl
    .Copy , 1
    .Parameter = "2"
    .SetFocus
End With
```

PasteFace Method

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthPasteFaceC"} {ewc HLP95EN.DLL, DYNALINK,  
"Example":"ofmthPasteFaceX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthPasteFaceA"} {ewc  
HLP95EN.DLL, DYNALINK, "Specifics":"ofmthPasteFaceS"}
```

Pastes the contents of the Clipboard onto the specified command bar button control.

Syntax

expression.**PasteFace**

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

PasteFace Method Example

This example finds the built-in **FileOpen** button and pastes a custom face onto it from the Clipboard, where the user had previously altered it. This example will fail if a custom face doesn't already exist in the Clipboard; use the **CopyFace** method to copy a specified button face to the Clipboard.

```
Set myControl = CommandBars.FindControl(Type:=msoControlButton, Id:=23)  
myControl.PasteFace
```

Position Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproPositionC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofproPositionX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproPositionA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproPositionS"}

Returns or sets the position of the specified `command bar`. Can be one of the following **MsoBarPosition** constants: **msoBarLeft**, **msoBarTop**, **msoBarRight**, **msoBarBottom**, **msoBarFloating**, **msoBarPopup**, or **msoBarMenu**. Read/write **Long**.

Position Property Example

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
End If
Next
```

Priority Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproPriorityC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproPriorityX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproPriorityA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproPriorityS"}

Returns or sets the priority of the specified **command bar control**. A control's priority determines whether it can be dropped from a docked command bar if the command bar controls don't fit in a single row. Read/write **Long**.

Remarks

A priority of 1 means the control cannot be dropped. Controls with the highest priority numbers are dropped first.

A priority of 0 indicates an "automatic" value, which will choose an effective priority based on the control type. A priority of 1 means that the control will never be dropped. The other valid priorities are from 2 through 7, and these controls will be dropped in order of their values (starting with the highest values).

Priority Property Example

This example moves the first combo box control from the custom command bar named "Custom" to the position before the seventh control on the command bar, sets the tag to "selection box," and then assigns the control a high priority 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
    End If
Next
```

Protection Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproProtectionC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproProtectionX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproProtectionA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproProtectionS"}

Returns or sets the way the specified `command bar` is protected from user customization. Can be one of or a sum of the following **MsoBarProtection** constants: **msoBarNoProtection**, **msoBarNoCustomize**, **msoBarNoResize**, **msoBarNoMove**, **msoBarNoChangeVisible**, **msoBarNoChangeDock**, **msoBarNoVerticalDock**, or **msoBarNoHorizontalDock**. Read/write **Long**.

Protection Property Example

This example steps through the collection of command bars to find the command bar named "Forms." If this command bar is found, the example makes it visible and protects its docking state.

```
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
```

ReleaseFocus Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthReleaseFocusC"} {ewc HLP95EN.DLL, DYNALINK,
"Example":"ofmthReleaseFocusX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthReleaseFocusA"}
{ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthReleaseFocusS"}

Releases the user interface focus from all command bars.

Syntax

expression.**ReleaseFocus**

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

ReleaseFocus Method Example

This example releases the user interface focus from all command bars.

```
CommandBars.ReleaseFocus
```

RemoveItem Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthRemoveItemC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthRemoveItemX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthRemoveItemA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthRemoveItemS"}

Removes a list item from the specified command bar combo box control.

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

Syntax

expression.**RemoveItem**(*Index*)

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

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

RemoveItem Method Example

The following example determines whether there are more than three items in the combo box control named "Combo1" on the custom command bar named "Custom." If there are more than three items, the example removes the second item, alters the style to not show the combo box label, 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("Custom")
Set myControl = myBar.Controls("Combo1")
With myControl
    If .ListCount > 3 Then
        .RemoveItem 2
        .Style = msoComboNormal
        .Text = "New Default"
        Set ctrl = .Parent
        ctrl.Tag = "Contents Changed"
    End If
End With
```

Reset Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthResetC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthResetX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthResetA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthResetS"}

Resets the specified built-in command bar to its default configuration of controls, or resets the specified built-in command bar control to its default function and face.

Syntax

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.

Reset Method Example

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
```

RowIndex Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproRowIndexC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproRowIndexX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproRowIndexA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproRowIndexS"}

Returns or sets the docking order of the specified **command bar** 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

Command bars with lower numbers are docked first. Several command bars can share the same row index. If two or more command bars share the same row index, the command bar most recently assigned will be displayed first in its group.

RowIndex Property Example

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 1.

```
Set myBar = CommandBars("Custom")
With myBar
    .RowIndex = 1
    .Left = 140
End With
```

ShowPopup Method

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofmthShowPopupC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofmthShowPopupX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofmthShowPopupA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofmthShowPopupS"}

Displays the specified command bar 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.

Syntax

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.

ShowPopup Method Example

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:="Custom1", Position:=msoBarPopup, Temporary:=False)  
With myBar  
    .Controls.Add Type:=msoControlButton, Id:=3  
    .Controls.Add Type:=msoControlComboBox  
End With  
myBar.ShowPopup
```

State Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproStateC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofproStateX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproStateA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproStateS"}

Returns or sets the appearance of the specified command bar `button` control. Can be one of the following **MsoButtonState** constants: **msoButtonUp**, **msoButtonDown**, or **msoButtonMixed**.
Read/write **Long**.

State Property Example

This example determines whether the first control on the command bar named "Custom" has a built-in button face. If it does, the example copies the button face to the Clipboard so that it can be customized with an application such as Microsoft Paint. The example then sets the button state to **msoButtonUp**.

```
Set myControl = CommandBars("Custom").Controls(1)
With myControl
    If .BuiltInFace = True Then
        .CopyFace
    End If
    .State = msoButtonUp
End With
```

Style Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproStyleC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproStyleX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproStyleA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproStyleS"}

CommandBarButton object: Returns or sets the way the specified command bar button control is displayed. Can be one of the following **MsoButtonStyle** constants: **msoButtonAutomatic**, **msoButtonIcon**, **msoButtonCaption**, or **msoButtonIconandCaption**. Read/write **Long**.

CommandBarComboBox object: Returns or sets the way the specified command bar combo box control is displayed. Can be either of the following **MsoComboStyle** constants: **msoComboLabel** or **msoComboNormal**. Read/write **Long**.

Style Property Example

This example creates a shortcut menu containing a button control and a combobox 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  
myBar.ShowPopup
```

Tag Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproTagC"}
{ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproTagA"}

{ewc HLP95EN.DLL, DYNALINK, "Example":"ofproTagX":1}
{ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproTagS"}

Returns or sets information about the command bar control, such as data that can be used as an argument in procedures, or information that identifies the control. Read/write **String**.

Tag Property Example

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

```
Set allControls = CommandBars("My Custom Bar").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
```

TooltipText Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproTooltipTextC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofproTooltipTextX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproTooltipTextA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproTooltipTextS"}

Returns or sets the text displayed in the specified command bar control's ScreenTip. Read/write **String**.

Remarks

By default, the value of the **Caption** property is used as the ScreenTip.

TooltipText Property Example

This example sets the last control on the active menu bar to begin its own group, and then adds a ScreenTip to it.

```
Set myMenuBar = CommandBars.ActiveMenuBar
Set lastCtrl = myMenuBar _
    .Controls(myMenuBar.Controls.Count)
lastCtrl.BeginGroup = True
lastCtrl.TooltipText = "Click for help on UI feature"
End With
```

ActiveMenuBar Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproActiveMenuBarC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproActiveMenuBarX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproActiveMenuBarA"}
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproActiveMenuBarS"}

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

ActiveMenuBar Property Example

This example adds a temporary pop-up control named "Custom" to the end of the active menu bar, and then it adds a button control named "Import" to the command bar displayed by the control.

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

AddItem Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthAddItemC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthAddItemX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthAddItemA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthAddItemS"}

Adds a list item to the specified command bar **combo box control**. The combo box control must be a custom control and it must be either a drop-down list box or a combo box.

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

Syntax

expression.**AddItem**(*Text*, *Index*)

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

Text Required **String**. The item to be added to the specified control.

Index Optional **VARIANT**. The position of the specified item in the list of items. If this argument is omitted, the item is added at the end of the list.

AddItem Method Example

This example adds a combo box control to the command bar named "Custom," and then it adds two items to the list. 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
```

BeginGroup Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproBeginGroupC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproBeginGroupX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproBeginGroupA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproBeginGroupS"}

True if the specified command bar control is at the beginning of a group of controls on the command bar. Read/write **Boolean**.

BeginGroup Property Example

This example sets the last control on the active menu bar to be at the beginning of its own group.

```
Set myMenuBar = CommandBars.ActiveMenuBar
Set lastMenu = myMenuBar _
    .Controls(myMenuBar.Controls.Count)
lastMenu.BeginGroup = True
```

BuiltIn Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproBuiltInC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofproBuiltInX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproBuiltInA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproBuiltInS"}

True if the specified `command bar` or `command bar control` 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 who's **OnAction** property has been set. Read-only **Boolean**.

BuiltIn Property Example

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
```

BuiltInFace Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproBuiltInFaceC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproBuiltInFaceX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproBuiltInFaceA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproBuiltInFaceS"}

True if the face of the specified command bar button control is its built-in face. This property can only be set to **True**, which will reset the face to the built-in one. Read/write **Boolean**.

BuiltInFace Property Example

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.

```
Set myControl = CommandBars("My Custom Bar").Controls(1)
With myControl
    If .BuiltInFace = True Then .CopyFace
End With
```

Caption Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproCaptionC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproCaptionX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproCaptionA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproCaptionS"}

Returns or sets the caption text for the specified command bar control. Read/write **String**.

Note The caption for a control is also displayed as its default ScreenTip.

Caption Property Example

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
```

Clear Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthClearC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthClearX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthClearA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthClearS"}

Removes all list items from the specified command bar combo box control (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.

Syntax

expression.Clear

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

Clear Method Example

This example checks the number of items in the combo box control named "Names" on the command bar named "Custom." If there are more than three items in the list, the example clears the list, adds a new first item to the list, and displays this new item as the default for the combo box control.

