# **Microsoft Office Object Model**

AnswerWizard <sup>L</sup>AnswerWizardFiles

Assistant L Balloon

BalloonCheckboxes

BalloonLabels

COMAddIns L COMAddIn

<u>CommandBarButton</u>
<u>CommandBar</u>
<u>CommandBar</u>
<u>CommandBarControls</u>
<u>CommandBarControl</u>

<u>CommandBarComboBox</u> <u>CommandBar</u> <u>CommandBarControls</u> <u>CommandBarControl</u>

<u>CommandBarPopup</u> <u>CommandBar</u> <u>CommandBarControls</u> <u>CommandBarControl</u>

**CommandBars** 

<u>CommandBar</u> <u>CommandBarControls</u> <u>CommandBarControl</u>

DocumentLibraryVersions

DocumentProperties

<u>FileDialog</u> <u>FileDialogFilters</u> <u>FileDialogFilter</u> <u>FileDialogSelectedItems</u>

**FileSearch** 

<u>FileTypes</u> <u>FoundFiles</u>

-<u>PropertyTests</u>

<sup>L</sup><u>PropertyTest</u>

<u>SearchFolders</u>

<sup>L</sup><u>ScopeFolder</u>

<sup>L</sup><u>ScopeFolders</u>

SearchScopes

<u>SearchScope</u>

<sup>L</sup><u>ScopeFolder</u>

└<u>ScopeFolders</u>

HTMLProject LHTMLProjectItems LHTMLProjectItem **LanguageSettings** 

**MsoEnvelope** 

<u>NewFile</u>

ODSOColumns

ODSOFilters

<u>OfficeDataSourceObject</u>

Permission

Scripts

SharedWorkspace
SharedWorkspaceFiles
SharedWorkspaceFile
SharedWorkspaceFolders
SharedWorkspaceFolder
SharedWorkspaceLinks
SharedWorkspaceLink
SharedWorkspaceMembers
SharedWorkspaceMembers
SharedWorkspaceTasks
SharedWorkspaceTask

SignatureSet

<u>SmartDocument</u>

<u>Sync</u>

WebPageFonts

# **New Objects**

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

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

Object	Description
<b>DocumentLibraryVersion</b>	A single saved backup copy or version of a shared document.
<b>DocumentLibraryVersions</b>	A collection of <b>DocumentLibraryVersion</b> objects.
<b>Permission</b>	Manages document permissions; also, a collection of <b>UserPermission</b> objects.
<b>SharedWorkspace</b>	Manages a document saved in a shared workspace.
<u>SharedWorkspaceFile</u>	A file saved in a shared workspace.
<u>SharedWorkspaceFiles</u>	A collection of <b>SharedWorkspaceFile</b> objects.
<u>SharedWorkspaceFolder</u>	A subfolder in a shared workspace.
<u>SharedWorkspaceFolders</u>	A collection of <b>SharedWorkspaceFolder</b> objects.
<u>SharedWorkspaceLink</u>	A link saved in a shared workspace.
<u>SharedWorkspaceLinks</u>	A collection of <b>SharedWorkspaceLink</b> objects.
<u>SharedWorkspaceMember</u>	A user of a shared workspace.
<b>SharedWorkspaceMembers</b>	A collection of <b>SharedWorkspaceMember</b> objects.
<u>SharedWorkspaceTask</u>	A task saved in a shared workspace.
<u>SharedWorkspaceTasks</u>	A collection of <b>SharedWorkspaceTask</b> objects.
<u>SmartDocument</u>	Manages an attached XML expansion pack which transforms a document into a smart document.
<u>Sync</u>	Manages the synchronization between the local copy and the server copy of a shared document.

		· · · ·	
	Ser	Perm	ission
$\sim$	UCII		

A set of permissions on a document for a single user.

# **New Properties (Alphabetical List)**

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

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

<b>New Property</b>	Object(s)
<u>AssignedTo</u>	SharedWorkspaceTask
<b>Comments</b>	DocumentLibraryVersion
<b>Connected</b>	SharedWorkspace
<u>CreatedBy</u>	SharedWorkspaceFile, SharedWorkspaceFolder, SharedWorkspaceLink, SharedWorkspaceTask
<u>CreatedDate</u>	SharedWorkspaceFile, SharedWorkspaceFolder, SharedWorkspaceLink, SharedWorkspaceTask
<b>DocumentAuthor</b>	Permission
<u>DomainName</u>	SharedWorkspaceMember
<u>DueDate</u>	SharedWorkspaceTask
<u>Email</u>	SharedWorkspaceMember
<b>EnableTrustedBrowser</b>	Permission
<u>ErrorType</u>	Sync
<b>ExpirationDate</b>	UserPermission
<u>FolderName</u>	SharedWorkspaceFolder
<u>Folders</u>	SharedWorkspace
<b>IsVersioningEnabled</b>	DocumentLibraryVersions
<b>ItemCountExceeded</b>	SharedWorkspaceFiles, SharedWorkspaceFolders, SharedWorkspaceLinks, SharedWorkspaceMembers, SharedWorkspaceTasks

<b>LastRefreshed</b>	SharedWorkspace
<b>LastSyncTime</b>	Sync
Links	SharedWorkspace
<u>Members</u>	SharedWorkspace
<b>Modified</b>	DocumentLibraryVersion
<u>ModifiedBy</u>	SharedWorkspaceFile, SharedWorkspaceFolder, SharedWorkspaceLink, SharedWorkspaceTask, DocumentLibraryVersion
<b>ModifiedDate</b>	SharedWorkspaceFile, SharedWorkspaceFolder, SharedWorkspaceLink, SharedWorkspaceTask
<u>Notes</u>	SharedWorkspaceLink
<b><u>Permission</u></b>	UserPermission
<b><u>PermissionFromPolicy</u></b>	Permission
<b>PolicyDescription</b>	Permission
<b>PolicyName</b>	Permission
<b><u><b>RequestPermissionURL</b></u></b>	Permission
<b>SolutionID</b>	SmartDocument
SolutionURL	SmartDocument
<b>SourceURL</b>	SharedWorkspace
<u>Status</u>	SharedWorkspaceTask, Sync
<b>StoreLicenses</b>	Permission
<u>Tasks</u>	SharedWorkspace
<u>UserId</u>	UserPermission
<u>WorkspaceLastChangedBy</u>	y Sync

# **New Properties (by Object)**

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

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

Object	<b>New Properties</b>
DocumentLibraryVersion	<u>Comments</u> , <u>Modified</u> , <u>ModifiedBy</u>
DocumentLibraryVersions	<u>IsVersioningEnabled</u>
Permission	DocumentAuthor, EnableTrustedBrowser, <u>PermissionFromPolicy</u> , <u>RequestPermissionURL</u> , <u>StoreLicenses</u> , <u>PolicyDescription</u> , <u>PolicyName</u>
SharedWorkspace	Connected
SharedWorkspaceFile	<u>CreatedBy</u> , <u>CreatedDate</u> , <u>ModifiedBy</u> , <u>ModifiedDate</u>
SharedWorkspaceFiles	<b>ItemCountExceeded</b>
SharedWorkspaceFolder	<u>CreatedBy</u> , <u>CreatedDate</u> , <u>FolderName</u> , <u>ModifiedBy</u> , <u>ModifiedDate</u>
SharedWorkspace	Folders
SharedWorkspaceFolders	<b>ItemCountExceeded</b>
SharedWorkspace	LastRefreshed
SharedWorkspaceLink	<b>Notes</b> , <b>CreatedBy</b> , <b>CreatedDate</b> , <b>ModifiedBy</b> , <b>ModifiedDate</b>
SharedWorkspace	Links
SharedWorkspaceLinks	<b>ItemCountExceeded</b>
SharedWorkspaceMember	DomainName , Email
SharedWorkspace	<u>Members</u>
SharedWorkspaceMembers	<b>ItemCountExceeded</b>
SharedWorkspace	SourceURL

SharedWorkspaceTask	AssignedTo , CreatedBy , CreatedDate , DueDate , ModifiedBy , ModifiedDate
SharedWorkspace	<u>Tasks</u>
SharedWorkspaceTasks	<b>ItemCountExceeded</b>
SharedWorkspaceTask	<u>Status</u>
SmartDocument	SolutionID , SolutionURL
Sync	<u>ErrorType</u> , <u>LastSyncTime</u> , <u>Status</u> , <u>WorkspaceLastChangedBy</u>
UserPermission	ExpirationDate , Permission , UserId

# **New Methods (Alphabetical List)**

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

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

New Method	Object
<b>ApplyPolicy</b>	Permission
<b><u>CreateNew</u></b>	SharedWorkspace
<b>Disconnect</b>	SharedWorkspace
<u>GetUpdate</u>	Sync
<u>OpenVersion</u>	Sync
<b><u>PickSolution</u></b>	SmartDocument
<b>PutUpdate</b>	Sync
<u>Refresh</u>	SharedWorkspace
<b><u>RefreshPane</u></b>	SmartDocument
<b><u>RemoveAll</u></b>	Permission
<b><u>RemoveDocument</u></b>	SharedWorkspace
<b><u>ResolveConflict</u></b>	Sync
<b><u>Restore</u></b>	DocumentLibraryVersion
<u>Unsuspend</u>	Sync

# New Methods (by Object)

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

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

<b>New Method</b>	Object
DocumentLibraryVersion <u>Restore</u>	
Permission	<u>ApplyPolicy</u> , <u>RemoveAll</u>
SharedWorkspace	<u>CreateNew</u> , <u>Disconnect</u> , <u>Refresh</u> , <u>RemoveDocument</u>
SmartDocument	PickSolution , RefreshPane
Sync	<u>GetUpdate</u> , <u>OpenVersion</u> , <u>PutUpdate</u> , <u>ResolveConflict</u> , <u>Unsuspend</u>

# Security Notes for Microsoft Office Solution Developers

About Setting Microsoft Office 2003 Security in a Testing Environment Microsoft Office 2003 Macro and Add-in Security Settings Matrix About Enabling "Trust all installed add-ins and templates" About Modifying the Microsoft Windows Registry About Making Microsoft Windows Application Programming Interface (API) Function Calls About Digital Code Signing About Secure Deployment of Managed COM Add-ins in Microsoft Office 2003 About Automating the Visual Basic Editor About Passwords About Microsoft Office Outlook 2003 Security Settings

# About Setting Microsoft Office 2003 Security in a Testing Environment

To install and run an unsigned COM add-in or Microsoft Visual Basic for Applications (VBA) macro, the security settings in the **Security** dialog box (**Tools** menu, **Macro** submenu, **Security** command) must be set to **Medium**, with the **Trust all installed add-ins and templates** check box cleared. It is strongly recommended that you do this only in a testing environment. After you have completed your testing, set the security level back to **Very High** or **High**.

**Caution** By setting the security level to **Medium**, with the **Trust all installed add-ins and templates** check box cleared, users will have the choice to either enable or disable unsigned COM add-ins and VBA macros when they are prompted. If your security level is set to **Very High** or **High**, with the **Trust all installed add-ins and templates** check box cleared, all unsigned COM add-ins and VBA macros will be disabled automatically. Therefore, it is strongly recommended that all users keep their security levels set to **Very High** or **High** with the **Trust all installed add-ins and templates** check box cleared.

## **Microsoft Office 2003 Macro and Add-in Security Settings Matrix**

The following table lists the available Microsoft Office 2003 security settings, along with their corresponding behaviors, in the **Security** dialog box (**Macros** submenu, **Tools** menu).

Security level	Digitally signed?	From trusted sources?	Office 2003 will
Very High	Yes	Yes	Load the add-in or macro silently
	Yes	No	Not load the add-in or macro
	No	N/A	Not load the add-in or macro
High	Yes	Yes	Load the add-in or macro silently
	Yes	No	Prompt to trust the source and enable the add-in or macro to run
	No	N/A	Not load the add-in or macro
Medium	Yes	Yes	Load the add-in or macro silently
	Yes	No	Prompt to trust the source and enable the add-in or macro to run
	No	N/A	Prompt to enable or disable the add-in or macro
Low	Yes or No	Yes or No	Load the add-in or macro silently

#### Office 2003 macro and add-in security setting options

**Note** The availability of, and options within, the **Security** dialog box vary depending on the specific Office application. Additionally, specific Office applications silently load signed add-ins and macros only from specific directories, along with registered COM add-ins and smart tags recognizers.

For more information on these settings and on the other issues discussed in this topic, search for "Office macro security" in the Microsoft Developer Network (MSDN) Library.

# About Enabling "Trust all installed add-ins and templates"

The **Trust all installed add-ins and templates** check box is commonly misunderstood. By default it is enabled, but Microsoft recommends that customers with high security requirements disable it, and this is a good "defense in depth" approach.

If you have no need to run unsigned personal macros or unsigned COM add-ins, you should disable this option. When Office application security level is set to **Very High** or **High**, with the **Trust all installed add-ins and templates** option disabled, all add-ins and templates that are not signed will automatically be disabled. If the add-ins and templates are signed using a certificate not listed in the **Trusted Sources** list, a user will be prompted to trust the source in order to allow the add-in or template to run.

Enabling the **Trust all installed add-ins and templates** check box will allow all COM add-ins that have been installed in the registry (which requires administrative privileges) or VBA macros that are stored in your personal or workgroup locations to run, regardless of whether they are signed or not. It should also be noted that end-users do not need administrative privileges to install VBA macros to certain template and startup folders. Examples of these locations in Office Word 2003 are:

- \Documents and Settings\<user name>\Application Data\Microsoft\Word\STARTUP
- \Documents and Settings\<user name>\Application Data\Microsoft\Templates

With the **Trust all installed add-ins and templates** option enabled, you can be attacked if you download, register, and run a malicious COM add-in or if you run a malicious template from someone else.

If you decide to run personal VBA macros or to run locally installed COM addins, and you don't want to purchase a digital certificate, but you want to disable the **Trust all installed add-ins and templates** option, here are the available alternatives:

- Sign your personal VBA macros or locally installed COM add-ins using a test certificate created using either selfcert.exe or Authenticode tools, depending on whether it's a macro or add-in. This way, you can keep your security settings set to Very High or High, with the Trust all installed add-ins and templates option disabled. For more information on how to generate a test certificate, refer to the About Digital Code Signing section below.
- Set your Office application security level to **Medium** with **Trust all installed add-ins and templates** disabled. With these security settings, you will be prompted to enable or disable a COM add-in or VBA macro when the application is launched. It's strongly recommended that you use these settings only in a testing environment.

## **About Modifying the Microsoft Windows Registry**

Modifying the Microsoft Windows registry in any manner, whether through the Registry Editor or programmatically, always carries some degree of risk. Incorrect modification can cause serious problems that may require you to reinstall your operating system. It is always a good practice to back up a computer's registry first before modifying it. If you are running Microsoft Windows NT, Microsoft Windows 2000, Microsoft Windows XP, or Microsoft Windows Server 2003, you should also update your Emergency Repair Disk (ERD).

For information about how to edit the registry, view the "Changing Keys and Values" Help topic in the Registry Editor (Regedit.exe) or the "Add and Delete Information in the Registry" and "Edit Registry Information" topics in the Registry Editor (Regedt32.exe).

## **About Making Microsoft Windows Application Programming Interface (API) Function Calls**

Before calling Microsoft Windows API functions, you should understand how arguments and data types are handled by the Windows API DLLs. Incorrectly calling Windows API functions may result in invalid page faults or other unexpected behaviors. For more information on calling Windows API functions, see the topic "The Windows API and Other Dynamic-Link Libraries" in the Microsoft Office 2003 Developer Online Documentation or the Microsoft Developer Network (MSDN) Library.

## **About Digital Code Signing**

Many security-conscious users and administrators set their Microsoft Office 2003 security levels to **Very High** or **High** with the **Trust all installed add-ins and templates** check box cleared (located in the **Security** dialog box, **Macro** submenu, **Tools** menu), which is highly recommended. With these settings, a signed and trusted COM add-in or VBA macro will be loaded, and a non-signed COM add-in or VBA macro will be disabled automatically. The only time a user will be prompted to either enable or disable a COM add-in or VBA macro is when a COM add-in or VBA macro is signed but the software publisher is not included in the **Trusted Sources** list.

Microsoft Authenticode technology allows software publishers to digitally sign executable (EXE) files, ActiveX control (OCX) files, cabinet (CAB) files, and dynamic-link library (DLL) files. For a step-by-step guide on how to digitally sign a COM add-in using Microsoft Authenticode technology, search for "digital code signing" in the MSDN Library.

## **About Secure Deployment of Managed COM Add-ins** in Microsoft Office 2003

To comply with Office 2003 security, managed COM add-ins (COM add-ins targeting the common language runtime) must be digitally signed, and users' security settings should be set to their highest levels. Additionally, you will need to incorporate into your managed COM add-in project a small unmanaged proxy called a *shim* in order to avoid unexpected security warnings. For details, search for "deployment managed add-ins" in the MSDN Library.

### **About Automating the Visual Basic Editor**

In Office 2003, when calling the features of the **Microsoft Visual Basic for Applications Extensibility** object model, you may receive an error message that programmatic access to the Visual Basic project is not trusted. To prevent this message from appearing, point to **Macro** on the **Tools** menu, and then click **Security**. On the **Trusted Sources** tab, check the **Trust access to Visual Basic Project** box. By checking this box, macros in any documents that you open can access the core Microsoft Visual Basic objects, methods, and properties, which represents a possible security hazard. The default behavior in Office 2003 is to not allow macros to programmatically access the Visual Basic object model. The recommended behavior is to check the **Trust access to Visual Basic Project** box only for the duration of a macro that accesses the Visual Basic object model. The **Trust access to Visual Basic Project** box should be unchecked after the macro has finished running.

## **About Passwords**

Avoid using hard-coded passwords in your applications. If a password is required in a procedure, request the password from the user, store it in a variable, and then use the variable in your code.

Always use strong passwords. Strong passwords should contain:

- Both lowercase and uppercase characters.
- Numbers.
- Symbols (such as #, \$, %, and ^).
- At least eight characters.

Strong passwords should not contain patterns, themes, or words found in a dictionary.

Examples of strong passwords include:

- \$tR0n9p@\$s
- G80dn[s\$M4!

**Note** You should change your password frequently; for example, every one to three months.

# **About Microsoft Office Outlook 2003 Security Settings**

### **COM Add-ins Using Default Security**

In Microsoft Office Outlook 2003, all COM add-ins that run on a computer which is not configured to obtain security settings from a Microsoft Exchange Server are considered trusted by default. This implies that the add-ins that run on clients that are not Exchange clients and the add-ins that use default security in Exchange environments are trusted automatically. As in Outlook 2002, Office Outlook 2003 trusts only the main **Application** object that is passed to the **OnConnection** event of the add-in.

#### **COM Add-ins Using Security Settings from an Exchange Server**

There has been no change in the way Office Outlook 2003 trusts COM add-ins in an Exchange environment when the security settings are obtained from the Exchange server. An add-in will be considered trusted only if it's registered in the Security Settings folder. Again, as in Outlook 2002, Office Outlook 2003 trusts only the main **Application** object that is passed to the **OnConnection** event of the add-in.

#### Improvements to Outlook Object Model Guard and the Impact

Office Outlook 2003 inherits the Outlook 2002 object model guard behavior and, in addition, blocks code that attempts to access the **Body** and **HTMLBody** properties of various Outlook items. This allows users to verify that the program or add-in accessing the **Body** and **HTMLBody** properties of these items is trustworthy, before they allow access to the contents of the items. Although this change forces the display of security warnings in existing COM add-ins that access the **Body** or **HTMLBody** properties of items, this will help prevent malicious code from running unknown to the user.

You can avoid the display of security warnings by deriving all objects, properties, and methods from the **Application** object passed to the **OnConnection** procedure of the add-in. Office Outlook 2003 trusts only the

**Application** object passed to the **OnConnection** procedure of the add-in. If you create a new **Application** object -- for example, by using the **CreateObject** method -- that object and any of its subordinate objects, properties, and methods will not be trusted and the blocked properties and methods will raise security warnings.

### **New Object Model Blocks in Office Outlook 2003**

The following are the additional properties that have been blocked in Office Outlook 2003:

- The **IMAddress** property of a **ContactItem** object.
- The **HTMLBody** property of a **MailItem** object.
- The Body property of the following objects: ContactItem, MailItem, PostItem, AppointmentItem, TaskItem, TaskRequestItem, TaskRequestAcceptItem, TaskRequestDeclineItem, TaskRequestUpdateItem, DistListItem, JournalItem, MeetingItem, ReportItem, RemoteItem, NoteItem, or DocumentItem.

Also, if you use a third-party add-ins, custom solutions, or other programs that integrate with Office Outlook 2003, you may receive one or more of the following warnings:

- "A program is trying to automatically send e-mail on your behalf. Do you want to allow this? If you unexpectedly receive this message, it may be caused by a virus, and you should choose No."
- "A program is trying to access e-mail addresses you have stored in Outlook. Do you want to allow this? If you unexpectedly receive this message, it may be caused by a virus, and you should choose No. "

These warning messages are commonly associated with software that is designed to synchronize Outlook data with handheld computers, but may occur with any type of add-in or custom solution.

# **AnswerWizardFiles Collection Object**

AnswerWizard <sup>L</sup>AnswerWizardFiles

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

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

### Using the AnswerWizardFiles Collection

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

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

```
MsgBox customAnswerWizardFiles.Count
```

Use the **Add** method to make additional files available to the current Answer Wizard. The following example adds the file Custom\_1.aw to the list of Answer Wizard files in the active Office application.

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

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

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

MsgBox customAnswerWizard.Files.Item(1)

# **BalloonCheckBoxes Collection Object**

**BalloonCheckboxes** 

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

## Using the BalloonCheckBoxes Collection

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

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

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

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

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

## Remarks

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

# **BalloonLabels Collection Object**

BalloonLabels Label

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

## **Using the BalloonLabels Collection**

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

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

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

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

## Remarks

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

# **COMAddIns Collection Object**

<u>COMAddIns</u> <u>COMAddIn</u>

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

## **Using the COMAddIns Collection**

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

MsgBox Application.COMAddIns.Count

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

Application.COMAddIns.Update

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

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



## **CommandBarControls Collection Object**

Multiple objects <sup>L</sup><u>CommandBarControls</u> <sup>L</sup>Multiple objects

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

#### Using the CommandBarControls Collection

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

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

Use the **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**, <u>CommandBarButton</u>, <u>CommandBarComboBox</u>, or <u>CommandBarPopup</u> 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
```



# **CommandBars Collection Object**

<u>CommandBars</u> <sup>L</sup>Multiple objects

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

#### **Using the CommandBars Collection**

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

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

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

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

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

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

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

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

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

### **DocumentLibraryVersions Collection**

DocumentLibraryVersions <sup>L</sup>DocumentLibraryVersion

The **DocumentLibraryVersions** property of the Microsoft Office Word 2003 **Document**, the Microsoft Office Excel 2003 **Workbook**, and the Microsoft Office PowerPoint 2003 **Presentation** objects returns a **DocumentLibraryVersions** object. The **DocumentLibraryVersions** object represents a collection of **DocumentLibraryVersion** objects.

### Using the DocumentLibraryVersions Collection

Use the **DocumentLibraryVersions** object with documents stored in a Windows SharePoint Services document library on the server to determine whether versioning is enabled for the active document and, if versioning is enabled, to manage the document's collection of **DocumentLibraryVersion** objects.

Each **DocumentLibraryVersion** object represents one saved version of the active document. When versioning is enabled, a new version is created on the server when the actions listed below occur; additional versions are not created each time the user saves changes to the open document.

- Check In
- Save A new version is created on the server when the user first saves the document after opening it. Additional changes saved while the document is open apply to the same version.
- Restore
- Upload

The **DocumentLibraryVersions** object model is available whether versioning is enabled or disabled on the active document. The **DocumentLibraryVersions** property of the **Document**, **Workbook** and **Presentation** objects does not return Nothing when the active document is not stored in a document library or versioning is not enabled. Use the **IsVersioningEnabled** property to determine whether the document library is configured to save a backup copy, or version, each time the document is edited on the web site.

#### Example

The following example checks to see whether versioning is enabled for the active document and, if so, displays information about each saved version.

```
Dim dlvVersions As Office.DocumentLibraryVersions
Dim dlvVersion As Office.DocumentLibraryVersion
Dim strVersionInfo As String
Set dlvVersions = ActiveDocument.DocumentLibraryVersions
If dlvVersions.IsVersioningEnabled Then
    strVersionInfo = "This document has " & _
        dlvVersions.Count & " versions: " & vbCrLf
    For Each dlvVersion In dlvVersions
        strVersionInfo = strVersionInfo &
             " - Version #: " & dlvVersion.Index & vbCrLf & _

    Modified by: " & dlvVersion.ModifiedBy & vbCrLf
    Modified on: " & dlvVersion.Modified & vbCrLf &

             н
             п
               - Comments: " & dlvVersion.Comments & vbCrLf
             п
    Next
Else
    strVersionInfo = "Versioning not enabled for this document."
End If
MsgBox strVersionInfo, vbInformation + vbOKOnly, "Version Inform
Set dlvVersion = Nothing
Set dlvVersions = Nothing
```

# **DocumentProperties Collection Object**

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

### **Using the DocumentProperties Collection**

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

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

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

# **FileDialogFilters Collection**

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

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

#### **Using the FileDialogFilters collection**

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

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

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

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

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

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

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

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

```
'Microsoft Excel files
If InStr(1, fdf.Extensions, "xls", vbTextCompare) > 0 Then
MsgBox "Description of filter: " & fdf.Description
End If
Next fdf
```

End Sub

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

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

# FileDialogSelectedItems Collection

FileDialog <sup>L</sup>FileDialogSelectedItems

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

#### Using the FileDialogSelectedItems collection

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

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

End Sub

## **FileTypes Collection**

FileSearch LFileTypes

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

#### **Using the FileTypes collection**

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

Set ft = Application.FileSearch.FileTypes

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

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

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

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

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

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

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

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

# **FoundFiles Object**

FileSearch LFoundFiles

Represents the list of files returned from a file search.

#### **Using the FoundFiles Object**

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

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

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

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

# **HTMLProjectItems Collection**

<u>HTMLProject</u> L<u>HTMLProjectItems</u>

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

### Using the HTMLProjectItems Collection

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

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

# **ODSOColumns Object**

ODSOColumns CODSOColumn

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

#### Using the ODSOColumns object

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

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

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

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

## **ODSOFilters Object**

ODSOFilters ODSOFilter

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

#### Using the ODSOFilters object

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

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

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

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

End Sub

### **PropertyTests Collection Object**

<u>FileSearch</u> PropertyTests

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

#### **Using the PropertyTests Collection**

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

Application.FileSearch.PropertyTests.Count

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

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

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

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

# **ScopeFolders Collection**

ScopeFolder <sup>L</sup>ScopeFolders <sup>L</sup>ScopeFolder <sup>L</sup>ScopeFolders

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

#### **Using the ScopeFolders collection**

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

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

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

# **Scripts Collection Object**

Scripts Script

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

### **Using the Scripts Collection**

The **Scripts** collection contains all of the **Script** objects in a given document, in source order (the order in which **Script** objects were added to the source file). Source order isn't affected by the location (header or body text) of the script in the document. The **Scripts** collection can be accessed by using the **Scripts** property of the appropriate object (for example, the **Document** object in Microsoft Word).

You can use **Script** objects to access a script or to add a script to a Microsoft Word document, a Microsoft Excel worksheet, or a Microsoft PowerPoint slide. You can also use the **Scripts** collection to access any HTML page or script that's opened in a Microsoft Office application.

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

### **Adding a Script**

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

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

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

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

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

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

" scripts in this document. ") End If

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

ActiveDocument.Scripts("ScriptOne").Delete

# **SearchFolders Collection**

<u>FileSearch</u> <u>SearchFolders</u> <u>ScopeFolder</u> <u>ScopeFolders</u>

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

#### **Using the SearchFolders collection**

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

Set sfs = Application.FileSearch.SearchFolders

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

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

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

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

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

```
Sub SearchEveryFolder()
```

```
'Declare variables that reference a
```

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

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

#### For Each sf In sfs

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

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

#### End If

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

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

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

End If Next sf End Sub

### **SearchScopes Collection**

<u>FileSearch</u> <u>SearchScopes</u> <u>SearchScope</u> <u>ScopeFolder</u>

A collection of **SearchScope** objects.

#### **Using the SearchScopes collection**

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

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

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

# **SharedWorkspaceFiles** Collection

SharedWorkspace <sup>L</sup>SharedWorkspaceFiles <sup>L</sup>SharedWorkspaceFile

A collection of the **SharedWorkspaceFile** objects in the current shared workspace.

#### Using the SharedWorkspaceFiles Collection

Use the **Files** property of the **SharedWorkspace** object to return a **SharedWorkspaceFiles** collection.