```
Set myBar = CommandBars("Custom Bar")
Set myControl = myBar.Controls("Names")
With myControl
    If .ListCount > 3 Then
        .Clear
        .AddItem Text:="Bendel", Index:=1
        .ListIndex = 1
    End If
End With
```

CommandBar Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproCommandBarC"} {ewc HLP95EN.DLL, DYNALINK,
"Example":"ofproCommandBarX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproCommandBarA"}
{ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproCommandBarS"}

Returns a **CommandBar** object that represents the menu displayed by the specified pop-up control.
Read-only.

CommandBar Property Example

This example sets `thirdLevel` to the fourth control on the command bar named "Drawing."

```
Set thirdLevel = CommandBars("Drawing") _  
    .Controls(1).CommandBar.Controls(4)
```

Controls Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproControlsC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproControlsX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproControlsA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproControlsS"}

Returns a **CommandBarControls** object that represents all the controls on the specified command bar or popup control. Read-only.

For information about returning a single member of a collection, see **Returning an Object from a Collection**.

Controls Property Example

This example adds a combo box control to the command bar named "Custom" and adds two items to the combo box list. 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
```

CopyFace Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthCopyFaceC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthCopyFaceX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthCopyFaceA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthCopyFaceS"}

Copies the face of the specified command bar button control to the Clipboard.

Syntax

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.

CopyFace Method Example

This example finds the built-in **Open** button and copies its button face to the Clipboard.

```
Set myControl = CommandBars.FindControl(Type:=msoControlButton, Id:=23)
myControl.CopyFace
```

DescriptionText Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproDescriptionTextC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproDescriptionTextX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproDescriptionTextA"}
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproDescriptionTextS"}

Returns or sets the description for the specified command bar control. The description is displayed in the status bar of the container application when the user positions the pointer over a command bar control. Read/write **String**.

Remarks

Not all applications display a status bar. This property is also used for Balloon Help on the Macintosh.

DescriptionText Property Example

This example adds a command bar control with a spelling checker button face to a custom command bar. The example also sets the text "Starts the spelling checker" to appear in the status bar whenever the user positions the pointer over the button.

```
Set myBar = CommandBars.Add("Custom", msoBarTop, , 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
```

DisplayTooltips Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproDisplayTooltipsC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofproDisplayTooltipsX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproDisplayTooltipsA"}
{ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproDisplayTooltipsS"}

True if ScreenTips are displayed whenever the user positions the pointer over command bar controls.
Read/write **Boolean**.

Remarks

Setting the **DisplayTooltips** property in a container application immediately affects all the command bars in that application, in any other Office 97 application running at that time, and in any Office 97 application opened after that time, until the property is set again.

DisplayTooltips Property Example

This example displays large controls and ScreenTips on all command bars if the menu bar named "Custom Menu Bar" is the active menu bar.

```
Set allBars = CommandBars
If allBars.ActiveMenuBar.Name = "Custom Menu Bar" Then
    allBars.LargeButtons = True
    allBars.DisplayTooltips = True
End If
```

DropDownLines Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproDropDownLinesC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproDropDownLinesX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproDropDownLinesA"}
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproDropDownLinesS"}

Returns or sets the number of lines in the specified command bar **combo box control**. The combo box control must be a custom control and it must be either a drop-down list box or a combo box.

Read/write **Long**.

Note If this property is set for a combo box control that's either an edit box or a built-in combo box control, an error occurs.

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.

DropDownLines Property Example

This example adds a combo box control to the command bar named "Custom" and then adds two items to the combo box list. 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

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproDropDownWidthC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproDropDownWidthX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproDropDownWidthA"}
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproDropDownWidthS"}

Returns or sets the width (in pixels) of the list for the specified command bar combo box control.

Read/write **Long**.

Note If this property is set for a built-in combo box control, an error occurs.

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.

DropDownWidth Property Example

This example adds a combo box control to the command bar named "Custom" and then adds two items to the combo box list. 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
```

Enabled Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproEnabledC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproEnabledX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproEnabledA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproEnabledS"}

True if the specified command bar or command bar control 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.

Enabled Property Example

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 command bar "Visual Basic" is reset to its default state and the command bar named "VB Custom Bar" is disabled.

```
Set myBar = CommandBars("VB Custom Bar")
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 represented by the variable `myBar`. The first control is disabled and will appear grey. The second control is enabled by default.

```
With myBar
    .Controls.Add Type:=msoControlButton, Id:=3
    .Controls(1).Enabled = False
    .Controls.Add Type:=msoControlButton, Id:=3
End With
myBar.Visible = True
```

FaceId Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproFaceIdC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproFaceIdX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproFaceIdA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproFaceIdS"}

Returns or sets the ID number for the button face that's currently assigned to the specified command bar button control. Read/write **Long**.

Remarks

The **FaceId** property dictates the look, but 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).

FaceId Property Example

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. When you click this button, the **Open** dialog box is displayed.

```
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
```

FindControl Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthFindControlC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthFindControlX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthFindControlA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthFindControlS"}

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

Syntax

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

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

Type Optional **Variant**. The type of control to be searched for. Can be one of the following

MsoControlType constants:

msoControlCustom	msoControlGraphicDropdown
msoControlButton	msoControlPopup
msoControlEdit	msoControlGraphicPopup
msoControlDropdown	msoControlButtonPopup
msoControlComboBox	msoControlGauge
msoControlButtonDropdown	msoControlLabel
msoControlSplitDropdown	msoControlExpandingGrid
msoControlGenericDropdown	msoControlGrid
msoControlGraphicCombo	msoControlOCXDropDown
msoControlSplitButtonMRUPopup	msoControlSplitButtonPopup
msoControlSplitExpandingGrid	

Id Optional **Variant**. The identifier of the control to be searched for.

Tag Optional **Variant**. The tag value of the control to be searched for.

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

Recursive Optional **Boolean**. **True** to include the command bar and all of its popup sub-toolbars in the search. The default value is **False**.

Remarks

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**. None of the arguments for the **FindControl** method have a default value.

FindControl Method Example

This example adds the **Save** button to the **Help** menu on the menu bar. Using the **FindControl** method ensures that the **Help** menu will be found even if the user has customized the menu bar.

```
Set helpMenu = CommandBars.FindControl _  
    (Type:=msoControlPopup, Id:=helpId)  
Set helpMenuDrop = helpMenu.Control.CommandBar  
helpMenuDrop.Controls.Add Type:=msoControlButton, Id:=saveId
```

DisplayKeysInTooltips Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproDisplayKeysInTooltipsC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproDisplayKeysInTooltipsX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproDisplayKeysInTooltipsA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproDisplayKeysInTooltipsS"}

True if shortcut keys are displayed in the ToolTips for each command bar control . Read/write **Boolean**.

Remarks

To display shortcut keys in ToolTips, you must also set the DisplayTooltips property to **True**.

DisplayKeysInTooltips Property Example

This example sets the options for all command bars in Microsoft Office.

```
With CommandBars
    .LargeButtons = True
    .DisplayTooltips = True
    .DisplayKeysInTooltips = True
    .MenuAnimationStyle = msoMenuAnimationUnfold
End With
```

MenuAnimationStyle Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":":ofproMenuAnimationStyleC"} {ewc HLP95EN.DLL, DYNALINK, "Example":":ofproMenuAnimationStyleX":":1"} {ewc HLP95EN.DLL, DYNALINK, "Applies To":":ofproMenuAnimationStyleA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":":ofproMenuAnimationStyleS"}

Returns or sets the way the specified **command bar** is animated. Can be one of the following **msoMenuAnimation** types: **msoMenuAnimationNone**, **msoMenuAnimationRandom**, **msoMenuAnimationUnfold**, or **msoMenuAnimationSlide**. Read/write.

MenuAnimationStyle Property Example

This example sets the options for all command bars in Microsoft Office.

```
With CommandBars
    .LargeButtons = True
    .DisplayTooltips = True
    .DisplayKeysInTooltips = True
    .MenuAnimationStyle = msoMenuAnimationUnfold
End With
```

ActionControl Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproActionControlC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproActionControlX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproActionControlA"}
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproActionControlS"}

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.

ActionControl Property Example

This example disables the command bar control that initiated the running procedure while a series of statements runs, and then enables the control.

```
Set theCtrl = CommandBars.ActionControl
theCtrl.Enabled = False
    'insert OnAction process here
theCtrl.Enabled = True
```

Context Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproContextC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofproContextX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproContextA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproContextS"}

Returns or sets a string that determines where the specified command bar 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.

Type Property





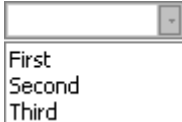
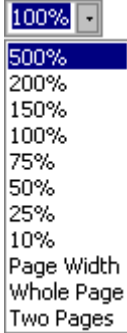

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproTypeC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofproTypeX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproTypeA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproTypeS"}

DocumentProperty object: Returns or sets the document property type. Can be one of the following **MsoDocProperties** constants: **msoPropertyTypeBoolean**, **msoPropertyTypeDate**, **msoPropertyTypeFloat**, **msoPropertyTypeNumber**, or **msoPropertyTypeString**. Read-only for built-in document properties; read/write for custom document properties.

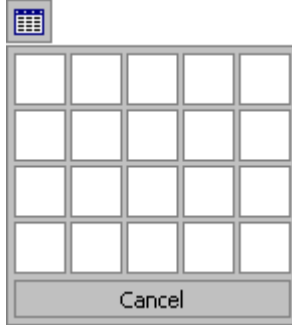
CommandBar object: Returns the type of command bar. Can be one of the following **MsoBarType** constants: **msoBarTypeNormal**, **msoBarTypeMenuBar**, or **msoBarTypePopup**. Read-only **Long**.

CommandBarControl object: Returns the type of command bar control. Read-only **Long**.

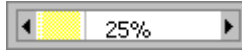
Can be one of the following **MsoControlType** constants.