```
Dim swsFiles As Office.SharedWorkspaceFiles
Set swsFiles = ActiveWorkbook.SharedWorkspace.Files
MsgBox "There are " & swsFiles.Count & _
    "file(s)
    vbInformation + vbOKOnly, _
    "Collection Information"
Set swsFiles = Nothing
```

# **SharedWorkspaceFolders** Collection

SharedWorkspace <sup>L</sup>SharedWorkspaceFolders <sup>L</sup>SharedWorkspaceFolder

A collection of the **SharedWorkspaceFolder** objects in the current shared workspace.

#### Using the SharedWorkspaceFolders Collection

Use the **Folders** property of the **SharedWorkspace** object to return a **SharedWorkspaceFolders** collection.

```
Dim swsFolders As Office.SharedWorkspaceFolders
Set swsFolders = ActiveWorkbook.SharedWorkspace.Folders
MsgBox "There are " & swsFolders.Count & _
        "folder(s) in the current shared workspace.", _
        vbInformation + vbOKOnly, _
        "Collection Information"
Set swsFolders = Nothing
```

# SharedWorkspaceLinks Collection

SharedWorkspace Links

A collection of the **SharedWorkspaceLink** objects in the current shared workspace.

#### Using the SharedWorkspaceLinks Collection

Use the **Links** property of the **SharedWorkspace** object to return a **SharedWorkspaceLinks** collection.

```
Dim swsLinks As Office.SharedWorkspaceLinks
Set swsLinks = ActiveWorkbook.SharedWorkspace.Links
MsgBox "There are " & swsLinks.Count & _
    " link(s) in the current shared workspace.", _
    vbInformation + vbOKOnly, _
    "Collection Information"
Set swsLinks = Nothing
```

# SharedWorkspaceMembers Collection

SharedWorkspace LSharedWorkspaceMembers

A collection of the **SharedWorkspaceMember** objects in the current shared workspace.

#### Using the SharedWorkspaceMembers Collection

Use the **Members** property of the **SharedWorkspace** object to return a **SharedWorkspaceMembers** collection.

```
Dim swsMembers As Office.SharedWorkspaceMembers
Set swsMembers = ActiveWorkbook.SharedWorkspace.Members
MsgBox "There are " & swsMembers.Count & ______"
member(s) in the current shared workspace.", ______
vbInformation + vbOKOnly, ______"
Collection Information"
Set swsMembers = Nothing
```

# **SharedWorkspaceTasks Collection**

SharedWorkspace LSharedWorkspaceTasks

A collection of the **SharedWorkspaceTask** objects in the current shared workspace.

#### Using the SharedWorkspaceTasks Collection

Use the **Tasks** property of the **SharedWorkspace** object to return a **SharedWorkspaceTasks** collection.

```
Dim swsTasks As Office.SharedWorkspaceTasks
Set swsTasks = ActiveWorkbook.SharedWorkspace.Tasks
MsgBox "There are " & swsTasks.Count & _
    " task(s) in the current shared workspace.", _
    vbInformation + vbOKOnly, _
    "Collection Information"
Set swsTasks = Nothing
```

# **SignatureSet Collection**

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

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

#### **Using the SignatureSet collection**

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

Set sigs = ActiveDocument.Signatures

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

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

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

```
Function AddSignature(ByVal strIssuer As String, _
    strSigner As String) As Boolean
On Error GoTo Error_Handler
Dim sig As Signature
'Display the dialog box that lets the
'user select a digital signature.
'If the user selects a signature, then
'it is added to the Signatures
'collection. If the user doesn't, then
'an error is returned.
Set sig = ActiveDocument.Signatures.Add
'Test several properties before committing the Signature object
If sig.Issuer = strIssuer And _
    sig.Signer = strSigner And _
```

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



# WebPageFonts Collection Object

WebPageFonts LWebPageFont

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

### **Using the WebPageFonts Collection**

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

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

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

Dim myFont As WebPageFont
Set myFont = \_
Application.DefaultWebOptions.Fonts.Item\_
(msoCharacterSetEnglishWesternEuropeanOtherLatinScript)

# **AnswerWizard Object**

AnswerWizard <sup>L</sup>AnswerWizardFiles

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

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

#### Using the AnswerWizard Object

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

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

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

```
customAnswerWizard.ResetFileList
```

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

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

MsgBox customAnswerWizardFiles.Count

# **Assistant Object**

Assistant Balloon

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

Represents the Microsoft Office Assistant.

### **Using the Assistant Object**

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

### Remarks

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

The following example displays and animates the Assistant.

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

# **Balloon Object**

Assistant Balloon

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

#### **Using the Balloon Object**

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

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

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

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

### **Defining and Reusing Balloons**

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

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

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

# BalloonCheckBox Object

#### BalloonCheckbox

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

### Using the BalloonCheckBox Object

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

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

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

### Remarks

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

# **BalloonLabel Object**

BalloonLabels BalloonLabel

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

#### Using the BalloonLabel Object

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

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

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

### Remarks

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

## **COMAddIn Object**

#### COMAddIns COMAddIn

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

#### Using the COMAddIn Object

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

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

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

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

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

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



## **CommandBar Object**

Multiple objects CommandBar

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

#### Using the CommandBar Object

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

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

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

```
Set newItem = CommandBars("Tools").Controls.Add(Type:=msoControlButt
With newItem
    .BeginGroup = True
```

```
.Caption = "Make Report"
.FaceID = 0
.OnAction = "qtrReport"
End With
```

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

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

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

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



### **CommandBarButton Object**

CommandBarButton CommandBar

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

#### Using the CommandBarButton Object

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

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

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

You can also use the **<u>FindControl</u>** method to return a **CommandBarButton** object.



### CommandBarComboBox Object

<u>CommandBarComboBox</u> <u>CommandBar</u>

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

#### Using the CommandBarComboBox Object

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

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

```
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)
```



## **CommandBarControl Object**

Multiple objects <sup>L</sup><u>CommandBarControl</u> <sup>L</sup>CommandBar

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

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

#### Using the CommandBarControl Object

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

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

You can also use the **FindControl** method to return a **CommandBarControl** object. The following example searches for a control of type **msoControlGauge**; if it finds one, it displays the index number of the control and the name of the command bar that contains it. In this example, the variable 1b1 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
```



### **CommandBarPopup Object**

<u>CommandBarPopup</u> <sup>L</sup>Multiple objects

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

### Using the CommandBarPopup Object

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

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

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

### Remarks

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

## **DocumentLibraryVersion Object**

<u>DocumentLibraryVersions</u> <u>DocumentLibraryVersion</u>

The **DocumentLibraryVersion** object represents a single saved version of a shared document which has versioning enabled and which is stored in a document library on the server. Each **DocumentLibraryVersion** object is a member of the active document's **DocumentLibraryVersions** collection.

### Using the DocumentLibraryVersion Object

Each **DocumentLibraryVersion** object represents one saved version of the active document. When versioning is enabled, a new version is created on the server when the actions listed below occur; additional versions are not created each time the user saves changes to the open document.

- Check In
- Save A new version is created on the server when the user first saves the document after opening it. Additional changes saved while the document is open apply to the same version.
- Restore
- Upload

Use the **Modified**, **ModifiedBy**, and **Comments** properties to return information about a saved version of a shared document.

Use the **Open** method to open a previous version, or the **Restore** method to restore a previous version in place of the current version. Use the **Delete** method to delete a version.

### Example

The following example displays the properties of each saved version of the active document.

```
Dim dlvVersions As Office.DocumentLibraryVersions
Dim dlvVersion As Office.DocumentLibraryVersion
Dim strVersionInfo As String
Set dlvVersions = ActiveDocument.DocumentLibraryVersions
If dlvVersions.IsVersioningEnabled Then
    strVersionInfo = "This document has " & _
        dlvVersions.Count & " versions: " & vbCrLf
    For Each dlvVersion In dlvVersions
        strVersionInfo = strVersionInfo &
             " - Version #: " & dlvVersion.Index & vbCrLf & _

    Modified by: " & dlvVersion.ModifiedBy & vbCrLf
    Modified on: " & dlvVersion.Modified & vbCrLf &

             н
             п
               - Comments: " & dlvVersion.Comments & vbCrLf
             н
    Next
Else
    strVersionInfo = "Versioning not enabled for this document."
End If
MsgBox strVersionInfo, vbInformation + vbOKOnly, "Version Inform
Set dlvVersion = Nothing
Set dlvVersions = Nothing
```

### **DocumentProperty Object**

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

### **Using the DocumentProperty Object**

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

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

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

# **FileDialog Object**

<u>FileDialog</u> <sup>L</sup>Multiple objects

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

### Using the FileDialog object

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

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

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

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

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

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

End Sub

# FileDialogFilter Object

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

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

### Using the FileDialogFilter object

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

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

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

# **FileSearch Object**

<u>FileSearch</u> <sup>L</sup>Multiple objects

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

## **Using the FileSearch Object**

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

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

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

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

# **HTMLProject Object**

### HTMLProject HTMLProjectItems

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

#### Using the HTMLProject Object

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

ActiveDocument.HTMLProject.Open (msoHTMLProjectOpenSourceView)

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

# HTMLProjectItem Object

### HTMLProjectItems LHTMLProjectItem

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

### Using the HTMLProjectItem Object

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

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

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

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

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

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

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



## LanguageSettings Object

#### LanguageSettings

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

#### Using the LanguageSettings Object

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

MsoAppLanguageID can be one of these MsoAppLanguageID constants. msoLanguageIDExeMode msoLanguageIDHelp msoLanguageIDInstall msoLanguageIDUI msoLanguageIDUIPrevious

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

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

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

```
If Application.LanguageSettings. _
LanguagePreferredForEditing(msoLanguageIDEnglishUS) Then
MsgBox "U.S. English is one of the chosen editing languagess."
```

End If

# **MsoEnvelope Object**

### **MsoEnvelope**

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

### Using the MsoEnvelope object

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

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

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

## **NewFile Object**

### **NewFile**

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

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

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

### Using the NewFile object

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

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

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

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

# **ODSOColumn Object**

ODSOColumns ODSOColumn

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

### Using the ODSOColumn object

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

## Remarks

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

# **ODSOFilter Object**

ODSOFilters ODSOFilter

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

## **Using the ODSOFilter object**

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

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

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

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

End With End Sub

# **OfficeDataSourceObject Object**

<u>OfficeDataSourceObject</u>

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

### Using the OfficeDataSourceObject object

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

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

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

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

End With End Sub

# **Permission Object**

Permission <sup>L</sup>UserPermission</sup>

The **Permission** property of the Microsoft Office Word 2003 **Document** object, the Microsoft Office Excel 2003 **Workbook** object, and the Microsoft Office PowerPoint 2003 **Presentation** object returns a **Permission** object.

## **Using the Permission Object**

Use the **Permission** object to restrict permissions to the active document and to return or set specific permissions settings.

Use the **Enabled** property to determine whether permissions are restricted on the active document. Use the **Count** property to return the number of users with permissions, and the **RemoveAll** method to reset all existing permissions.

The **DocumentAuthor**, **EnableTrustedBrowser**, **RequestPermissionURL**, and **StoreLicenses** properties provide additional information about permissions settings.

The **Permission** object gives access to a collection of **UserPermission** objects. Use the **UserPermission** object to associate specific sets of rights with individual users. While some permissions granted through the user interface (such as **msoPermissionPrint**) apply to all users, you can use the **UserPermission** object to assign them on a per-user basis with per-user expiration dates.

Microsoft Office Information Rights Management supports the use of administrative permission policies which list users and groups and their document permissions. Use the **ApplyPolicy** method to apply a permission policy, and the **PermissionFromPolicy**, **PolicyName**, and **PolicyDescription** properties to return policy information.

The **Permission** object model is available whether permissions are restricted on the active document or not . The **Permission** property of the **Document**, **Workbook** and **Presentation** objects does not return Nothing when the active document does not have restricted permissions. Use the **Enabled** property to determine whether a document has restricted permissions.

Use of the **Permission** object raises an error when the Windows Rights Management client is not installed.

### Example

The following example returns information about the permissions settings on the active document.

```
Dim irmPermission As Office.Permission
Dim strIRMInfo As String
Set irmPermission = ActiveWorkbook.Permission
If irmPermission. Enabled Then
    strIRMInfo = "Permissions are restricted on this document."
    strIRMInfo = strIRMInfo & " View in trusted browser: " &
        irmPermission.EnableTrustedBrowser & vbCrLf &
        " Document author: " & irmPermission.DocumentAuthor & vb
        " Users with permissions: " & irmPermission.Count & vbCr
        " Cache licenses: " & irmPermission.StoreLicenses & vbCr
        " Request permission URL: " & irmPermission.RequestPermi
    If irmPermission.PermissionFromPolicy Then
        strIRMInfo = strIRMInfo & " Permissions applied from pol
           Policy name: " & irmPermission.PolicyName & vbCrLf &
        н
           Policy description: " & irmPermission.PolicyDescripti
    Else
        strIRMInfo = strIRMInfo & " Default permissions applied.
               Default policy name: " & irmPermission.PolicyName
            п
               Default policy description: " & irmPermission.Pol
    End If
Else
    strIRMInfo = "Permissions are NOT restricted on this documen
End If
MsgBox strIRMInfo, vbInformation + vbOKOnly, "IRM Information"
Set irmPermission = Nothing
```

## **PropertyTest Object**

<u>FileSearch</u> PropertyTests

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
```

# **ScopeFolder Object**

Multiple objects <sup>L</sup><u>ScopeFolder</u>

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

## Using the ScopeFolder object

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

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

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

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

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

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

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

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

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

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

The following example displays the root path of each directory in My Computer.

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

```
Sub DisplayRootScopeFolders()
```

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

# **Script Object**

## Scripts Script

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

#### **Using the Script Object**

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

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

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

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

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

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

```
MsgBox ("Script is located in the header. ")
End If
```

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

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

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

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

# SearchScope Object

FileSearch <sup>L</sup>SearchScopes

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

### Using the SearchScope object

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

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

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

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

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

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

Next ss End With End Sub

# **SharedWorkspace Object**

SharedWorkspace <sup>L</sup>Multiple objects

The **SharedWorkspace** property of a Microsoft Office Word 2003 **Document** object, Microsoft Office Excel 2003 **Workbook** object, and Microsoft Office PowerPoint 2003 **Presentation** object returns a **SharedWorkspace** object which allows the developer to add the active document to a Microsoft Windows SharePoint Services document workspace on the server and to manage other objects in the shared workspace.

### **Using the SharedWorkspace Object**

Use the **SharedWorkspace** object to add the active Word, Excel or PowerPoint document to a Windows SharePoint Services document workspace on the server in order to take advantage of the workspace's collaboration features, or to disconnect or remove the document from the workspace. Use the **SharedWorkspace** object's collections to manage files, folders, links, members and tasks associated with the shared document.

The **SharedWorkspace** object model is available whether or not a document is stored in a workspace. The **SharedWorkspace** property of the **Document**, **Workbook** and **Presentation** objects does not return Nothing when the document is not shared. Use the **Connected** property of the **SharedWorkspace** object to determine whether the active document is in fact saved in and connected to a shared workspace.

Users require appropriate permissions to use the objects, properties and methods in the **SharedWorkspace** object hierarchy.

Use the <u>SharedWorkspaceFiles</u> collection, accessed through the **Files** property of the **SharedWorkspace** object, to manage documents and files saved in a shared workspace.

Use the <u>SharedWorkspaceFolders</u> collection, accessed through the **Folders** property of the **SharedWorkspace** object, to manage subfolders within the main document library folder of a shared workspace.

Use the SharedWorkspaceLinks collection, accessed through the **Links** property of the **SharedWorkspace** object, to manage links to additional documents and information of interest to the members who are collaborating on the document(s) in the shared workspace.

Use the SharedWorkspaceMembers collection, accessed through the **Members** property of the **SharedWorkspace** object, to manage users who have rights to participate in a shared workspace and to collaborate on the shared document(s) saved in the workspace.

Use the SharedWorkspaceTasks collection, accessed through the **Tasks** property

of the **SharedWorkspace** object, to manage tasks assigned to the members who are collaborating on the document(s) in the shared workspace.

Use the **CreateNew** method to create a new document workspace and to add the active document to the workspace. Use the **Name** and **URL** properties to return information about the workspace.

The **SharedWorkspace** object uses a local cache of objects and properties from the server. The developer may need to update this cache before performing certain operations, or to save cached property changes back to the server. Use the **Refresh** method of the **SharedWorkspace** object to refresh the local cache from the server, and the **LastRefreshed** property to determine when the refresh operation last took place. Use the **Save** method of the **SharedWorkspaceLink** and **SharedWorkspaceTask** objects after modifying their properties locally, in order to upload the changes to the server.

Use the **Disconnect** method to disconnect the local copy of the active document from the shared workspace, while leaving the shared copy intact in the workspace. Use the **RemoveDocument** method to remove the shared document from the shared workspace entirely.

Users require appropriate permissions to use the objects, properties and methods in the **SharedWorkspace** object hierarchy. Use the **Role** argument when adding members to the **SharedWorkspaceMembers** collection to specify the set of permissions specific to each workspace member.

The following example displays the properties of the shared workspace to which the active document is connected.

```
Dim swsWorkspace As Office.SharedWorkspace
Dim strSWSInfo As String
Set swsWorkspace = ActiveWorkbook.SharedWorkspace
strSWSInfo = swsWorkspace.Name & vbCrLf & ______
" - URL: " & swsWorkspace.URL & vbCrLf & ______
"The shared workspace contains " & vbCrLf & ______
" - Files: " & swsWorkspace.Files.Count & vbCrLf & ______
" - Folders: " & swsWorkspace.Folders.Count & vbCrLf & ______
" - Links: " & swsWorkspace.Links.Count & vbCrLf & ______
" - Members: " & swsWorkspace.Members.Count & vbCrLf & _______
" - Tasks: " & swsWorkspace.Tasks.Count & vbCrLf & ______
" - Tasks: " & swsWorkspace.Tasks.Count & vbCrLf MsgBox strSWSInfo, vbInformation + vbOKOnly, ______
"Shared Workspace Information"
```

```
Set swsWorkspace = Nothing
```

When using the **SharedWorkspace** object model, it is possible to create conditions where the **SharedWorkspace** object cache is not synchronized with the user interface displayed in the **Shared Workspace** pane of the active document. For example, if the **CreateNew** method programmatically adds the active document to a new workspace while the **Shared Workspace** pane is open, the **Shared Workspace** pane will continue to display the **Create** button. In circumstances like these, if the user makes a selection in the **Shared Workspace** pane that is no longer valid, an error is raised and a refresh operation is carried out to synchronize the display with the current document state and shared workspace data.

The **Document**, **Workbook** and **Presentation** objects also have a **Sync** property which returns a <u>Sync</u> object. Use the **Sync** object and its properties and methods to manage the synchronization of the local and the server copies of the shared document.

# SharedWorkspaceFile Object

SharedWorkspace SharedWorkspaceFiles

The **SharedWorkspaceFile** object represents a file saved in a shared document workspace. Member of the <u>SharedWorkspaceFiles</u> collection.

### Using the SharedWorkspaceFile Object

Use the **SharedWorkspaceFile** object to manage documents and files saved in a shared workspace.

Although the **SharedWorkspaceFile** object has a **URL** property that returns the file's complete path and filename, it does not have a **FileName** property. Use a simple function to extract the filename from the file's URL as in the following example. An additional supporting function decodes escaped space characters in the URL.

```
Private Function FilenameFromURL(FileURL As String) As String
   Dim intLastSeparator As Integer
   FileURL = URLDecode(FileURL)
   intLastSeparator = InStrRev(FileURL, "/")
   FilenameFromURL = Right(FileURL, Len(FileURL) - intLastSeparator
End Function
Private Function URLDecode(URLtoDecode As String) As String
   URLDecode = Replace(URLtoDecode, "%20", " ")
End Function
```

Use the **Item**(*Index*) method of the **SharedWorkspaceFiles** collection to return a specific **SharedWorkspaceFile** object.

Use the **CreatedBy**, **CreatedDate**, **ModifiedBy**, and **ModifiedDate** properties to return information about the history of each file.

The following example returns the number of files in the shared workspace and information about each file, using the supporting functions shown above.

```
Dim swsFile As Office.SharedWorkspaceFile
Dim strFileInfo As String
strFileInfo = "The shared workspace contains " & ______
ActiveWorkbook.SharedWorkspace.Files.Count & " File(s)." & vbCrL
For Each swsFile In ActiveWorkbook.SharedWorkspace.Files
strFileInfo = strFileInfo & FilenameFromURL(swsFile.URL) & v
        " - URL: " & swsFile.URL & vbCrLf & ______
        " - Created by: " & swsFile.CreatedBy & vbCrLf & ______
        " - Created on: " & swsFile.CreatedDate & vbCrLf & _______
        " - Modified by: " & swsFile.ModifiedBy & vbCrLf & _______
        " - Modified on: " & swsFile.ModifiedDate & vbCrLf
```

Next

```
MsgBox strFileInfo, vbInformation + vbOKOnly, _
    "Files in Shared Workspace"
Set swsFile = Nothing
```

# SharedWorkspaceFolder Object

SharedWorkspace SharedWorkspaceFolders

The **SharedWorkspaceFolder** object represents a folder in a shared document workspace. Member of the <u>SharedWorkspaceFolders</u> collection.

#### Using the SharedWorkspaceFolder Object

Use the **SharedWorkspaceFolder** object to manage subfolders within the main document library folder of a shared workspace.

The **Count** property of the **SharedWorkspaceFolders** collection does not include the workspace's main folder and returns 0 (zero) if no subfolders have been created.

The **SharedWorkspaceFolder** object does not expose the **CreatedBy**, **CreatedDate**, **ModifiedBy**, and **ModifiedDate** properties available on the **SharedWorkspaceFile**, **SharedWorkspaceLink**, and **SharedWorkspaceTask** objects.

Use the **Item**(*Index*) method of the **SharedWorkspaceFolders** collection to return a specific **SharedWorkspaceFolder** object.

Use the **FolderName** property to return the name of the shared workspace folder. The following example returns the name of the first subfolder in the **SharedWorkspaceFolders** collection in the format

parentfoldername/foldername.

```
Dim swsFolder As SharedWorkspaceFolder
Set swsFolder = ActiveWorkbook.SharedWorkspace.Folders(1)
MsgBox swsFolder.FolderName, vbInformation + vbOKOnly, "Folder N
Set swsFolder = Nothing
```

# SharedWorkspaceLink Object

SharedWorkspace Links

The **SharedWorkspaceLink** object represents a URL link saved in a shared document workspace. Member of the <u>SharedWorkspaceLinks</u> collection.

#### Using the SharedWorkspaceLink Object

Use the **SharedWorkspaceLink** object to manage links to additional documents and information of interest to the members who are collaborating on the document(s) in the shared workspace.

Use the **Item**(*Index*) method of the **SharedWorkspaceLinks** collection to return a specific **SharedWorkspaceLink** object.

Use the **Description** property to set the link description that will appear on the **Links** tab of the **Shared Workspace** pane and on the workspace web page. Use the **Url** property to set the destination address of the link. Use the **Notes** property to supply additional information about the link.

Use the **Save** method to upload changes to the server after you modify properties of the **SharedWorkspaceLink** object.

Use the **CreatedBy**, **CreatedDate**, **ModifiedBy**, and **ModifiedDate** properties to return information about the history of each link.

The following example modifies the first link in the shared workspace to point to the Microsoft Developer Network home page, then uploads the changes to the server.

```
Dim swsLink As Office.SharedWorkspaceLink
Set swsLink = ActiveWorkbook.SharedWorkspace.Links(1)
With swsLink
    .Description = "MSDN Home Page"
    .URL = "http://msdn.microsoft.com/"
    .Notes = "My favorite site for developers!"
    .Save
End With
Set swsLink = Nothing
```

# SharedWorkspaceMember Object

SharedWorkspace SharedWorkspaceMembers

The **SharedWorkspaceMember** object represents a user who has rights in a shared document workspace. Member of the <u>SharedWorkspaceMembers</u> collection.

### Using the SharedWorkspaceMember Object

Use the **SharedWorkspaceMember** object to manage users who have rights to participate in a shared workspace and to collaborate on the shared document(s) saved in the workspace.

The *Role*specified when the user is added as a member of the workspace (for example, "Reader" or "Contributor") determines that user's rights in the workspace and cannot be accessed or modified later through properties of the **SharedWorkspaceMember** object.

Use the **Item**(*Index*) method of the **SharedWorkspaceMembers** collection to return a specific **SharedWorkspaceMember** object.

Use the **SharedWorkspaceMember** object's 3 distinct name properties to retrieve identifying information about the member.

- the **Name** property returns the members display name or friendly name;
- the **Email** property returns the member's email address; and,
- the **DomainName** property returns the member's domain and user name in the format domain\user.

The following example displays the number of members in the active document's shared workspace, along with their names, domain user names, and email addresses.