Constant	Control type
msoControlButton	
msoControlButtonDropdown	 
msoControlButtonPopup	
msoControlComboBox	
msoControlCustom	Reserved for future use.
msoControlDropdown	
msoControlEdit	

msoControlExpandingGrid



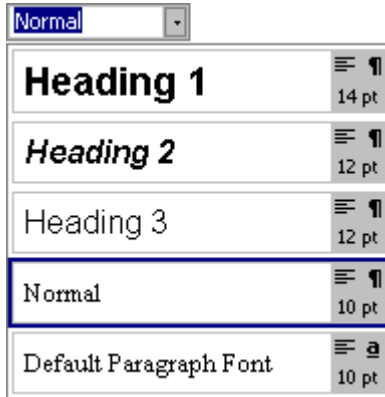
msoControlGauge



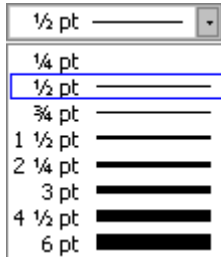
msoControlGenericDropdown

Reserved for future use.

msoControlGraphicCombo



msoControlGraphicDropdown



msoControlGraphicPopup

Reserved for future use.

msoControlGrid



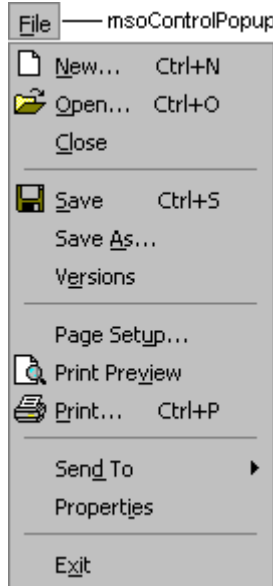
msoControlLabel

Reserved for future use.

msoControlOCXDropDown



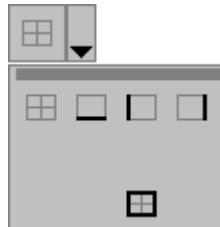
msoControlPopup



msoControlSplitButtonMRUPopup



msoControlSplitButtonPopup



msoControlSplitDropdown



msoControlSplitExpandingGrid

Reserved for future use.

Type Property Example

This example finds the first control with the tag value "Change this bar." If the control with that tag value is a command bar button, the example adds a new combo box control to the end of the command bar and changes the tag value of the found button to "Changed."

```
Set ctrl = CommandBars _  
    .FindControl(Tag:="Change this bar")  
If ctrl Is Nothing Then Goto notFound  
If ctrl.Type = msoControlButton Then  
    Set br = ctrl.Parent  
br.Controls.Add(Type:=msoControlComboBox)  
    ctrl.Tag = "Changed"  
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
```

Visible Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproVisibleC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproVisibleX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproVisibleA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproVisibleS"}

CommandBar and **CommandBarControl** object: **True** if the specified command bar or command bar control is visible. Read/write **Boolean**.

Assistant object: **True** if the Office Assistant is visible. Read-write **Boolean**.

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**.

Visible Property Example

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
```


Width Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproWidthC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproWidthX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproWidthA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproWidthS"}

Returns or sets the width (in pixels) of the specified command bar or command bar control.
Read/write **Long**.

Width Property Example

This example adds a custom control before the second control on the command bar named "Custom." The example sets the height of the custom control to half the height of the command bar and sets its width to 50 pixels.

```
Set myBar = cmdbar.CommandBars("Custom")
barheight = myBar.Height
Set myControl = myBar.Controls
    .Add(Type:=msoControlComboBox, Id:=4, Before:=2, Temporary:=True)
With myControl
    .Height = barheight / 2
    .Width = 50
End With
myBar.Visible = True
```

Application Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproApplicationC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproApplicationX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproApplicationA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproApplicationS"}

Returns an **Application** object that represents the container application for the specified object (you can use this property with an Automation object to return that object's container application). Read-only.

Application Property Example

This example returns the application in which the command bar named "Documents" was created.

```
Set Appobj = CommandBars("Document").Application
```

Creator Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproCreatorC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproCreatorX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproCreatorA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproCreatorS"}

Returns the four-character code for the application in which the specified object was created. Macintosh only. Read-only **Long**.

Parent Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproParentC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofproParentX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproParentA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproParents"}

Returns the parent object for the specified object. Read-only.

Parent Property Example

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
```

ShortcutText Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproShortcutTextC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofproShortcutTextX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproShortcutTextA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproShortcutTextS"}

Returns or sets the shortcut key text displayed next to the specified button control when that button appears on a menu, submenu, or shortcut menu. Read/write **String**.

Remarks

You can set this property only for command bar buttons that have an attached **OnAction** macro.

ShortcutText Property Example

This example places a charting button on the command bar named "Custom" and assigns an **OnAction** macro to the button, along with a shortcut key and a caption.

```
Set myControl = CommandBars("Custom").Controls _  
    .Add(Type:=msoControlButton, Id:= 17)  
With myControl  
    .OnAction = "MySub"  
    .Caption = "Graph Results"  
    .Control.ShortcutText = "CTRL+SHIFT+M"  
End With
```

Height Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproHeightC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproHeightX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproHeightA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproHeightS"}

Returns or sets the height of the specified command bar or command bar control, in pixels.

Read/write **Long**.

Remarks

Setting the **Height** property will cause an error if the command bar isn't in a resizable state (that is, if it's docked or protected from resizing).

Height Property Example

This example adds a custom control before the second control on the command bar named "Custom." The example sets the height of the control to half the height of the command bar and sets its width to 50 pixels.

```
Set myBar = cmdbar.CommandBars("Custom")
barHeight = myBar.Height
Set myControl = myBar.Controls
    .Add(Type:=msoControlComboBox, Before:=2, Temporary:=True)
With myControl
    .Height = barHeight / 2
    .Width = 50
End With
myBar.Visible = True
```

Id Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproldC"}
{ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproldA"}

{ewc HLP95EN.DLL, DYNALINK, "Example":"ofproldX":1}
{ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproldS"}

Returns the ID for the specified built-in command bar control. Read-only **Long**.

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.

Id Property Example

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("Custom2").Controls(1)
With ctrl
    If .Id < 25 Then
        .FaceId = 17
        .Tag = "Changed control"
    End If
End With
```

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
```

Index Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproIndexC"} {ewc HLP95EN.DLL, DYNALINK,
"Example":"ofproIndexX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproIndexA"} {ewc
HLP95EN.DLL, DYNALINK, "Specifics":"ofproIndexS"}

Returns the index number of the specified object in the collection.. Read-only **Long**.

Remarks

The position of the first command bar control is 1. Separators are not counted in the **CommandBarControls** collection.

Index Property Example

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 found 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
```

Item Property

Returns an object from a collection, or returns text associated with an object.

Item is the default member of the object or collection. For example, the following two statements both assign a **CommandBar** object to `cmdBar`.

```
Set cmdBar = CommandBars.Item("Standard")
Set cmdBar = CommandBars("Standard")
```

The following two statements both assign the text of the first label in the **Balloon** object assigned to `myBalloon` to `lblText`.

```
lblText = myBalloon.Labels(1).Item
lblText = myBalloon.Labels(1)
```

Select one of the following objects to see a detailed description of the **Item** property for that object.

Assistant

BalloonCheckboxes

BalloonCheckbox

BalloonLabel

BalloonLabels

CommandBarControls

CommandBars

DocumentProperties

FileFindResults

FoundFiles

PropertyTests

Item Property (CommandBars and CommandBarControls Collections)

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproItemCommandBarsObjC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofproItemCommandBarsObjX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproItemCommandBarsObjA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproItemCommandBarsObjS"}

Returns a **CommandBar** object from a **CommandBars** collection, or returns a **CommandBarControl** object from a **CommandBarControls** collection. Read-only.

Syntax

expression.Item(*Index*)

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

Index Required **Variant**. The name or index number of the object to be returned. Note that you can only use index numbers to return **PropertyTest** objects from the **PropertyTests** collection.

Item Property (BalloonCheckboxes Collection)

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproItemBalloonCheckboxesObjC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproItemBalloonCheckboxesObjX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproItemBalloonCheckboxesObjA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproItemBalloonCheckboxesObjS"}

Returns a **BalloonCheckbox** object from a **BalloonCheckboxes** collection. Read-only.

Syntax

expression.Item(*iCbx*)

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

iCbx Required **Long**. The index number of the check box to be returned.

Item Property (BalloonLabels Collection)

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproItemBalloonLabelsObjC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproItemBalloonLabelsObjX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproItemBalloonLabelsObjA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproItemBalloonLabelsObjS"}

Returns a **BalloonLabel** object from a **BalloonLabels** collection. Read-only.

Syntax

expression.Item(*iLbl*)

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

iLbl Required **Long**. The index number of the label to be returned.

Item Property (Assistant, BalloonLabel, and BalloonCheckbox Objects)

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproItemAssistantObjC"}           {ewc HLP95EN.DLL, DYNALINK,  
"Example":"ofproItemAssistantObjX":1}           {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproItemAssistantObjA"}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproItemAssistantObjS"}
```

Returns the text associated with the specified object. Read-only **String**.

Syntax

expression.**Item**

expression Required. An expression that returns an **Assistant**, **BalloonCheckbox**, or **BalloonLabel** object.

Item Property (DocumentProperties Collection)

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproItemDocumentPropertiesObjC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofproItemDocumentPropertiesObjX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproItemDocumentPropertiesObjA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproItemDocumentPropertiesObjS"}

Returns an **DocumentProperty** object from a **DocumentProperties** collection. Read-only.

Syntax

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

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

Index Required **Variant**. The name or index number of the document property to be returned.

Item Property (PropertyTests Collection)

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproItemPropertyTestsObjC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproItemPropertyTestsObjX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproItemPropertyTestsObjA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproItemPropertyTestsObjS"}

Returns an **PropertyTest** object from a **PropertyTests** collection. Read-only.

Syntax

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

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

Index Required **Variant**. The index number of the property test to be returned.

Item Property (FoundFiles and FileFindResults Objects)

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproItemFoundFilesObjC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproItemFoundFilesObjX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproItemFoundFilesObjA"}
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproItemFoundFilesObjS"}

Returns a file name from the list of file names represented by the **FoundFiles** or **FileFindResults** object. Read-only **String**.

Syntax

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

expression Required. An expression that returns a **FoundFiles** or **FileFindResults** object (**FileFindResults** is Macintosh only).

Index Required **Variant**. The index number of the file name to be returned.

Left Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproLeftC"}
{ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproLeftA"}

{ewc HLP95EN.DLL, DYNALINK, "Example": "ofproLeftX": 1}
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproLeftS"}

CommandBar or **CommandBarControl** object: Returns or sets the distance (in pixels) from the left edge of the specified command bar or command bar control to the left edge of the screen. Returns the distance from the left side of the docking area. Read/write **Long** for **CommandBar**, read-only **Long** for **CommandBarControl**.