```
Dim swsMember As Office.SharedWorkspaceMember
Dim strMemberInfo As String
strMemberInfo = "The shared workspace contains " & ______
ActiveWorkbook.SharedWorkspace.Members.Count & " member(s)."
If ActiveWorkbook.SharedWorkspace.Members.Count > 0 Then
For Each swsMember In ActiveWorkbook.SharedWorkspace.Members
strMemberInfo = strMemberInfo & swsMember.Name & vbCrLf
" - " & swsMember.DomainName & vbCrLf & _____"
" - " & swsMember.Email & vbCrLf
Next
End If
MsgBox strMemberInfo, vbInformation + vbOKOnly, _____"
Members in Shared Workspace"
Set swsMember = Nothing
```

# SharedWorkspaceTask Object

SharedWorkspace LSharedWorkspaceTasks

The **SharedWorkspaceTask** object represents a task in a shared document workspace. Member of the <u>SharedWorkspaceTasks</u> collection.

#### Using the SharedWorkspaceTask Object

Use the **SharedWorkspaceTask** object to manage tasks assigned to the members who are collaborating on the document(s) in the shared workspace.

Use the **Item**(*Index*) method of the **SharedWorkspaceTasks** collection to return a specific **SharedWorkspaceTask** object.

Use the **Title** property to set the text of the task that will appear on the **Tasks** tab of the **Shared Workspace** task pane and on the shared workspace Web page. Use the **Description** property to supply additional information about the task.

Assign the task to a member of the workspace using the **AssignedTo** property and the member's domain user name. Specify a due date for the task using the **DueDate** property.

Use the enumerations for task **Priority** and **Status** to indicate the relative importance of the task and to update the task's status.

Use the **Save** method to upload changes to the server after you modify properties of the **SharedWorkspaceTask** object.

Use the **CreatedBy**, **CreatedDate**, **ModifiedBy**, and **ModifiedDate** properties to return information about the history of each task.

The following example returns the number of tasks in the shared workspace and information about each task.

```
Dim swsTask As Office.SharedWorkspaceTask
Dim strTaskInfo As String
strTaskInfo = "The shared workspace contains " & __
ActiveWorkbook.SharedWorkspace.Tasks.Count & " Task(s)." & vbCrL
For Each swsTask In ActiveWorkbook.SharedWorkspace.Tasks
    strTaskInfo = strTaskInfo & swsTask.Title & vbCrLf & _
        " - Description: " & swsTask.Description & vbCrLf & _
        " - Description: " & swsTask.Description & vbCrLf & _
        " - Assigned to: " & swsTask.AssignedTo & vbCrLf & _
        " - Due date: " & swsTask.DueDate & vbCrLf & _
        " - Priority: " & swsTask.Priority & vbCrLf & _
        " - Status: " & swsTask.Status & vbCrLf
Next
MsgBox strTaskInfo, vbInformation + vbOKOnly, _
```

```
"Tasks in Shared Workspace"
Set swsTask = Nothing
```

## **Signature Object**

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

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

#### Using the Signature object

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

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

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

#### End If

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

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

### SmartDocument Object

#### **SmartDocument**

The **SmartDocument** property of the Microsoft Office Word 2003 **Document** object and the Microsoft Office Excel 2003 **Workbook** object returns a **SmartDocument** object.

### **Using the SmartDocument Object**

Use the **SmartDocument** object to manage the XML expansion pack attached to the active document.

Use the **SmartDocument** object's **SolutionID** and the **SolutionUrl** properties to retrieve information about the XML expansion pack attached to the active document or workbook. Use the **PickSolution** method to allow the user to select an available XML expansion pack from a list to attach to the active document or workbook. Use the **RefreshPane** method to refresh the smart document's **Document Actions** task pane.

The **SmartDocument** object model is available whether or not a document has an XML expansion pack attached. The **SmartDocument** property of the **Document** or **Workbook** objects does not return Nothing when the active document has no XML expansion pack attached. Examine the **SolutionID** property to determine whether the active document has an XML expansion pack attached.



## Sync Object

#### <u>Sync</u>

The **Sync** property of the Microsoft Office Word 2003 **Document** object, the Microsoft Office Excel 2003 **Workbook** object, and the Microsoft Office PowerPoint 2003 **Presentation** object returns a **Sync** object.

### **Using the Sync Object**

Use the **Sync** object to manage the synchronization of the local and server copies of a shared document stored in a Windows SharePoint Services document workspace. The **Status** property returns important information about the current state of synchronization. Use the **GetUpdate** method to refresh the sync status. Use the **LastSyncTime**, **ErrorType**, and **WorkspaceLastChangedBy** properties to return additional information.

See the <u>Status Property</u> for additional information on the differences and conflicts that can exist between the local and server copies of shared documents.

Use the **PutUpdate** method to save local changes to the server. Close and reopen the document to retrieve the latest version from the server when no local changes have been made. Use the **ResolveConflict** method to resolve differences between the local and the server copies, or the **OpenVersion** method to open a different version alongside the currently open local version of the document.

The **GetUpdate**, **PutUpdate**, and **ResolveConflict** methods of the **Sync** object do not return status codes because they complete their tasks asynchronously. The **Sync** object provides important status information through a single event, which the developer can access through the following application-specific events:

- in Word, through the **Sync** event of the **Document** object or the **DocumentSync** event of the **Application** object;
- in Excel, through the **Sync** event of the **Workbook** object or the **WorkbookSync** event of the **Application** object;
- in PowerPoint, through the **PresentationSync** event of the **Application** object.

The **Sync** event described above returns an **msoSyncEventType** value.

MsoSyncEventType can be one of the following msoSyncEventType constants. msoSyncEventDownloadInitiated (0) msoSyncEventDownloadSucceeded (1) msoSyncEventDownloadFailed (2) msoSyncEventUploadInitiated (3)

```
msoSyncEventUploadSucceeded (4)
msoSyncEventUploadFailed (5)
msoSyncEventDownloadNoChange (6)
msoSyncEventOffline (7)
```

The **Sync** object model is available whether sharing and synchronization are enabled or disabled on the active document. The **Sync** property of the **Document**, **Workbook** and **Presentation** objects does not return Nothing when the active document is not shared or synchronization is not enabled. Use the **Status** property to determine whether the document is shared and whether synchronization is enabled.

Not all document synchronization problems raise trappable run-time errors. After using the methods of the **Sync** object, it's a good idea to check the **Status** property; if the **Status** property is **msoSyncStatusError**, check the **ErrorType** property for additional information on the type of error that has occurred.

In many circumstances, the best way to resolve an error condition is to call the **GetUpdate** method. For example, if a call to **PutUpdate** results in an error condition, then a call to **GetUpdate** will reset the status to **msoSyncStatusLocalChanges**.

#### Example

The following example demonstrates various methods of the **Sync** object based on the status of the active document.

```
Dim objSync As Office.Sync
Dim strStatus As String
Set objSync = ActiveDocument.Sync
If objSync.Status > msoSyncStatusNoSharedWorkspace Then
    Select Case objSync.Status
        Case msoSyncStatusConflict
            objSync.ResolveConflict msoSyncConflictMerge
            ActiveDocument.Save
            objSync.ResolveConflict msoSyncConflictClientWins
            strStatus = "Conflict resolved by merging changes."
        Case msoSyncStatusError
            strStatus = "Last error type: " & objSync.ErrorType
        Case msoSyncStatusLatest
            strStatus = "Document copies already in sync."
        Case msoSyncStatusLocalChanges
            objSync.PutUpdate
            strStatus = "Local changes saved to server."
        Case msoSyncStatusNewerAvailable
            objSync.GetUpdate
            strStatus = "Local copy updated from server."
        Case msoSyncStatusSuspended
            objSync.Unsuspend
            strStatus = "Synchronization resumed."
    End Select
Else
    strStatus = "Not a shared workspace document."
End If
MsgBox strStatus, vbInformation + vbOKOnly, "Sync Information"
Set objSync = Nothing
```

## **UserPermission Object**

Permission LuserPermission

The **UserPermission** object associates a set of permissions on the active document with a single user and an optional expiration date. Represents a member of the active document's **Permission** collection.

### **Using the UserPermission Object**

Use the **Add** method of the **Permission** object to grant specific permissions on the active document to a new user, with an optional expiration date. Use the **Remove** method of the **UserPermission** object to remove a user and the user's permissions.

While some permissions granted through the user interface (such as **msoPermissionPrint**) apply to all users, you can use the **UserPermission** object to assign them on a per-user basis with per-user expiration dates.

### Example

The following example determines whether the active document has restricted permissions, then lists users and their assigned permissions by returning the **UserID**, **Permission**, and **ExpirationDate** properties of each **UserPermission** in the document's **Permission** collection.

```
Dim irmPermission As Office.Permission
Dim irmUserPerm As Office.UserPermission
Dim strIRMInfo As String
Set irmPermission = ActiveWorkbook.Permission
If irmPermission. Enabled Then
    For Each irmUserPerm In irmPermission
        strIRMInfo = strIRMInfo & irmUserPerm.UserId & vbCrLf &
            " - Permissions: " & irmUserPerm.Permission & vbCrLf
            " - Expiration Date: " & irmUserPerm.ExpirationDate
   Next
   MsgBox strIRMInfo, _
       vbInformation + vbOKOnly, "IRM Information"
Else
   MsgBox "This document is not restricted.",
        vbInformation + vbOKOnly, "IRM Information"
End If
Set irmUserPerm = Nothing
Set irmPermission = Nothing
```



## WebPageFont Object

#### WebPageFonts LWebPageFont

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

#### Using the WebPageFont Object

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

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

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

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



### **ActivateWizard Method**

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

Resumes or suspends Office Assistant Help during a custom wizard.

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

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

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

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

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

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

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

### Example

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

Assistant.**ActivateWizard** WizardID:=lHelpForWiz, \_ Act:=msoWizardActSuspend, Animation:=msoAnimationGoodbye



# Add Method

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

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

expression.Add(FileName)

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

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

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

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

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

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

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

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

Constant	Description
msoBarLeft, msoBarTop,	Indicates the left, top, right, and bottom
msoBarRight, msoBarBottom	coordinates of the new command bar
msoBarFloating	Indicates that the new command bar won't be docked
	Indicates that the new command bar will be

msoBarPopup	a shortcut menu

#### msoBarMenuBar Macintosh only

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

### Remarks

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

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

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

#### expression.Add(Description, Extensions, Position)

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

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

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

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

### Remarks

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

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

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

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

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

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

Adds a new file type to a file search.

expression.Add(FileType)

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

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

MsoFileType can be one of these MsoFileType constants.

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

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

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

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

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

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

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

MsoCondition can be one of these MsoCondition constants. **msoConditionAnyNumberBetween msoConditionAnytimeBetween msoConditionAtLeast** 

**msoConditionAtMost msoConditionBeginsWith** msoConditionDoesNotEqual **msoConditionEndsWith msoConditionEquals** msoConditionEqualsCompleted msoConditionEqualsDeferred msoConditionEqualsHigh msoConditionEqualsInProgress **msoConditionEqualsLow** msoConditionEqualsNormal msoConditionEqualsNotStarted msoConditionEqualsWaitingForSomeoneElse msoConditionFileTypeAllFiles msoConditionFileTypeBinders msoConditionFileTypeCalendarItem msoConditionFileTypeContactItem msoConditionFileTypeDatabases msoConditionFileTypeDataConnectionFiles msoConditionFileTypeDesignerFiles msoConditionFileTypeDocumentImagingFiles msoConditionFileTypeExcelWorkbooks msoConditionFileTypeJournalItem msoConditionFileTypeMailItem msoConditionFileTypeNoteItem msoConditionFileTypeOfficeFiles msoConditionFileTypeOutlookItems msoConditionFileTypePhotoDrawFiles msoConditionFileTypePowerPointPresentations msoConditionFileTypeProjectFiles msoConditionFileTypePublisherFiles msoConditionFileTypeTaskItem msoConditionFileTypeTemplates

msoConditionFileTypeVisioFiles msoConditionFileTypeWebPages msoConditionFileTypeWordDocuments **msoConditionFreeText** msoConditionIncludes msoConditionIncludesFormsOf msoConditionIncludesNearEachOther msoConditionIncludesPhrase msoConditionInTheLast msoConditionInTheNext **msoConditionIsExactly** msoConditionIsNo msoConditionIsNot msoConditionIsYes msoConditionLastMonth msoConditionLastWeek **msoConditionLessThan msoConditionMoreThan msoConditionNextMonth** msoConditionNextWeek msoConditionNotEqualToCompleted msoConditionNotEqualToDeferred msoConditionNotEqualToHigh msoConditionNotEqualToInProgress msoConditionNotEqualToLow msoConditionNotEqualToNormal msoConditionNotEqualToNotStarted msoConditionNotEqualToWaitingForSomeoneElse **msoConditionOn msoConditionOnOrAfter** msoConditionOnOrBefore **msoConditionThisMonth msoConditionThisWeek** 

#### msoConditionToday msoConditionTomorrow msoConditionYesterday

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

MsoFilterComparison can be one of these MsoFilterComparison constants.

msoFilterComparisonContains msoFilterComparisonEqual msoFilterComparisonGreaterThan msoFilterComparisonGreaterThanEqual msoFilterComparisonIsBlank msoFilterComparisonIsNotBlank msoFilterComparisonLessThan msoFilterComparisonLessThanEqual msoFilterComparisonNotContains msoFilterComparisonNotEqual

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

MsoFilterConjunction can be one of these MsoFilterConjunction constants. msoFilterConjunctionAnd msoFilterConjunctionOr

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

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

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

Creates a new set of permissions on the active document for the specified user. Returns a **UserPermission** object.

expression.Add(UserID, Permission, ExpirationDate)

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

*UserID* Required **String**. The email address (in the format *user@domain.com*) of the user to whom permissions on the active document are being granted.

*Permission* Optional <u>msoPermission</u>. The permissions on the active document that are being granted to the specified user.

**MsoPermission** can be one or a combination of these **MsoPermission** constants.

msoPermissionChange msoPermissionEdit msoPermissionExtract msoPermissionFullControl msoPermissionObjModel msoPermissionPrint msoPermissionRead msoPermissionSave msoPermissionView

*ExpirationDate* Optional **Date**. The expiration date for the permissions that are being granted.

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

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

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

*expression* Required. The **Scripts** collection.

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

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

MsoScriptLocation can be one of these MsoScriptLocation constants. **msoScriptLocationInBody** *default* **msoScriptLocationInHead** 

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

MsoScriptLanguage can be one of these MsoScriptLanguage constants.