Assistant object: Sets or returns the horizontal position of the Office Assistant window (in points), relative to the screen. Read/write **Long**.

Left Property Example

This example adjusts the position of the custom command bar named "Custom" by moving it to the left 110 pixels more than the default. The example also makes this command bar the first one to be docked by changing the row index number to 1.

```
Set myBar = CommandBars("Custom")
With myBar
    .RowIndex = 1
    .Left = 140
End With
```

This example displays the Office Assistant and moves it to the specified position.

```
With Assistant
    .Visible = True
    .Left = 300
    .Top = 300
End With
```

Move Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthMoveC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthMoveX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthMoveA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthMoveS"}

CommandBarControl object: Moves the specified command bar control to an existing command bar.

Assistant object: Moves the Office Assistant to the specified location.

Syntax 1

expression.**Move**(**Bar**, **Before**)

Syntax 2

expression.**Move**(**xLeft**, **yTop**)

expression Syntax 1: Required. An expression that returns a **CommandBarControl**, **CommandBarButton**, **CommandBarPopup**, or **CommandBarComboBox** object.

Syntax 2: Required. An expression that returns an **Assistant** object.

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 same command bar (the command bar that the control currently resides on).

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.

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.

Move Method Example

This example moves the first combo box control from the custom 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
```

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
```

Name Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproNameC"} {ewc HLP95EN.DLL, DYNALINK,
"Example":"ofproNameX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproNameA"} {ewc
HLP95EN.DLL, DYNALINK, "Specifics":"ofproNameS"}

Returns or sets the name of the specified object. Read/write **String** for the **CommandBar** object, read-only **String** for all other objects.

Remarks

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.

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.

Name Property Example

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
```

command bar

A programmable object that you use in Visual Basic to control a menu or toolbar. All the following items are represented by **CommandBar** objects:

- Menu bars, toolbars, and shortcut menus
- Menus on menu bars and toolbars
- Submenus on menus, submenus, and shortcut menus

command bar control

A built-in or custom control on a menu bar, toolbar, menu, submenu, or shortcut menu. Custom controls you can add to command bars include buttons, edit boxes, drop-down list boxes, combo boxes, and pop-up controls (controls that display a menu or submenu).

button control

A button on a toolbar or a menu item on a menu, submenu, or shortcut menu that runs a command when it's clicked. Toolbar buttons and menu items share the same properties and methods.

On a toolbar, a button control can be displayed as an icon only, an icon and a caption, or a caption only. On a menu, a button control can be displayed either as an icon and a caption or as a caption only.

combo box control

A custom edit box, drop-down list box, or combo box on a menu bar, toolbar, menu, submenu, or shortcut menu. A custom combo box control can be displayed with or without a label. When a toolbar is docked vertically, any custom combo box controls that it contains aren't visible.

Many built-in controls, such as the **Undo** button, are also considered to be combo box controls; however, most properties and methods for modifying combo box controls aren't available for those built-in controls.

pop-up control

A built-in or custom control on a menu bar or toolbar that displays a menu when it's clicked; or a built-in or custom menu item on a menu, submenu, or shortcut menu that displays a submenu when the pointer is positioned over it.

Many built-in controls, such as the **Font Color** button, are also considered to be pop-up controls; however, most properties and methods for modifying pop-up controls aren't available for those built-in controls.

CommandBars Collection Object

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofobjCommandBarsC "} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofobjCommandBarsX":1} {ewc HLP95EN.DLL, DYNALINK, "Properties":"ofobjCommandBarsP "} {ewc HLP95EN.DLL, DYNALINK, "Methods":"ofobjCommandBarsM "} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofobjCommandBarsS "}

CommandBars (CommandBar)

Assistant

CommandBarControls (CommandBarControl)

Assistant

Assistant

CommandBarButton

Assistant

Assistant

CommandBarComboBox

Assistant

Assistant

CommandBarPopup

A collection of **CommandBar** objects that represent the command bars 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 **Add** 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:=msoBarFloating)
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 pop-up control that displays that menu. Then apply the **CommandBar** property to the pop-up control to return the command bar that represents that menu.

CommandBar Object

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofobjCommandBarC "} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofobjCommandBarX": 1} {ewc HLP95EN.DLL, DYNALINK, "Properties": "ofobjCommandBarP "} {ewc HLP95EN.DLL, DYNALINK, "Methods": "ofobjCommandBarM "} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofobjCommandBarS "}
```

Assistant

Assistant
Assistant

Assistant
Assistant
CommandBarButton

Assistant
Assistant
CommandBarComboBox

Assistant
Assistant
CommandBarPopup

Represents a **command bar** in the container application. The **CommandBar** object is a member of the **CommandBars** 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:=msoControlButton)
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 pop-up control that displays that menu. Then apply the **CommandBar** 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 pop-up control, this example adds the **Save** command to the bottom of that menu.

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

CommandBarControls Collection Object

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofobjCommandBarControlsC "} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofobjCommandBarControlsX": 1} {ewc HLP95EN.DLL, DYNALINK, "Properties": "ofobjCommandBarControlsP "} {ewc HLP95EN.DLL, DYNALINK, "Methods": "ofobjCommandBarControlsM "} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofobjCommandBarControlsS" }
```

Assistant

Assistant
Assistant

Assistant
Assistant
Assistant

Assistant
Assistant
Assistant

Assistant
Assistant
CommandBarPopup

A collection of **CommandBarControl** objects that represent the command bar controls on a command bar.

Using the CommandBarControls Collection

Use the **Controls** 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 **Add** 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
```

CommandBarControl Object

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofobjCommandBarControlC "} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofobjCommandBarControlX": 1} {ewc HLP95EN.DLL, DYNALINK, "Properties": "ofobjCommandBarControlP"} {ewc HLP95EN.DLL, DYNALINK, "Methods": "ofobjCommandBarControlM "} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofobjCommandBarControlS "}
```

Assistant

Assistant
Assistant

Assistant
Assistant
Assistant

Assistant
Assistant
Assistant

Assistant
Assistant
Assistant

Represents a command bar control. The **CommandBarControl** object is a member of the **CommandBarControls** collection. The properties and methods of the **CommandBarControl** object are all shared by the **CommandBarButton**, **CommandBarComboBox**, and **CommandBarPopup** 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**, **msoControlGrid**, 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 *lbl* 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
```


CommandBarButton Object

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofobjCommandBarButtonC "} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofobjCommandBarButtonX": 1} {ewc HLP95EN.DLL, DYNALINK, "Properties": "ofobjCommandBarButtonP"} {ewc HLP95EN.DLL, DYNALINK, "Methods": "ofobjCommandBarButtonM "} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofobjCommandBarButtonS "}
```

Assistant

Assistant
Assistant

Assistant
Assistant
Assistant

Assistant
Assistant
Assistant

Assistant
Assistant
Assistant

Represents a button control on a command bar.

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

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofobjCommandBarComboBoxC "} {ewc HLP95EN.DLL, DYNALINK,
"Example": "ofobjCommandBarComboBoxX": 1} {ewc HLP95EN.DLL, DYNALINK,
"Properties": "ofobjCommandBarComboBoxP "} {ewc HLP95EN.DLL, DYNALINK,
"Methods": "ofobjCommandBarComboBoxM "} {ewc HLP95EN.DLL, DYNALINK,
"Specifics": "ofobjCommandBarComboBoxS "}

Assistant

Assistant
Assistant

Assistant
Assistant
Assistant

Assistant
Assistant
Assistant

Assistant
Assistant
Assistant

Represents a combo box control on a command bar.

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.

```
Set combo = CommandBars("Custom").Controls(2)
With combo
    .AddItem "First Item", 1
    .AddItem "Second Item", 2
    .DropDownLines = 3
    .DropDownWidth = 75
    .ListIndex = 0
End With
```

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)
```

CommandBarPopup Object

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofobjCommandBarPopUpC "} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofobjCommandBarPopUpX":1} {ewc HLP95EN.DLL, DYNALINK, "Properties": "ofobjCommandBarPopUpP"} {ewc HLP95EN.DLL, DYNALINK, "Methods": "ofobjCommandBarPopUpM "} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofobjCommandBarPopUpS "}
```

Assistant

Assistant
Assistant

Assistant
Assistant
Assistant

Assistant
Assistant
Assistant

Assistant
Assistant
Assistant

Represents a pop-up control on a command bar.

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**, or **msoControlSplitButtonMRUPopup**.)

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.

FileType Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproFileTypeC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproFileTypeX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproFileTypeA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproFileTypeS"}

FileSearch object: Returns or sets the type of file to be searched for during a file search. Can be one of the following **MsoFileType** constants: **msoFileTypeAllFiles**, **msoFileTypeBinders**, **msoFileTypeDatabases**, **msoFileTypeExcelWorkbooks**, **msoFileTypeOfficeFiles**, **msoFileTypePowerPointPresentations**, **msoFileTypeTemplates**, or **msoFileTypeWordDocuments**. The default value is **msoFileTypeOfficeFiles**. Read/write **Variant**.

FileFind object: Returns or sets the type of file to be searched for. Can be set to a number returned by the **MacId** function. Macintosh only. Read/write **Long**.

Remarks

Use the arguments listed in the following table with the **MacID** function to return the appropriate Macintosh file type.

File type	Argument
Word document	MSWD
Microsoft Excel workbook	XCEL
PowerPoint presentation	PPT3 (for use with all versions of PowerPoint)
Text file	TTXT

FileType Property Example

This example searches for all Binder files located in the My Documents folder. The example displays the name and location of each found file in a message dialog.

```
Set fs = Application.FileSearch
With fs
    .LookIn = "C:\My Documents"
    .FileType = msoFileTypeBinders
    If .Execute > 0 Then
        MsgBox "There were " & .FoundFiles.Count & _
            " Binder file(s) found."
        For i = 1 To .FoundFiles.Count
            MsgBox .FoundFiles(i)
        Next i
    Else
        MsgBox "There were no Binder files found."
    End If
End With
```

FoundFiles Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproFoundFilesC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproFoundFilesX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproFoundFilesA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproFoundFilesS"}

Returns a **FoundFiles** object that contains the names of all files found in a search operation. Read-only.

FoundFiles Property Example

This example steps through the list of found files and displays the path for each file.

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

LastModified Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproLastModifiedC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproLastModifiedX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproLastModifiedA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproLastModifiedS"}

Returns or sets a constant that represents the amount of time since the specified file was last modified and saved. Can be one of the following **msoLastModified** constants: **msoLastModifiedAnyTime**, **msoLastModifiedLastMonth**, **msoLastModifiedLastWeek**, **msoLastModifiedThisMonth**, **msoLastModifiedThisWeek**, **msoLastModifiedToday**, or **msoLastModifiedYesterday**. The default value is **msoLastModifiedAnyTime**. Read/write **Variant**.

LastModified Property Example

This example sets options for a file search. The files that this search will return have been previously modified and are located in the C:\My Documents folder or in a subfolder of that folder.

```
Set fs = Application.FileSearch
With fs
    .LookIn = "C:\My Documents"
    .SearchSubFolders = True
    .LastModified = msoLastModifiedYesterday
End With
```

LookIn Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproLookInC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproLookInX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproLookInA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproLookInS"}

Returns or sets the folder to be searched during the specified file search. Read/write **String**.

LookIn Property Example

This example searches the My Documents folders for all files whose names 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
```

MatchAllWordForms Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproMatchAllWordFormsC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproMatchAllWordFormsX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproMatchAllWordFormsA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproMatchAllWordFormsS"}

True if the specified file search will be expanded to include all forms of the specified word 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.

MatchAllWordForms Property Example

This example returns all files that contain the word "run," "running," "runs," or "ran" in the body of the file, or in the file properties. 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

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproMatchTextExactlyC"} {ewc HLP95EN.DLL, DYNALINK,
"Example":"ofproMatchTextExactlyX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproMatchTextExactlyA"}
{ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproMatchTextExactlyS"}

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**.

MatchTextExactly Property Example

This example searches the C:\My Documents folder and returns all files that contain the word "Run" either in their body text or in their file properties.

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

NewSearch Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthNewSearchC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthNewSearchX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthNewSearchA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthNewSearchS"}

Resets all the search criteria settings to their default settings.

Syntax

expression.**NewSearch**

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

Remarks

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.

NewSearch Method Example

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"
    .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
```

PropertyTests Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproPropertyTestsC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproPropertyTestsX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproPropertyTestsA"}
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproPropertyTestsS"}

Returns a **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 [Returning an Object from a Collection](#).

PropertyTests Property Example

This example displays all the search criteria for the first property test in the 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
MsgBox myString
End With
```

SearchSubFolders Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproSearchSubFoldersC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproSearchSubFoldersX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproSearchSubFoldersA"}
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproSearchSubFoldersS"}

True if the specified search includes all the subfolders in the folder specified by the LookIn property.
Read/write **Boolean**.

SearchSubFolders Property Example

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
```

TextOrProperty Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproTextOrPropertyC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproTextOrPropertyX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproTextOrPropertyA"}
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproTextOrPropertyS"}

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 specified 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 t "San."

TextOrProperty Property Example

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
```

Connector Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproConnectorC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproConnectorX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproConnectorA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproConnectorS"}

Returns the connector between two similar property test values. Can be either of the following **MsoConnector** constants: **msoConnectorAnd** or **msoConnectorOr**. The default value is **msoConnectorAnd**. Read-only **Variant**.

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**).

Connector Property Example

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
```

Condition Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproConditionC"} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproConditionX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproConditionA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproConditionS"}

Returns the condition of the specified search criteria. Read-only **Variant**.

Can be one of the following **MsoCondition** constants:

msoConditionAnyNumberBetween	msoConditionInTheNext
msoConditionAnytime	msoConditionIsExactly
msoConditionAnytimeBetween	msoConditionIsNo
msoConditionAtLeast	msoConditionIsNot
msoConditionAtMost	msoConditionIsYes
msoConditionBeginsWith	msoConditionLastMonth
msoConditionDoesNotEqual	msoConditionLastWeek
msoConditionEndsWith	msoConditionLessThan
msoConditionEquals	msoConditionMoreThan
msoConditionFileTypeAllFiles	msoConditionNextMonth
msoConditionFileTypeBinders	msoConditionNextWeek
msoConditionFileTypeDatabases	msoConditionOn
msoConditionFileTypeExcelWorkbooks	msoConditionOnorAfter
msoConditionFileTypeOfficeFiles	msoConditionOnorBefore
msoConditionFileTypePowerPointPresentations	msoConditionThisMonth
msoConditionFileTypeTemplates	msoConditionThisWeek
msoConditionFileTypeWordDocuments	msoConditionToday
msoConditionIncludesNearEachOther	msoConditionTomorrow
msoConditionIncludesPhrase	msoConditionYesterday
msoConditionInTheLast	

Condition Property Example

This example returns the connection value for search criteria for the first property test.

```
With Application.FileSearch.PropertyTests(1)
    MsgBox "The condition you've set is: " & .Condition
End With
```

SecondValue Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproSecondValueC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofproSecondValueX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproSecondValueA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproSecondValueS"}

Returns an optional second value property test (as in a range) for the specified 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** constants **msoConditionAnyTimeBetween** or **msoConditionAnyNumberBetween**.

SecondValue Property Example

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
```

Remove Method

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmthRemoveC"} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofmthRemoveX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofmthRemoveA"} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofmthRemoveS"}

Removes a **PropertyTest** object from the **PropertyTests** collection.

Syntax

expression.**Remove**(*Index*)

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

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

Remove Method Example

This example removes the first search criterion from the collection.

```
Application.FileSearch.PropertyTests.Remove(1)
```

FileSearch Object

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofobjFileSearchC "} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofobjFileSearchX":1} {ewc HLP95EN.DLL, DYNALINK, "Properties":"ofobjFileSearchP "} {ewc HLP95EN.DLL, DYNALINK, "Methods":"ofobjFileSearchM "} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofobjFileSearchS "}

FileSearch

Assistant

PropertyTests {PropertyTest}

Assistant

FoundFiles

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 the specified files and displays both the number of files found and the title of each found 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
```


PropertyTest Object

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofobjPropertyTestC "} {ewc HLP95EN.DLL, DYNALINK,  
"Example":"ofobjPropertyTestX":1} {ewc HLP95EN.DLL, DYNALINK, "Properties":"ofobjPropertyTestP "} {ewc  
HLP95EN.DLL, DYNALINK, "Methods":"ofobjPropertyTestM "} {ewc HLP95EN.DLL, DYNALINK,  
"Specifics":"ofobjPropertyTestS "}
```

FileSearch

Assistant

PropertyTests {PropertyTest}

Assistant

FoundFiles

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  
MsgBox myString  
End With
```

PropertyTests Collection Object

```
{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofobjPropertyTestsC "} {ewc HLP95EN.DLL, DYNALINK,  
"Example":"ofobjPropertyTestsX":1} {ewc HLP95EN.DLL, DYNALINK, "Properties":"ofobjPropertyTestsP "}  
{ewc HLP95EN.DLL, DYNALINK, "Methods":"ofobjPropertyTestsM "} {ewc HLP95EN.DLL, DYNALINK,  
"Specifics":"ofobjPropertyTestsS "}
```

Assistant

Assistant

Assistant

Assistant

FoundFiles

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.

```
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 found files must be Word documents, and the second one specifies that the found files must have been modified between January 1, 1996 and June 30, 1996.

```
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:="1/1/96", SecondValue:="6/30/96", _  
    Connector:=msoConnectorAnd  
End With
```

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 With
```

```
End If
MsgBox myString
End With
```

FoundFiles Object

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofobjFoundFilesC "} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofobjFoundFilesX":1} {ewc HLP95EN.DLL, DYNALINK, "Properties":"ofobjFoundFilesP "} {ewc HLP95EN.DLL, DYNALINK, "Methods":"ofobjFoundFilesM "} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofobjFoundFilesS "}

Assistant

Assistant
Assistant

Assistant
Assistant

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 found files and displays the path and file name of each found file.

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

Use **FoundFiles(index)**, where *index* is the index number, to return the path and file name of a found file. The following example steps through the collection of found files and displays the path and file name of each one.

```
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
```


DocumentProperties Collection Object

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofobjDocumentPropertiesC "}          {ewc HLP95EN.DLL, DYNALINK, "Example": "ofobjDocumentPropertiesX": 1}          {ewc HLP95EN.DLL, DYNALINK, "Properties": "ofobjDocumentPropertiesP"}          {ewc HLP95EN.DLL, DYNALINK, "Methods": "ofobjDocumentPropertiesM "}          {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofobjDocumentPropertiesS "}
```

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 **BuiltinDocumentProperties** property to return a **DocumentProperties** collection that contains all the built-in properties of a container document. Use the **CustomDocumentProperties** property to return a **DocumentProperties** collection that contains all the custom properties of the document.

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

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofobjDocumentPropertyC "} {ewc HLP95EN.DLL, DYNALINK,
"Example": "ofobjDocumentPropertyX":1} {ewc HLP95EN.DLL, DYNALINK, "Properties": "ofobjDocumentPropertyP "}
{ewc HLP95EN.DLL, DYNALINK, "Methods": "ofobjDocumentPropertyM "} {ewc HLP95EN.DLL, DYNALINK,
"Specifics": "ofobjDocumentPropertyS "}
```

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 names of all the available built-in document properties are shown in the following list:

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 Pages	Number of Multimedia Clips

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.

LinkToContent Property

```
{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproLinkToContentC "} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproLinkToContentX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproLinkToContentA "}  
{ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproLinkToContentS "}
```

True if the value of the specified 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, this property is always **False**.

Use the **LinkSource** property to set the source for the specified linked property. Setting the **LinkSource** property sets the **LinkToContent** property to **True**.

LinkToContent Property Example

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.LinkSource
    End If
    MsgBox stat
End Sub
```

LinkSource Property

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofproLinkSourceC "} {ewc HLP95EN.DLL, DYNALINK, "Example":"ofproLinkSourceX":1} {ewc HLP95EN.DLL, DYNALINK, "Applies To":"ofproLinkSourceA "} {ewc HLP95EN.DLL, DYNALINK, "Specifics":"ofproLinkSourceS "}

Returns or sets the source of the specified 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**.