```
msoScriptLanguageASP
msoScriptLanguageJava
msoScriptLanguageOther
msoScriptLanguageVisualBasic default
```

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

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

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

### Remarks

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

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

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

Adds a search folder to a file search.

expression.Add(ScopeFolder)

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

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

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

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

expression.Add

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

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

Adds a file to the document library in a shared workspace. Returns a **SharedWorkspaceFile** object.

expression.Add(FileName, ParentFolder, OverwriteIfFileAlreadyExists, KeepInSync)

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

*FileName* Required **String**. The path and filename of the file to be added to

the current shared workspace.

**ParentFolder** Optional **SharedWorkspaceFolder**. The subfolder in which to place the file, if not the main document library folder within the shared workspace. Add the file to the main document library folder by leaving this optional argument empty.

*OverwriteIfFileAlreadyExists* Optional **Boolean**. **True** to overwrite an existing file by the same name. Default is **False**.

*KeepInSync* Optional **Boolean**. **True** to keep the local copy of the document synchronized with the copy in the shared workspace. Default is **False**.

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

Adds a folder to the document library in a shared workspace. Returns a **SharedWorkspaceFolder** object.

expression.Add(FolderName, ParentFolder)

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

*FolderName* Required **String**. The name of the folder to be added to the current shared workspace.

**ParentFolder** Optional **SharedWorkspaceFolder**. The subfolder in which to place the new folder, if not the main document library folder within the shared workspace. Add the folder to the main document library folder by leaving this optional argument empty.

Add method as it applies to the SharedWorkspaceLinks object.

Adds a link to the list of links in a shared workspace. Returns a **SharedWorkspaceLink** object.

expression.Add(URL, Description, Notes)

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

*URL* Required **String**. The URL of the web site to which a link is being added.

*Description* Optional **String**. Optional description of the link.

*Notes* Optional **String**. Optional notes about the link.

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

Adds a member to the list of members in a shared workspace. Returns a **SharedWorkspaceMember** object.

expression.Add(Email, DomainName, DisplayName, Role)

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

*Email* Required **String**. The new member's email address in the format *user@domain.com*. Raises an error if the user is not a valid candidate for membership in the shared workspace.

*DomainName* Required **String**. The new member's Windows user name in the format *domain\user*.

*DisplayName* Required **String**. The friendly name to display for the new member.

*Role* Optional String. An optional role that determines the tasks the new member can accomplish in the shared workspace; for example, "Contributor". An invalid role name raises an error.

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

Adds a task to the list of tasks in a shared workspace. Returns a **SharedWorkspaceTask** object.

expression.Add(Title, Status, Priority, Assignee, Description, Due Date)

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

*Title* Required **String**. The title of the new task.

*Status* Optional <u>msoSharedWorkspaceTaskStatus</u>. The status of the new task. Default is **msoSharedWorkspaceTaskStatusNotStarted**.

MsoSharedWorkspaceTaskStatus can be one of these msoSharedWorkspaceTaskStatus constants. msoSharedWorkspaceTaskStatusComplete msoSharedWorkspaceTaskStatusDeferred msoSharedWorkspaceTaskStatusInProgress msoSharedWorkspaceTaskStatusNotStarted msoSharedWorkspaceTaskStatusWaiting

*Priority* Optional <u>msoSharedWorkspaceTaskPriority</u>. The priority of the new task. Default is **msoSharedWorkspaceTaskPriorityNormal**.

MsoSharedWorkspaceTaskPriority can be one of these MsoSharedWorkspaceTaskPriority constants. msoSharedWorkspaceTaskPriorityHigh MsoSharedWorkspaceTaskPriorityLow MsoSharedWorkspaceTaskPriorityNormal

*Assignee* Optional **SharedWorkspaceMember**. The member to whom the new task is assigned.

*Description* Optional **String**. The description of the new task.

*DueDate* Optional **Date**. The due date of the new task.

### Remarks

The schema that defines shared workspace tasks and their properties for a SharePoint Products and Technologies site can be modified on the server in such a way that the **Add** method of the **SharedWorkspaceTasks** collection may raise an error, or may disregard the values of certain arguments. In particular, the task status and priority enumerations can be customized. Some examples of the problems that can result are mentioned below:

- If a *Status* argument is supplied, and the status field has been removed from the customized tasks schema, the argument will be ignored and no error will be raised.
- If a *Status* value is supplied that lies outside the status values recognized by the customized tasks schema, the argument will be ignored, the default value will be used, and no error will be raised.
- If a new required field has been added to the customized tasks schema, then the **Add** method will fail with an erro, and it will no longer be possible to use the **Add** method to add new tasks.

### Example

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

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

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

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

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

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

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

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

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

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

The following example assigns a combination of read and edit permissions on the current document to a user and specifies an expiration date for these document permissions.

```
Dim objUserPerm As Office.UserPermission
Set objUserPerm = ActiveWorkbook.Permission.Add( _
    "user@domain.com", _
    msoPermissionRead + msoPermissionEdit, #12/31/2005#)
MsgBox "Permissions added for " & _
    objUserPerm.UserId, _
    vbInformation + vbOKOnly, _
    "Permissions Added"
Set objUserPerm = Nothing
```

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

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

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

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

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

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

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

The following example adds a new file to the files collection of the shared workspace.

```
Dim swsfile As Office.SharedWorkspaceFile
```

```
Set swsfile = ActiveWorkbook.SharedWorkspace.Files.Add( _
    "C:\MyWorkbook.xls", _
    , True, True)
MsgBox "New file URL: " & swsfile.URL, _
    vbInformation + vbOKOnly, _
    "New File in Shared Workspace Files"
Set swsfile = Nothing
```

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

The following example adds a new folder to the folders collection of the shared workspace.

```
Dim swsFolder As Office.SharedWorkspaceFolder
Set swsFolder = ActiveWorkbook.SharedWorkspace.Folders.Add("MyNe
MsgBox "New folder: " & swsFolder.FolderName, _
vbInformation + vbOKOnly, _
"New Folder in Shared Workspace"
Set swsFolder = Nothing
```

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

The following example adds a new link to the links collection of the shared workspace.

```
Dim swsLink As Office.SharedWorkspaceLink
Set swsLink = ActiveWorkbook.SharedWorkspace.Links.Add( _
    "http://msdn.microsoft.com", _
    "Microsoft Developer Network Home Page", _
    "My favorite developer site!")
MsgBox "New link: " & swsLink.Description, _
    vbInformation + vbOKOnly, _
    "New Link in Shared Workspace"
Set swsLink = Nothing
```

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

The following example adds a new member to the members collection of the shared workspace in the role of a site contributor.

```
Dim swsMember As Office.SharedWorkspaceMember
Set swsMember = ActiveWorkbook.SharedWorkspace.Members.Add( _
    "user@domain.com", _
    "domain\user", _
    "New User", _
```

```
"Contributor")
MsgBox "New member: " & swsMember.Name, _
vbInformation + vbOKOnly, _
"New Member in Shared Workspace)"
Set swsMember = Nothing
```

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

The following example adds a new task to the tasks collection of the shared workspace, specifies a due date, and assigns the task to the first member of the shared workspace.

```
Dim swsTask As Office.SharedWorkspaceTask
Dim swsMember As Office.SharedWorkspaceMember
Set swsMember = ActiveWorkbook.SharedWorkspace.Members(1)
Set swsTask = ActiveWorkbook.SharedWorkspace.Tasks.Add( _
    "Complete document by year-end", _
    msoSharedWorkspaceTaskStatusNotStarted, _
    msoSharedWorkspaceTaskPriorityNormal, _
    swsMember, _
    "My first shared workspace task", #12/31/2005#)
MsgBox "New task: " & swsTask.Title, _
    vbInformation + vbOKOnly, _
    "New Task in Shared Workspace"
Set swsMember = Nothing
Set swsTask = Nothing
```



# **AddItem Method**

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

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

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

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

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

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

## Example

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

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

## **AddToSearchFolders Method**

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

#### $expression. {\bf Add To Search Folders}$

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

### Remarks

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

## Example

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

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

# **ApplyFilter Method**

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

#### expression.ApplyFilter

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

### Example

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

End Sub

# **ApplyPolicy Method**

Applies the specified permission policy to the active document.

### expression.ApplyPolicy(Filename)

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

*Filename* Required **String**. The path and filename of the permission policy template file.

### Remarks

Microsoft Office Information Rights Management supports the use of administrative permission policies which list users and groups and their document permissions. The **ApplyPolicy** method applies a permission policy to the active document.

### Example

The following example enables permissions on the active document and applies an administrative permission policy.

```
Dim irmPermission As Office.Permission
Set irmPermission = ActiveWorkbook.Permission
Dim strIRMInfo As String
Select Case irmPermission.Enabled
    Case True
        strIRMInfo = "Permissions are already restricted on this
    Case False
        With irmPermission
            .Enabled = True
            .ApplyPolicy ("\\server\share\permissionpolicy.xml")
        End With
        strIRMInfo = "Permissions are now restricted on this doc
            vbCrLf & _
            " and the permission policy has been applied."
End Select
MsgBox strIRMInfo, vbInformation + vbOKOnly, "IRM Information"
Set irmPermission = Nothing
```



# **Clear Method**

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

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

### expression.Clear

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

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

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

# **ClearFileList Method**

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

expression.ClearFileList

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

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

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

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

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

# **Close Method**

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

expression.Close

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

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

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

# **Commit Method**

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

#### expression.Commit

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

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

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



# **Copy Method**

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

### expression.Copy(Bar, Before)

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

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

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

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

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



# **CopyFace Method**

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

### expression.CopyFace

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

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

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

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

## **CreateNew Method**

Creates a new document workspace on the server and adds the active document to the new shared workspace.

#### expression.CreateNew(URL, Name)

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

*URL* Optional **String**. The URL for the parent folder in which the new shared workspace is to be created. If you do not supply a URL, the new shared workspace is created in the user's default server location.

*Name* Optional **String**. The name of the new shared workspace. Defaults to the name of the active document without its file extension. For example, if you create a shared workspace for "Budget.xls", the name of the new shared workspace becomes "Budget".

Use the **CreateNew** method to create a new shared workspace for the active document. Omit the 2 optional arguments to create the workspace using the name of the active document in the user's default server location.

The **CreateNew** method raises an error if the active document has changes that have not been saved. The document must be saved before it can be added to a shared workspace.

The following example creates a new shared workspace at the URL http://server/sites/mysite/, names the workspace "My Shared Budget Document", and adds the active document to the workspace. The **URL** property of the new shared workspace returns

http://server/sites/mysite/My%20Shared%20Budget%20Document/, the **Name** property returns "My Shared Budget Document, and **Count** property of the **Files** collection shows a single file.

```
Dim sws As Office.SharedWorkspace
Dim strSWSInfo As String
Set sws = ActiveWorkbook.SharedWorkspace
sws.CreateNew "http://server/sites/mysite/", "My Shared Budget D
strSWSInfo = "Name: " & sws.Name & vbCrLf & _
    "URL: " & sws.URL & vbCrLf & _
    "File(s): " & sws.Files.Count
MsgBox strSWSInfo, vbInformation + vbOKOnly, _
    "New Shared Workspace Information"
Set sws = Nothing
```



## **Delete Method**

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

Deletes the specified object from its collection.

#### expression.Delete(FileName)

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

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

Delete method as it applies to the **CommandBar**, **Script**, **Scripts**, and **Signature** objects.

Deletes the specified object from the collection.

#### expression. Delete

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

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

Delete method as it applies to the **CommandBarButton**, **CommandBarComboBox**, **CommandBarControl**, and **CommandBarPopup** <u>objects</u>.

Deletes the specified object from its collection.

#### expression.Delete(Temporary)

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

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

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

Removes a document library version from the **DocumentLibraryVersions** collection.

#### expression.**Delete**

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

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

Removes a custom document property.

expression. Delete

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

You cannot delete a built-in document property.

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

Removes a file dialog filter.

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

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

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

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

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

### expression.Delete(Index, DeferUpdate)

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

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

*DeferUpdate* Optional **Boolean**.

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

Deletes the current shared workspace and all data within it.

expression. Delete

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

The Delete method will fail if the user does not have permission to delete the current shared workspace.

If the active document was opened directly from the workspace, then after

deleting the workspace, the document will no longer be saved on disk. The developer must remember to save the document to a new location before closing it, or the document will cease to exist.

Delete method as it applies to the **SharedWorkspaceFile**, **SharedWorkspaceLink**, **SharedWorkspaceMember**, and **SharedWorkspaceTask** objects.

Deletes the current object.

#### expression. Delete

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

The Delete method will fail if the user does not have permission to delete the current object from the shared workspace.

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

Deletes the current shared workspace folder and all data within it.

### expression.Delete(DeleteEvenIfFolderContainsFiles)

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

*DeleteEvenIfFolderContainsFiles* Optional **Boolean**. **True** to delete the folder without warning even if the folder contains files. Default is **False**.

The Delete method will fail if the user does not have permission to delete the current folder from the shared workspace.

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

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

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

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

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

ActiveDocument.CustomDocumentProperties("CustomNumber").Delete

## **Disconnect Method**

Disconnects the local copy of the active document from the shared workspace.

expression.Disconnect()

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

Use the **Disconnect** method to detach the local copy of the active document from the shared workspace. This method leaves the shared document on the server; however, the local copy will no longer be synchronized with the shared copy and will no longer benefit from the other collaboration features of the shared workspace. Use the **RemoveDocument** method to remove the shared document from the server.

The following example determines whether the active document is connected to a shared workspace, then offers the user the option of disconnecting it.