LinkSource Property Example

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.LinkSource
    End If
    MsgBox stat
End Sub
```

Value Property

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofproValueC "} {ewc HLP95EN.DLL, DYNALINK, "Example": "ofproValueX": 1} {ewc HLP95EN.DLL, DYNALINK, "Applies To": "ofproValueA "} {ewc HLP95EN.DLL, DYNALINK, "Specifics": "ofproValueS "}

DocumentProperty object: Returns or sets the value of the specified document property. Read/write **VARIANT**.

PropertyTest object: Returns the value of a property test for a file search. Read-only **VARIANT**.

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.

Value Property Example

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
```

This example displays the 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 .Value = "" Then
        MsgBox "You haven't specified a value."
    Else
        MsgBox "The value you've set is: " & _
            & .Value
    End If
End With
```

Item not available

The item is not available on this release of Office. It may be that your language version doesn't support this item, or that the item is designed for non-Windows versions only.

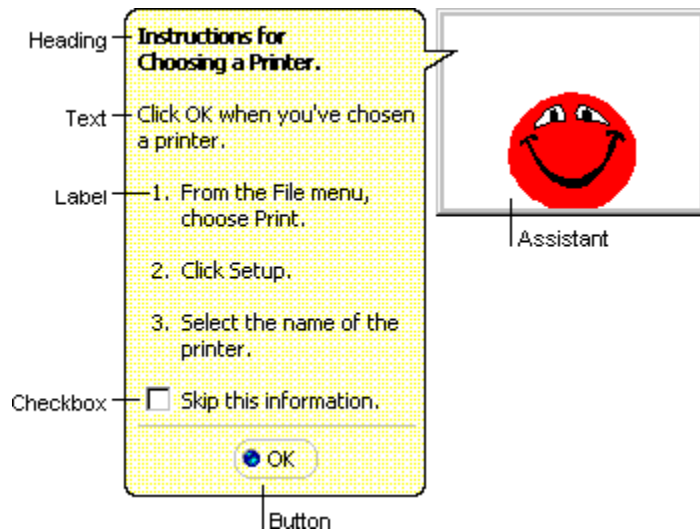
Overview of the Office Assistant

{ewc HLP95EN.DLL, DYNALINK, "See Also": "ofhowOverviewAssistantC"}
"Specifics": "ofhowOverviewAssistantS"}

{ewc HLP95EN.DLL, DYNALINK,

Microsoft Office 97 introduces a new type of Help component: the Office Assistant. The Office Assistant, the Office Assistant balloon, and all the items inside the balloon are controlled programmatically through the objects, properties, and methods of the **Assistant** object.

The following illustration shows the Office Assistant, the Office Assistant balloon, and all the programmable objects inside the balloon.



For information about how you can make design-time and run-time changes to the Office Assistant, see the following topics.

[Using the Office Assistant](#)

[Creating and modifying balloons](#)

[Making run-time modifications to balloons](#)

Using the Office Assistant

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofhowUsingAssistantC"}
"Specifics":"ofhowUsingAssistantS"}

{ewc HLP95EN.DLL, DYNALINK,

You can use the Office Assistant in your application to offer help in specific areas, and you can use the controls in the Office Assistant balloon to run your own procedures.

Everything associated with the Office Assistant is represented in Visual Basic by the **Assistant** object and the objects it contains. Many of the changes you can make to the Assistant at run time are the same changes you can make at design time by right-clicking the Assistant. Each of the options listed on the **Options** tab in the **Office Assistant** dialog box corresponds to a property of the Assistant that you can set at run time. The properties listed in the following table are commonly used to modify the Office Assistant.

Property	Description
<u>Animation</u>	Animates the Office Assistant. Some animations are continuous; others occur only once.
<u>FileName</u>	Specifies the file name for the Office Assistant, complete with the .act extension. For example, the file name for the Clippit Office Assistant is Clippit.act.
<u>Name</u>	Returns the name of the Office Assistant. If you want to keep track of which Office Assistant the user selects, use this property.
<u>Visible</u>	Specifies whether the Assistant is visible to or hidden.

The following example selects Will as the Office Assistant, makes the Assistant visible, and sets an animation. The example also sets several of the customized Help properties that are available on the **Options** tab in the **Office Assistant** dialog box at design time

```
With Assistant
    .FileName = "Will.act"
    .Animation = msoAnimationBeginSpeaking
    .AssistWithHelp = True
    .GuessHelp = True
    .FeatureTips = False
    .Visible = True
End With
```

There are nine different Office Assistant actors you can choose to install with Microsoft Office. Only one Assistant actor can be active at a time. You can let the user select the Assistant at design time, or you can select an Assistant for the user at run time. You cannot modify the installed Assistant actor files, and you cannot substitute your own actor file in place of the files provided.

Creating and modifying balloons

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofhowCreatingBalloonsC"}
"Specifics":"ofhowCreatingBalloonsS"}

{ewc HLP95EN.DLL, DYNALINK,

You use the Office Assistant balloon to provide information for the user. The information inside the balloon can be a simple message, a request for more information, or a list of choices for the user. Changes you make to the Office Assistant balloon must be made at run time.

The following table contains the most commonly used properties and methods for changing either the appearance or function of a newly created (blank) balloon or an existing balloon. Changes you make to any balloon will appear the next time the **Show** method is used.

Property or method	Description
<u>Heading</u>	Specifies the bold text appearing at the top of the Office Assistant balloon.
<u>Text</u>	Specifies the text appearing in the body of the Office Assistant balloon. This text appears after the heading but before any check boxes, labels, or buttons.
<u>Labels</u>	Returns the collection of labels in the balloon. The format of the labels is determined by the BalloonType property. Labels appear after the balloon text. The list of labels can be numbered or bulleted, or it can be a list of buttons. Unlike check boxes, the user's choice is registered as soon as a button is clicked.
<u>Checkboxes</u>	Returns the collection of check boxes in the balloon. The user clicks the check box and then clicks the appropriate button at the bottom of the balloon (for example, OK or Next) to register his or her choice.
<u>Close</u>	Closes and dismisses a modeless balloon, but doesn't release the object variable. The object variable assigned to the balloon is still valid – you can display it again, or you can modify it and display it later. You can use this method only on modeless balloons.
<u>Show</u>	Displays the balloon, and all the objects inside it, to the user. Use this method only for Balloon objects; use the Visible property for the Assistant object.

The following example creates a balloon that helps the user select a printer. The example provides a check box option for users who want to skip the information in the balloon.

```
Set bln = Assistant.NewBalloon
With bln
    .Heading = "Instructions for Choosing a Printer."
    .Text = "Click OK when you've chosen a printer."
    lblTxt = "From the File menu, choose Print."
    .Labels(1).Text = lblTxt
    .Labels(2).Text = "Click Setup."
    .Labels(3).Text = "Select the name of the printer."
    .Checkboxes(1).Text = "Skip this information."
    .BalloonType = msoBalloonTypeNumbers
    .Mode = msoModeModal
    .Button = msoButtonSetOK
End With
```

```
.Show  
End With
```

Creating balloons

To create a new balloon, use the **NewBalloon** property of the **Assistant** object. The balloon you create will be blank. Use the **Heading** property to add a heading, and use the **Text** property to add text to the body of the balloon and then add controls, if needed. Finally, use the **Show** method to display the balloon. The **Show** method will show the balloon as it appears at that moment; therefore, it's important to use this method after setting all the other properties of the balloon. The following example creates a new balloon, sets the heading and body text, and creates three check box controls that the user can select.

```
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
```

Managing multiple balloons

The Office Assistant does not have a **Balloons** collection. To manage multiple balloons, you can set a separate object variable for each balloon you create and refer to the variable as needed. Or you can create an array of **Balloon** object variables and assign a balloon to each one. The following example creates an array and adds three blank **Balloon** objects to the array.

```
Dim myBalloonArray(3) As Balloon  
  
With Assistant  
    For i = 1 To 3  
        Set myBalloonArray(i) = .NewBalloon  
    Next  
End With
```

Making run-time modifications to balloons

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofhowModifyingBalloonsC"}
"Specifics":"ofhowModifyingBalloonsS"}

{ewc HLP95EN.DLL, DYNALINK,

After you create a balloon for the Office Assistant, you can customize it by adding bitmaps, icons, Windows metafiles, or Macintosh PICT files to the balloon's heading or text. You may want to add controls such as check boxes or buttons to your balloon so that you can respond to the user when he or she clicks an item in the balloon.

Adding icons and bitmaps to balloons

To customize your application, you can add icons and bitmaps to Office Assistant balloons. To add an icon, assign an **MsoIconType** constant to the **Icon** property of the **Balloon** object. To add a Windows or Macintosh bitmap, a Windows metafile, or a Macintosh PICT file to the text in an Office Assistant balloon, specify the type, the location, and the sizing factor (if applicable) when you set the **Text** property of a heading, text, check box, or label. The following example inserts a Windows bitmap file into the text of a balloon.

```
myBmp = "{bmp c:\windows\circles.bmp}"
myText1 = "This is before the picture, "
myText2 = " and this is after the picture"
Set bln = Assistant.NewBalloon
With bln
    .Heading = "Instructions for Choosing a Bitmap."
    .Text = myText1 & myBmp & myText2
    .Show
End With
```

Adding controls to balloons

There are two types of controls you can add to a balloon: check boxes and buttons. There are five check boxes in the balloon when it's created; you can make any one of them visible by specifying text for the control. There are five label buttons in the balloon as well (if, that is, the balloon type is **msoBalloonTypeButtons**), and you can expose any one of them in the same way you would a check box. If you try to add more than five check boxes or five label buttons to a balloon, an error occurs.

You can change the appearance of any control on a balloon by changing the control's **Text** property. You can change the functionality of a check box or a button in a balloon by specifying another procedure that will be run whenever the check box or button is selected (clicked).

The following example creates a balloon with a heading, text, and three region choices. When the user selects a check box and then clicks **OK**, the appropriate procedure is run.

```
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
Select Case True
    Case .CheckBoxes(1).Checked
        runregion1
    Case .CheckBoxes(2).Checked
        runregion2
    Case .CheckBoxes(3).Checked
        runregion3
End Select
```

End With

Overview of command bars

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofhowOverviewOfCommandBarsC"}
"Specifics":"ofhowOverviewOfCommandBarsS"}

{ewc HLP95EN.DLL, DYNALINK,

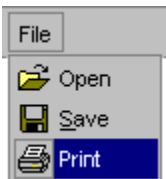
In Microsoft Office 97, toolbars, menu bars, and shortcut menus are all controlled programmatically as one type of object: command bars. All the following items are represented in Visual Basic by **CommandBar** objects:

- Menu bars, toolbars, and shortcut menus
- Menus on menu bars and toolbars
- Submenus on menus, submenus, and shortcut menus

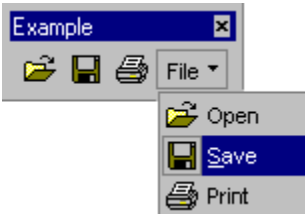
You can modify any built-in menu bar or toolbar, and you can create and modify custom toolbars, menu bars, and shortcut menus to deliver with your Visual Basic application. You present the features of your application as individual buttons on toolbars or as groups of command names on menus. Because toolbars and menus are both command bars, you use the same kind of controls on both of them. For example, the docked toolbar shown in the following illustration contains three buttons.



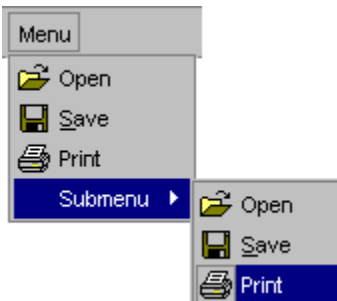
The menu shown in the following illustration contains the same three commands as in the previous illustration, but here they're displayed as menu items.



In Microsoft Office 97, menu bars and toolbars can both contain menus. The floating toolbar shown in the following illustration contains three buttons, and it also contains a menu with the same three commands displayed as menu items.



In Visual Basic, buttons and menu items are represented by **CommandBarButton** objects. The pop-up controls that display menus and submenus are represented by **CommandBarPopUp** objects. In the following illustration, the control named "Menu" and the control named "Submenu" are both pop-up controls that display a menu and a submenu, respectively. Both the menu and the submenu are unique **CommandBar** objects with their own set of controls.



In Microsoft Office 97, you can also add text boxes, drop-down list boxes, and combo boxes to any command bar. These three types of controls are all represented in Visual Basic by

CommandBarComboBox objects.

Note Although they share similar appearances and behaviors, command bar controls and ActiveX controls aren't the same. You cannot add ActiveX controls to command bars, and you cannot add command bar controls to documents or forms.

The built-in command bar controls in container applications are also represented in Visual Basic by **CommandBarButton**, **CommandBarPopup**, and **CommandBarComboBox** objects, even though their appearances and behaviors may be more complex than the controls you can add yourself. Although you can modify the location and appearance of built-in controls, you cannot modify their built-in behavior. You can, however, assign a custom macro to any built-in control to completely replace its built-in behavior.

Using command bars

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofhowUsingCommandBarsC"}
"Specifics":"ofhowUsingCommandBarsS"}

{ewc HLP95EN.DLL, DYNALINK,

In general, to create or modify toolbars, menu bars, and shortcut menus that you want to deliver with your Visual Basic application, you should use the customization features of the container application. Changes made to toolbars, menu bars, and shortcut menus using the features of the container application are known as "design-time" changes. For information about using the container application to make design-time changes, see the the online Help for that application.

You can add and modify toolbars, menu bars, and shortcut menus (and their component parts) by using the **CommandBars** portion of the Microsoft Office object model in Visual Basic code. You can write code that runs once to create toolbars and menu bars; in effect, the code simulates making design-time changes. In some container applications, however, you may be required to use a combination of this kind of Visual Basic code and the customization interface to design your Visual Basic application. The following are some common areas where you must use a combination of code and the container application's interface:

- If your container application doesn't provide an interface for adding or modifying edit boxes, drop-down list boxes, or combo boxes on toolbars, you must use Visual Basic code to add and design one of these controls.
- If your container application provides an interface for creating toolbars but doesn't provide one for creating a new menu bar, you'll need to create a menu bar by using Visual Basic. After you've created the menu bar in Visual Basic, you can design menus on that menu bar by using the container application's interface.
- If your container application doesn't provide a way to display custom shortcut menus while the customization interface is displayed, you must use Visual Basic code to modify those shortcut menus.

You can also write code that exists in your Visual Basic application to make changes to toolbars and menu bars while your application is running (for example, you can write code to disable a command on a menu bar under certain conditions, or to add buttons to a toolbar in response to a user's actions). Changes brought about by your code while your Visual Basic application is running are known as "run-time" changes.

The following topics provide information about how you can make design-time and run-time changes to toolbars and menu bars from your Visual Basic code.

[Adding and modifying toolbars](#)

[Adding and managing menu bars and menu items](#)

[Adding and displaying shortcut menus](#)

Adding and modifying toolbars

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofhowToolbarsC"}
"Specifics":"ofhowToolbarsS"}

{ewc HLP95EN.DLL, DYNALINK,

All host applications have an extensive interface for adding and designing custom toolbars (adding built-in buttons, adding macros as buttons, even adding pop-up controls to toolbars). The design-time changes you'll usually make from Visual Basic code are ones that add or modify combo box controls. Otherwise, working with toolbars in code is almost completely limited to making run-time changes (changing the button state, changing the button appearance, changing the button action, and so on).

Making run-time modifications to toolbars

There are several modifications you can make to a toolbar at run time. One of these modifications is to change the state of a command bar button on the toolbar. Each button control has two active states: pushed (**True**) and not pushed (**False**). To change the state of a button control, use the appropriate constant for the **State** property, as explained in the table later in this topic.

Another modification you can make at run time is to change the appearance or action of a button. To change the appearance of a button but not its action, use the **CopyFace** and **PasteFace** properties. These properties are useful if you want to copy the face of a particular button onto the Clipboard or import it into another application to change some of its features. Use the **PasteFace** property to transfer the button image from the Clipboard onto a specific button.

To change a button's action to a function you've developed, assign the custom procedure name to the button's **OnAction** property.

The following table lists the most common properties and methods for changing the state, appearance, or action of a button.

<u>Property or method</u>	<u>Description</u>
<u>CopyFace</u> , <u>PasteFace</u>	Copies or pastes the image on the face of a button. Use the CopyFace method to copy the face of the specified button to the Clipboard. Use the PasteFace method to paste the contents of the Clipboard onto the face of the specified button. The PasteFace method will fail if the Clipboard is empty. If the image on the Clipboard is too large for the button face, the image won't be scaled down. Generally, it's more convenient to copy and paste a button face at design time, but you can also make changes to a button face at run time. You can also use the <u>FaceId</u> property to assign a different built-in button face to a button.
<u>Id</u>	Specifies the value that represents the button's built-in functionality. For example, a button that copies highlighted text to the Clipboard has an Id value of 19.
<u>State</u>	Specifies the appearance, or state, of the button. Can be one of the following constants: msoButtonDown , msoButtonMixed , or msoButtonUp .
<u>Style</u>	Specifies whether the button face displays its icon or its caption. Can be one of the following constants: msoButtonAutomatic , msoButtonIcon , msoButtonCaption , or msoButtonIconandCaption .

<u>OnAction</u>	Specifies the procedure to be run when the user clicks a button, displays a menu, or changes the contents of a <u>combo box controls</u> .
<u>Visible</u>	Specifies whether the control is to be displayed or hidden from the user.
<u>Enabled</u>	Enables or disables a command bar; the name of a disabled command bar won't appear in the list of available command bars.

The following example assumes that the first two controls on the `CustomButtons` command bar are buttons. The `HideThem` procedure hides the first button and assigns a procedure to the **OnAction** property of the second button. When the **OnAction** procedure is run, the first control will be made visible and the second control will be hidden. If the second button is pressed while the procedure is running, the procedure will be halted. Note that you must declare both `startBtn` and `stopBtn` as global variables.

```
Sub HideThem()
Set v = CommandBars("CustomButtons")
Set startBtn = v.Controls(1)
With startBtn
    .Visible = False
    .Caption = "Stop Processing"
End With
Set stopBtn = v.Controls(2)
stopBtn.OnAction = "onActionButtons"
End Sub
```

```
Sub onActionButtons()
stopBtn.Visible = False
With startBtn
    .Visible = True
    .Style = msoButtonCaption
End With
Do While startBtn.State <> True
'Continue processing sub
Loop
End Sub
```

Adding and modifying combo box controls

Edit boxes, drop-down list boxes, and combo boxes are powerful new controls you can add to toolbars in your Visual Basic application. However, most container applications require that you use Visual Basic code to design these controls. To design a combo box control, you use the properties and methods described in the following table.

<u>Property or method</u>	<u>Description</u>
<u>Add</u>	Adds a combo box control to a command bar by specifying one of the following MsoControlType constants for the Type argument: msoControlEdit , msoControlDropdown , or msoControlComboBox .
<u>AddItem</u>	Adds an item to the drop-down list portion of a drop-down list box or combo box. You can specify the index number of the new item in the existing list, but if this number is larger than the number of items in the list, AddItem fails.
<u>Caption</u>	Specifies the label for the combo box control. This is

the label that's displayed next to the control if you set the **Style** property to **msoComboLabel**.

Style

Specifies whether the caption for the specified control will be displayed next to the control. Can be either of the following constants: **msoComboLabel** (the label is displayed) or **msoComboNormal** (the label isn't displayed).

OnAction

Specifies the procedure to be run when the user changes the contents of the combo box control.

The following example adds a combo box with the label "Quarter" to a custom toolbar and assigns the macro named "ScrollToQuarter" to the control.