```
Dim r As Long
If ActiveWorkbook.SharedWorkspace.Connected Then
    r = MsgBox("Are you sure you want to disconnect this documen
        vbQuestion + vbOKCancel, "Are you sure?")
    If r = vbOK Then
        ActiveWorkbook.SharedWorkspace.Disconnect
        MsgBox "The document has been disconnected.", _
            vbInformation + vbOKOnly, "Disconnected"
    Else
        MsgBox "Disconnect cancelled.", _
            vbInformation + vbOKOnly, "Still Connected"
    End If
Else
        MsgBox "The active document is not connected to a shared wor
        vbInformation + vbOKOnly, "Not Connected"
    End If
Else
        MsgBox "The active document is not connected to a shared wor
        vbInformation + vbOKOnly, "Not Connected"
    End If
```



# **DoAlert Method**

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

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

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

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

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

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

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

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

MsoAlertIconType can be one of these MsoAlertIconType constants. msoAlertIconCritical msoAlertIconInfo

### msoAlertIconNoIcon msoAlertIconQuery msoAlertIconWarning

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

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

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

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

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

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

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

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

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

## **EndWizard Method**

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

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

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

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

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

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

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

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

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

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



### **Execute Method**

Execute method as it applies to the **FileSearch** object.

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

expression.Execute(SortBy, SortOrder, AlwaysAccurate)

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

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

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

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

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

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

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

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

#### expression. Execute

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

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

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

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

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

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

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



## **FindControl Method**

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

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

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

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

MsoControlType type can be one of these MsoControlType constants. **msoControlActiveX** msoControlAutoCompleteCombo **msoControlButton msoControlButtonDropdown msoControlButtonPopup** msoControlComboBox **msoControlCustom msoControlDropdown msoControlEdit msoControlExpandingGrid** msoControlGauge msoControlGenericDropdown **msoControlGraphicCombo** msoControlGraphicDropdown msoControlGraphicPopup msoControlGrid **msoControlLabel msoControlLabelEx msoControlOCXDropDown msoControlPane msoControlPopup** msoControlSpinner

msoControlSplitButtonMRUPopup msoControlSplitButtonPopup msoControlSplitDropdown msoControlSplitExpandingGrid msoControlWorkPane

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

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

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

*Recursive* Optional **Boolean**. **True** to include the command bar and all of its pop-up subtoolbars in the search. This argument only applies to the **CommandBar** object. The default value is **False**.

#### 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**.

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

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



## **FindControls Method**

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

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

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

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

MsoControlType type can be one of these MsoControlType constants.

**msoControlActiveX** msoControlAutoCompleteCombo **msoControlButton** msoControlButtonDropdown **msoControlButtonPopup** msoControlComboBox **msoControlCustom msoControlDropdown msoControlEdit msoControlExpandingGrid** msoControlGauge msoControlGenericDropdown **msoControlGraphicCombo** msoControlGraphicDropdown msoControlGraphicPopup **msoControlGrid msoControlLabel msoControlLabelEx** msoControlOCXDropDown **msoControlPane msoControlPopup** 

msoControlSpinner msoControlSplitButtonMRUPopup msoControlSplitButtonPopup msoControlSplitDropdown msoControlSplitExpandingGrid msoControlWorkPane

*Id* Optional **Variant**. The control's identifier.

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

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

### Remarks

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

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

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

## **GetUpdate Method**

Compares the local version of the shared document to the version on the server.

#### expression.GetUpdate

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

#### Remarks

Use the **GetUpdate** method to compare the local version of the shared document to the version on the server and to refresh the sync status.

Not all document synchronization problems raise trappable run-time errors. After performing an operation using the **Sync** object, it's a good idea to check the **Status** property; if the **Status** property is **msoSyncStatusError**, check the **ErrorType** property for additional information on the type of error that has occurred.

In many circumstances, the best way to resolve an error condition is to call the **GetUpdate** method. For example, if a call to **PutUpdate** results in an error condition, then a call to **GetUpdate** will reset the status to **msoSyncStatusLocalChanges**.

The following example compares the local and server copies of the document using the **GetUpdate** method and reports whether the server has a newer copy.

```
Dim objSync As Office.Sync
Dim strStatus As String
Set objSync = ActiveDocument.Sync
objSync.GetUpdate
If objSync.Status = msoSyncStatusNewerAvailable Then
    strStatus = "A newer version is available on the server."
    MsgBox strStatus, vbInformation + vbOKOnly, "Sync Informatic
End If
Set objSync = Nothing
```

# **Help Method**

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

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

#### expression.Help

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

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

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



## **Item Method**

Item method as it applies to the **COMAddIns** object.

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

expression.Item(Index)

*expression* Required. The specified **COMAddIns** collection.

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

Item method as it applies to the FileDialogFilters object.

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

expression.Item(Index)

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

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

Item method as it applies to the **FileDialogSelectedItems** object.

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

expression.Item(Index)

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

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

Item method as it applies to the **HTMLProjectItems** object.

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

#### expression.Item(Index)

*expression* Required. An **HTMLProjectItems** collection.

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

Item method as it applies to the **Scripts** object.

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

expression.Item(Index)

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

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

#### Remarks

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

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

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

# LoadFromFile Method

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

expression.LoadFromFile(Filename)

expression An HTMLProjectItem object.

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

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

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



## **Move Method**

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

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

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

expression.Move(Bar, Before)

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

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

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

Move method as it applies to the **Assistant** object.

Moves the Office Assistant to the specified location.

expression.Move(xLeft, yTop)

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

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

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

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

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

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

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

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

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

## **NewSearch Method**

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

Resets all the search criteria settings to their default settings.

#### expression.NewSearch

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

### 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.

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

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



# **Open Method**

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

Opens the specified version of the shared document from the **DocumentLibraryVersions** collection in read-only mode.

expression.Open()

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

As it applies to the **HTMLProject** and **HTMLProjectItem** objects.

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

#### expression.Open(OpenKind)

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

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

MsoHTMLProjectOpen can be one of these MsoHTMLProjectOpen constants. msoHTMLProjectOpenSourceView msoHTMLProjectOpenTextView

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

As it applies to the **DocumentLibraryVersion** object

The following example opens the previous saved version of the active document in read-only mode.

Dim dlvVersions As Office.DocumentLibraryVersions
Set dlvVersions = ActiveDocument.DocumentLibraryVersions
dlvVersions(dlvVersions.Count - 1).Open
Set dlvVersions = Nothing

As it applies to the HTMLProject object

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

AppActiveWorkbook.HTMLProject.Open \_
 (msoHTMLProjectOpenSourceView)

As it applies to the HTMLProjectItem object

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

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



# **OpenVersion Method**

Opens a different version of the shared document alongside the currently open local version.

expression.OpenVersion(SyncVersionType)

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

*SyncVersionType* Required <u>MsoSyncVersionType</u>.

MsoSyncVersionType can be one of the following msoSyncVersionType constants. **msoSyncVersionLastViewed** (0) **msoSyncVersionServer** (1)

Use the **OpenVersion** method to open the last version viewed (**msoSyncVersionLastViewed**) or the server copy (**msoSyncVersionServer**) of the shared document alongside the currently open local version.

The **msoSyncVersionLastViewed** option displays the copy of the document that is created whenever the user overwrites the local copy with the server copy. For example, if you call the **ResolveConflict** method with the **msoSyncConflictServerWins** option, then your local changes are saved and can be viewed by calling **OpenVersion(msoSyncVersionLastViewed)**.

Not all document synchronization problems raise trappable run-time errors. After performing an operation using the **Sync** object, it's a good idea to check the **Status** property; if the **Status** property is **msoSyncStatusError**, check the **ErrorType** property for additional information on the type of error that has occurred.

The following example prompts the user to open the server copy of the shared document alongside the currently open local version.

```
Dim objSync As Office.Sync
Dim lngChoice As VbMsgBoxResult
Set objSync = ActiveDocument.Sync
lngChoice = MsgBox("View server copy?", _
vbQuestion + vbOKCancel, "Open Server Version?")
If lngChoice = vbOK Then
objSync.OpenVersion msoSyncVersionServer
End If
Set objSync = Nothing
```



## **PasteFace Method**

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

#### expression.PasteFace

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

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

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

# **PickSolution Method**

Displays a dialog box which allows the user to choose an available XML expansion pack to attach to the active Microsoft Office Word 2003 document or Microsoft Office Excel 2003 workbook.

expression.PickSolution(ConsiderAllSchemas)

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

*ConsiderAllSchemas* Optional **Boolean**. **True** displays all available XML expansion packs installed on the user's computer. **False** displays only XML expansion packs applicable to the active document. Default value is **False**.

Use the **PickSolution** method to allow the user to select an XML expansion pack from a list. The schema attached to the active document or workbook determines which XML expansion packs are applicable.

The **PickSolution** method does not return a value to indicate whether the user selected an XML expansion pack or clicked **Cancel** in the dialog box. Check the **SolutionID** property after calling **PickSolution** to determine whether an XML expansion pack has been attached.

If the smart document developer has failed to specify "targetApplication" in the XML expansion pack manifest file, the list displayed by **PickSolution** may include XML expansion packs that are not targeted to the active application; for example, an Excel user may see XML expansion packs targeted exclusively to Word. In these circumstances, the user may select an XML expansion pack that is not appropriate for the active application.

For more information on smart documents or XML expansion packs for smart documents, please see the Smart Document Software Development Kit (SDK) on the Microsoft Developer Network (MSDN) Web site.

The following example checks the **SolutionID** property to determine whether the active Microsoft Word document already has an attached XML expansion pack; if not, it displays a dialog box which allows the user to choose an available XML expansion pack. It then displays the properties of the smart document.

```
Dim objSmartDoc As Office.SmartDocument
Dim strSmartDocInfo As String
Set objSmartDoc = ActiveDocument.SmartDocument
If objSmartDoc.SolutionID = "None" Or objSmartDoc.SolutionID = "
objSmartDoc.PickSolution True
End If
If objSmartDoc.SolutionID <> "None" And objSmartDoc.SolutionID <
strSmartDocInfo = "SolutionID: " & objSmartDoc.SolutionID &
"SolutionURL: " & objSmartDoc.SolutionURL
MsgBox strSmartDocInfo, vbInformation + vbOKOnly, "Smart Doc
Else
MsgBox "The user clicked Cancel."
End If
Set objSmartDoc = Nothing
```

## **PutUpdate Method**

Updates the server copy of the shared document with the local copy.

#### expression.PutUpdate()

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

The **PutUpdate** method can encounter a conflict condition if the client is unaware of recent changes to the server copy of the shared document. Call the **GetUpdate** method before calling **PutUpdate** to refresh the status of the server copy and to detect a possible conflict.

The **PutUpdate** method raises a run-time error if the local document has unsaved changes.

Not all document synchronization problems raise trappable run-time errors. After performing an operation using the **Sync** object, it's a good idea to check the **Status** property; if the **Status** property is **msoSyncStatusError**, check the **ErrorType** property for additional information on the type of error that has occurred.

In many circumstances, the best way to resolve an error condition is to call the **GetUpdate** method. For example, if a call to **PutUpdate** results in an error condition, then a call to **GetUpdate** will reset the status to **msoSyncStatusLocalChanges**.

The following example updates the server copy of the document from the local copy using the **PutUpdate** method if the local copy has been edited.

```
Dim objSync As Office.Sync
Dim strStatus As String
Set objSync = ActiveDocument.Sync
If objSync.Status = msoSyncStatusLocalChanges Then
    objSync.PutUpdate
    strStatus = "Local changes saved to server."
    MsgBox strStatus, vbInformation + vbOKOnly, "Sync Informatic
End If
Set objSync = Nothing
```

## **Refresh Method**

Refreshes the local cache of the **SharedWorkspace** object's files, folders, links, members, and tasks from the server.

expression.Refresh()

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

Use the Refresh method to ensure that you are working with the most up-to-date list of objects and their properties from the shared workspace.

The following example refreshes the shared workspace if it has not been refreshed in the last 3 minutes. The example also handles the error condition where the workspace has not yet been refreshed.

err\_NeverRefreshed:

ActiveWorkbook.SharedWorkspace.Refresh

# **RefreshDocument Method**

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

#### expression.RefreshDocument(Refresh)

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

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

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

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

ActiveWorkbook.HTMLProject.RefreshDocument (True)

## **RefreshPane Method**

Refreshes the **Document Actions** task pane for the active Microsoft Office Word 2003 document or Microsoft Office Excel 2003 workbook.

expression.RefreshPane()

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

The **RefreshPane** method raises an error if the active document does not have an XML expansion pack attached.

The following example determines whether the active Microsoft Excel workbook has an XML expansion pack attached. If so, it refreshes the smart document's Document Actions task pane.

```
Dim objSmartDoc As Office.SmartDocument
Set objSmartDoc = ActiveWorkbook.SmartDocument
If objSmartDoc.SolutionID <> "None" Then
        objSmartDoc.RefreshPane
Else
        MsgBox "No XML expansion pack attached."
End If
```

# **RefreshProject Method**

Refreshes the specified HTML project in the Microsoft Script Editor.

#### expression.RefreshProject

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

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

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

ActiveWorkbook.HTMLProject.RefreshProject (True)

# **RefreshScopes Method**

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

#### expression.RefreshScopes

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

The following example displays all of the currently available **ScopeFolder** objects on the C:\ drive in the My Computer scope and demonstrates the need for the **RefreshScopes** method when changes to the folder list occur.

```
Sub TestRefreshScopesMethod()
' Displays what happens before and after the RefreshScopes
' method is called when a new folder is added to the list
' of scope folders.
    ' List before the folder is created.
    Call ListFolderNames
    ' Create a new folder on the C:\ drive in My Computer.
    ' An error will occur if this folder already exists.
    MkDir Path:="C:\Delete_After_Using"
    ' List after the folder is created.
    ' The newly-created folder does not appear in the list.
    Call ListFolderNames
    ' Refresh the list of folders.
    Application.FileSearch.RefreshScopes
    ' The newly-created folder now appears in the list.
    Call ListFolderNames
End Sub
Sub ListFolderNames()
    Dim i as Integer
    Dim strResults As String
    ' Loop through all the top-level folder names on the C:\ drive
     in My Computer and report the results.
    '.SearchScopes.Item(1) = "My Computer"
    '.ScopeFolders.Item(2) = "C:\"
   With Application.FileSearch.SearchScopes.Item(1). _
        ScopeFolder.ScopeFolders.Item(2)
        For i = 1 To .ScopeFolders.Count
            strResults = strResults & .ScopeFolders. _
                Item(i).Name & vbCrLf
        Next i
```

MsgBox "Folder Names on C:\...." & vbCrLf & strResults

End With

End Sub



## **ReleaseFocus Method**

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

#### expression.ReleaseFocus

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

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

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



## **Remove Method**

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

Removes the specified object from the collection.

```
expression.Remove(Index)
```

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

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

Remove method as it applies to the NewFile object.

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

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

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

*FileName* Required **String**. The name of the file reference.

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

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

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

Remove method as it applies to the **UserPermission** object.

Removes the specified **UserPermission** object from the **Permission** collection of the active document.

```
expression.Remove()
```

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

The **UserPermission** object associates a set of permissions on the active document with a single user and an optional expiration date. The **Remove** method removes the user and the set of user permissions determined by the specified **UserPermission** object.

As it applies to the **FileTypes**, **PropertyTests**, and **SearchFolders** objects.

This example removes the first search criterion from the collection.

Application.FileSearch.PropertyTests.Remove(1)

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

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

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

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

The following example removes the second user's permissions on the active document from the document's **Permission** collection.

# **RemoveAll Method**

Removes all **UserPermission** objects from the **Permission** collection of the active document.

expression.RemoveAll()

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

The **RemoveAll** method removes all **UserPermissions** that have been added to the **Permission** collection and disables restrictions on the active document. After calling the **RemoveAll** method, the **Enabled** property of the **Permission** object returns **False** and the **Count** property returns 0 (zero).

The following example uses the **RemoveAll** method to remove all user permissions and to disable restrictions on the active document.

```
Dim irmPermission As Office.Permission
Set irmPermission = ActiveWorkbook.Permission
If irmPermission.Enabled Then
    irmPermission.RemoveAll
    MsgBox "All permissions removed." & vbCrLf & _
        "Count: " & irmPermission.Count & vbCrLf & _
        "Enabled: " & irmPermission.Enabled, _
        vbInformation + vbOKOnly, "IRM Information"
Else
        MsgBox "This document is not restricted.", _
        vbInformation + vbOKOnly, "IRM Information"
End If
Set irmPermission = Nothing
```

## **RemoveDocument Method**

Removes the active document from the shared workspace.

#### expression.RemoveDocument()

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

If the user does not have permission to remove the shared workspace document from the server, then the server copy remains intact, but the local copy of the document is disconnected from the shared workspace. In the case where the document has been opened directly from the workspace, then removed from the workspace using **RemoveDocument**, the document must be saved to another location before closing; otherwise, it will remain in the workspace.

Use the **Disconnect** method to detach the local copy of the document from the shared workspace without removing the shared copy.

The following example determines whether the active document is connected to a shared workspace, then offers the user the option of removing the document from the workspace.

```
Dim r As Long
If ActiveWorkbook.SharedWorkspace.Connected Then
    r = MsgBox("Are you sure you want to remove this document?",
        vbQuestion + vb0KCancel, "Are you sure?")
    If r = vb0K Then
        ActiveWorkbook.SharedWorkspace.RemoveDocument
        MsgBox "The document has been removed.",
        vbInformation + vb0KOnly, "Removed"
    Else
        MsgBox "Removal cancelled.",
        vbInformation + vb0KOnly, "Still In Workspace"
    End If
Else
        MsgBox "The active document is not connected to a shared wor
        vbInformation + vb0KOnly, "Not Connected"
End If
End If
```



## **RemoveItem Method**

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

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

#### expression.RemoveItem(Index)

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

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

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

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



## **Reset Method**

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

expression.Reset

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

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

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

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

# **ResetFileList Method**

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

#### expression.ResetFileList

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

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

**Caution** Incorrectly editing the registry may severely damage your operating system, requiring you to reinstall it. Microsoft cannot guarantee that problems resulting from editing the registry incorrectly can be resolved. Before editing the registry, back up any valuable data. For the most recent information about using and protecting your computer's registry, see Microsoft Windows Help.

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

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

# **ResetTips Method**

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

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

#### expression.ResetTips

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

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

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

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



# **ResolveConflict Method**

Resolves conflicts between the local and the server copies of a shared document.

expression.ResolveConflict(SyncConflictResolution)

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

*SyncConflictResolution* Required <u>MsoSyncConflictResolutionType</u>.

MsoSyncConflictResolutionType can be one of these soSyncConflictResolutionType constants. **msoSyncConflictClientWins** (0) **msoSyncConflictServerWins** (1) **msoSyncConflictMerge** (2)

Use the **ResolveConflict** method to resolve differences between the local copy of the active document and the server copy. Use the **msoSyncConflictMerge** option (not available for a Microsoft Excel Workbook) to merge the changes from each document into the other. Replace the server copy with local changes by using the **msoSyncConflictClientWins** option, or replace the local copy with the changed server copy by using the **msoSyncConflictServerWins** option.

The **msoSyncConflictMerge** option merges changes made to the server copy into the local copy, but does not actually resolve the conflict. In order to resolve the conflict with the merged changes winning, you must save the active document after merging changes, then call the **ResolveConflict** method again with the **msoSyncConflictClientWins** option.

The **ResolveConflict** method can encounter a conflict condition if the client is unaware of recent changes to the server copy of the shared document. Call the **GetUpdate** method before calling **ResolveConflict** to refresh the status of the server copy and to detect a possible conflict.

The **ResolveConflict** method raises a run-time error if the local document has unsaved changes or if no conflict exists between the 2 copies of the document.

Not all document synchronization problems raise trappable run-time errors. After performing an operation using the **Sync** object, it's a good idea to check the **Status** property; if the **Status** property is **msoSyncStatusError**, check the **ErrorType** property for additional information on the type of error that has occurred.

The following example attempts to resolve a conflict by merging changes between the local and the server copies of the active document.

```
Dim objSync As Office.Sync
Dim strStatus As String
Set objSync = ActiveDocument.Sync
If objSync.Status = msoSyncStatusConflict Then
    objSync.ResolveConflict msoSyncConflictMerge
    ActiveDocument.Save
    objSync.ResolveConflict msoSyncConflictClientWins
    strStatus = "Conflict resolved by merging changes."
    MsgBox strStatus, vbInformation + vbOKOnly, "Sync Informatic
End If
Set objSync = Nothing
```

## **Restore Method**

Restores a previous saved version of a shared document from the **DocumentLibraryVersions** collection.

expression.Restore()

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

Use the **Restore** method to return to an earlier saved version of the active document. The **Restore** method does several things:

- 1. It changes the open version of the shared document to read-only mode but leaves it open.
- 2. It opens the restored version in read/write mode.
- 3. It saves the restored version to the server as a new document version, making the restored version the latest version.

The **Restore** method raises a run-time error if the active document has changes that have not been saved.

The following example restores the previous version of the active document.

```
Dim dlvVersions As Office.DocumentLibraryVersions
Set dlvVersions = ActiveDocument.DocumentLibraryVersions
dlvVersions(dlvVersions.Count - 1).Restore
Set dlvVersions = Nothing
```

# **Save Method**

Uploads changes made programmatically to a **SharedWorkspaceLink** or a **SharedWorkspaceTask** to the server.

expression.Save()

*expression* Required. An expression that returns a **SharedWorkspaceLink** or a **SharedWorkspaceTask** object.

Use the **Save** method to upload changes to the server after changing the properties of a shared workspace link or task.

The following example sets a new **DueDate** for each task in the shared workspace and uploads the changes to the server using the **Save** method.

```
Dim swsTask As Office.SharedWorkspaceTask
Const dtmNewDueDate As Date = #12/31/2005#
For Each swsTask In ActiveWorkbook.SharedWorkspace.Tasks
    swsTask.DueDate = dtmNewDueDate
    swsTask.Save
Next
Set swsTask = Nothing
```

# SaveCopyAs Method

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

expression.SaveCopyAs(Filename)

expression An HTMLProjectItem object.

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

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

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

# SetAvoidRectangle Method

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

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

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

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

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

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

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

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



### **SetFocus Method**

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

#### Remarks

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

#### expression.SetFocus

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

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

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

### SetSortOrder Method

Sets the sort order for mail merge data.

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

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

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

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

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

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

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

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

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

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



# **Show Method**

Show method as it applies to the **Balloon** object.

Displays the specified balloon object. Returns an **MsoBalloonButtonType** constant that indicates which button or label the user clicked in the balloon.

MsoBalloonButtonType can be one of these MsoBalloonButtonType constants. msoBalloonButtonAbort **msoBalloonButtonBack** msoBalloonButtonCancel msoBalloonButtonClose msoBalloonButtonIgnore msoBalloonButtonNext msoBalloonButtonNo msoBalloonButtonNull **msoBalloonButtonOK msoBalloonButtonOptions msoBalloonButtonRetry** msoBalloonButtonSearch msoBalloonButtonSnooze **msoBalloonButtonTips** msoBalloonButtonYes msoBalloonButtonYesToAll

#### expression.Show

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

Show method as it applies to the **FileDialog** object.

Displays a file dialog box and returns a **Long** indicating whether the user pressed the action button (-1) or the cancel button (0). When you call the **Show** 

method, no more code will execute until the user dismisses the file dialog box. In the case of Open and SaveAs dialog boxes, use the **Execute** method right after the **Show** method to carry out the user's action.

#### $expression. \\ \textbf{Show}$

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

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

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

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

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

```
Set balNew = Assistant.NewBalloon
With balNew
    .Heading = "Are you sure you want to set the " & _
        "printer orientation to Landscape?"
    .BalloonType = msoBalloonTypeButtons
    .Button = msoButtonSetYesNoCancel
End With
Select Case balNew.Show
    Case msoBalloonButtonCancel ' User selected Cancel button.
        returnValue = MsgBox("Operation canceled.", _
            vbOKOnly, "Printer Message")
Case msoBalloonButtonYes ' User selected Yes button.
        returnValue = MsgBox("Printer set to " & _
            "Landscape.", vbOKOnly, "Printer Message")
```

End Select

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

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

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

End Sub



# **ShowPopup Method**

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

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

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

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

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

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

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

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

### **StartWizard Method**

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

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

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

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

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

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

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

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

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

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

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

#### Remarks

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

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

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

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

# **Unsuspend Method**

Resumes synchronization between the local copy and the server copy of a shared document.

expression.Unsuspend()

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

### Remarks

Use the **Unsuspend** method to resume document synchronization when the **Status** property returns **msoSyncStatusSuspended**.

Not all document synchronization problems raise trappable run-time errors. After performing an operation using the **Sync** object, it's a good idea to check the **Status** property; if the **Status** property is **msoSyncStatusError**, check the **ErrorType** property for additional information on the type of error that has occurred.

The following example resumes document synchronization if it has been suspended.

```
Dim objSync As Office.Sync
Set objSync = ActiveDocument.Sync
If objSync.Status = msoSyncStatusSuspended Then
    objSync.Unsuspend
    MsgBox "Synchronization resumed.", _
        vbInformation + vbOKOnly, "Sync Status"
End If
Set objSync = Nothing
```

## **Update Method**

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

expression.Update

*expression* The **COMAddIns** collection.

### Remarks

Before you can use a given COM add-in in a Microsoft Office application, that add-in must be registered in the Windows registry as a COM component with a corresponding Component Category ID. Normally the setup program for a COM add-in will add the necessary entries to the registry.

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

Application.COMAddIns.Update

# **ActionControl Property**

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

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

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

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

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

### **ActiveMenuBar Property**

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

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

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

# AdaptiveMenu Property

**True** if an individual menu is enabled to use adaptive menus. Read/write **Boolean**.

**Note** This property will not function while the **AdaptiveMenus** property is set to **False**.

This property checks or unchecks the check box control for the option to show an individual menu as full or personalized. Only recently used submenus show up in a menu if this property is set to **True**. All submenus show up if this property is set to **False**.

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

CommandBars("File").AdaptiveMenu = False

**Note** This property works only when the specified command bar ("File" in this case) is the MSOBarTypePopup. A RunTime Error is returned when this property is used with another command bar type such as MSOBarTypeMenuBar.

# **AdaptiveMenus Property**

**True** if adaptive menus are enabled for all applications in Microsoft Office. Read/write **Boolean**.

This property checks or unchecks the check box control for the option to show menus in Microsoft Office as full or personalized. This control is set in any application by doing the following:

- 1. On the **Tools** menu, select **Customize**.
- 2. Select the **Options** tab.
- 3. The **Always show full menus** option is located in the **Personalized Menus and Toolbars** section.

This example sets three options for all command bars in Microsoft Office, including custom command bars and the controls on them.

```
With CommandBars
.LargeButtons = True
.DisplayFonts = True
.AdaptiveMenus = True
End With
```

## **AllowMultiSelect Property**

**True** if the user is allowed to select multiple files from a file dialog box. Read/write **Boolean**.

#### expression.AllowMultiSelect

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

### Remarks

This property has no effect on Folder Picker dialog boxes or SaveAs dialog boxes because users should never be able to select multiple files in these types of file dialog boxes.

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

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

End Sub



# **Animation Property**

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

Returns or sets an animation action for the Office Assistant. When this property is applied to the **Assistant** object and the Assistant supports the specified animation, the Assistant is animated immediately (if the