```
Set newCombo = CommandBars("Custom1").Controls _  
    .Add(Type:=msoControlComboBox)  
With newCombo  
    .AddItem "Q1"  
    .AddItem "Q2"  
    .AddItem "Q3"  
    .AddItem "Q4"  
    .Style = msoComboNormal  
    .OnAction = "ScrollToQuarter"  
End With
```

While your application is running, the procedure assigned to the **OnAction** property of the combo box control is called each time the user changes the control. In the procedure, you can use the **ActionControl** property of the **CommandBars** object to find out which control was changed and to return the changed value. The **ListIndex** property will return the item typed or selected in the combo box.

Adding and managing menu bars and menu items

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofhowMenuBarsC"}
"Specifics":"ofhowMenuBarsS"}

{ewc HLP95EN.DLL, DYNALINK,

Some container applications don't provide an way for you to create new menu bars, so you'll need to create menu bars by using Visual Basic. After you've created a menu bar in Visual Basic, you can customize it either by using the container application's interface or by continuing to use Visual Basic.

Adding menu bars at run time

When you add a menu bar to an application at run time, you use the **Add** method for the **CommandBars** collection and specify **True** for the **MenuBar** argument. The following example adds a menu bar that cannot be moved. The example also docks this menu bar along the right side of the application window. The new menu bar becomes active whenever the user presses the ALT key.

```
Set menubar = CommandBars.Add _  
    (Name:="mBar", Position:=msoBarRight, MenuBar:=True)  
With menubar  
    .Protection = msoBarNoMove  
    .Visible = True  
End With
```

Making run-time modifications to menu bars

You can make modifications to the both the menu bar and the controls on that menu bar at run time. The changes you make to the menu bar may affect its appearance or its position; changes you make to the controls depend on the control type. The properties and methods listed in the following table are the most common ones used to modify menu bars at run time.

<u>Property or method</u>	<u>Description</u>
<u>Add</u>	Adds a menu bar by using the Add method of the CommandBars collection and specifying True for the MenuBar argument.
<u>Enabled</u>	If this property is set to True , the user can make the specified menu bar visible, using Visual Basic code. If this property is set to False , the user cannot make the menu bar visible, but it will appear in the list of available command bars.
<u>Protection</u>	Make it possible for you to protect the menu bar from specific user actions. Can be one of or a sum of the following constants: msoBarNoChangeDock , msoBarNoChangeVisible , msoBarNoCustomize , msoBarNoCustomize , msoBarNoHorizontalDock , msoBarNoMove , msoBarNoProtection , msoBarNoResize , and msoBarNoVerticalDock .
<u>Position</u>	Specifies the position of the new menu bar, relative to the application window. Can be one of the following constants: msoBarLeft , msoBarTop , msoBarRight , msoBarBottom , msoBarFloating , msoBarPopup (used to create shortcut menus), or msoBarMenuBar (specifies a menu bar for the Macintosh).
<u>Visible</u>	Specifies whether the control will be displayed or hidden from the user. If the control is hidden from the user, the menu bar name will still appear in the

list of available command bars.

The following example hides the active menu bar and replaces it with a temporary menu bar that's docked along the right side of the application window and is protected from the user.

```
Set oldMbar = CommandBars.ActiveMenuBar
Set newMbar = CommandBars.Add _
(Name:="newMenubar", Position:=msoBarRight, _
MenuBar:=True, temporary:=True)
With newMbar
    .Visible = True
    .Protection = msoBarNoMove
End With
```

Merging menu bars at run time

If you have custom menu bars in an application that's intended to be an add-in, you may want to specify how the controls will be represented in the container application. You can use the **OLEMenuGroup** property of the **CommandBarPopup** control to specify how the menu bar merging will occur.

If either the container application or the server doesn't implement command bars, "non-Office" merging will occur: the menu bar will be merged, along with all the toolbars from the server, and none of the toolbars from the container application will be merged. You can also use the **OLEUsage** property to specify how menu bar merging will occur.

If both the container application and the server implement command bars, "Office-in-Office" merging will occur: the command bar controls are embedded in the Office application, control by control. Both of these properties (**OLEMenuGroup** and **OLEUsage**) are relevant only for pop-up controls on menu bars, because menus are considered on the basis of their menu group category.

Making run-time modifications to menu items

The range of modifications you can make to a menu item depends on the control type. Generally, buttons are enabled, or hidden. Edit boxes, drop-down list boxes, and combo boxes are more versatile in that you can add or delete items from the list, and you can determine the action performed by looking at the value selected. You can change the action of any control to either a built-in or custom function.

The following table lists the most common properties and methods for changing the state, action, or contents of a control.

Property or method	Purpose
<u>Add</u>	Adds a menu item to a command bar. Can be one of the following MsoControlType constants for the Type argument for a built-in control: msoControlButton , msoControlEdit , msoControlDropdown , or msoControlComboBox .
<u>AddItem</u>	Adds an item to the drop-down list portion of a drop-down list box or combo box. You can specify the index number of the new item in the existing list, but if this number is larger than the number of items in the list, AddItem fails.
<u>Style</u>	Specifies whether the button face displays its icon or its caption. Can be one of the following constants: msoButtonAutomatic , msoButtonIcon , msoButtonCaption , or msoButtonIconandCaption .

OnAction

Specifies the procedure to be run whenever the user changes the value of the specified control.

Visible

Specifies whether the control will be displayed or hidden from the user.

This following example adds a temporary pop-up control named "Custom" at the end of the active menu bar, and then it adds a button control named "Import" to the Custom pop-up command bar.

```
Set myMenuBar = CommandBars.ActiveMenuBar
Set newMenu = myMenuBar.Controls.Add(Type:=msoControlPopup,
Temporary:=True)
newMenu.Caption = "Custom"
Set ctrl1 = newMenu.Controls
    .Add(Type:=msoControlButton, Id:=1)
ctrl1.Caption = "Import"
ctrl1.TooltipText = "Import"
ctrl1.Style = msoButtonCaption
```

Adding and displaying shortcut menus

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofhowShortcutMenusC"}
"Specifics":"ofhowShortcutMenusS"}

{ewc HLP95EN.DLL, DYNALINK,

A shortcut menu is a floating command bar that the user displays by right-clicking. It can contain the same control types as a command bar, and the controls behave in the same way as on a command bar. In most applications, however, you cannot create or modify shortcut menus from the application's interface. Therefore, you need to create and modify your shortcut menus at run time.

Adding shortcut menus at run time

The only difference between shortcut menus and other toolbars is that when you create the shortcut menu with the **Add** method, you must specify **msoBarPopup** as the **Position** argument. The following example creates a new shortcut menu, adds two controls (with captions) to it, and then uses the **ShowPopup** method to display the new menu.

```
Set x = CommandBars.Add("Custom", msoBarPopup)
Set y = x.Controls.Add
With y
    .FaceId = 26
    .Caption = "Analyze the data"
End With
Set z = x.Controls.Add
With z
    .FaceId = 17
    .Caption = "Graph the data"
End With
x.ShowPopup 200, 200
```

Displaying shortcut menus

Use the **ShowPopup** method to display shortcut menus, as demonstrated in the preceding example.

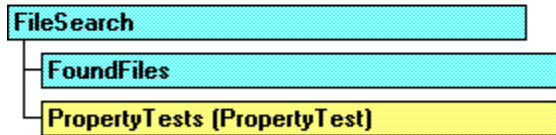
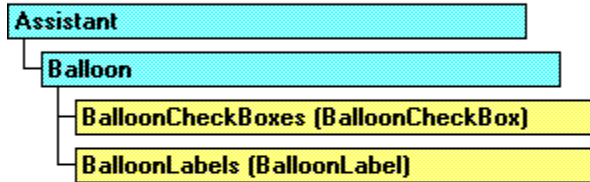
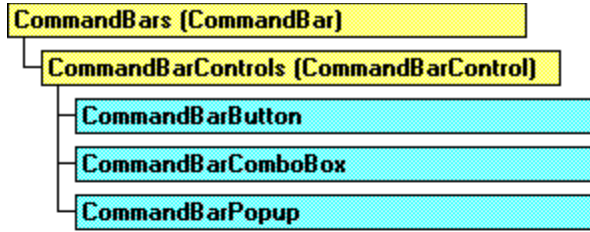
If the container application supports assigning event procedures to user actions, you can display a shortcut menu in response to a right-click event. However, not all applications support event procedures. Check the documentation for your container application to see whether the application supports event procedures.

Making run-time modifications to shortcut menus

Any changes you make to a shortcut menu must be made at run time, and the changes you make will generally be limited to changing the appearance or action of the controls on the menu. For more information about adding and managing menu items, see [Adding and managing menu bars and menu items](#).

Microsoft Office Objects

{ewc HLP95EN.DLL, DYNALINK, "See Also": "oftocObjectModelApplicationC"}



DocumentProperties (DocumentProperty)

Legend

 Object and collection  Object only

Returning an Object from a Collection

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofhowReturningAnObjectC "}

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 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.

Help Topic Not Available

The Help topic cannot be displayed because Visual Basic for Applications Help cannot be found or was not installed.

To install Visual Basic for Applications Help

1. Run Microsoft Office 97 Setup, and click **Add/Remove**.
1. Click **Microsoft Access**, **Microsoft Excel**, **Microsoft PowerPoint**, or **Microsoft Word**, and then click **Change Option**.
1. Click **Help**, and then click **Change Option**.
1. Make sure that the **Help for Visual Basic** or **Language Reference** (Microsoft Access) check box is selected.
1. Continue with Setup.

OLE Programmatic Identifiers (ActiveX Controls)

{ewc HLP95EN.DLL, DYNALINK, "See Also":"ofmscProgrammaticIdentifiersC;vafctCreateObject;vafctGetObject;OLE Programmatic Identifiers"}

You use an OLE programmatic identifier (sometimes called a ProgID) to create an Automation object. Use one of the following OLE programmatic identifiers to create an ActiveX control.

Use this identifier	To create this object
Forms.CheckBox.1	<u>CheckBox</u>
Forms.ComboBox.1	<u>ComboBox</u>
Forms.CommandButton.1	<u>CommandButton</u>
Forms.Frame.1	<u>Frame</u>
Forms.Image.1	<u>Image</u>
Forms.Label.1	<u>Label</u>
Forms.ListBox.1	<u>ListBox</u>
Forms.MultiPage.1	<u>MultiPage</u>
Forms.OptionButton.1	<u>OptionButton</u>
Forms.ScrollBar.1	<u>ScrollBar</u>
Forms.SpinButton.1	<u>SpinButton</u>
Forms.TabStrip.1	<u>TabStrip</u>
Forms.TextBox.1	<u>TextBox</u>
Forms.ToggleButton.1	<u>ToggleButton</u>

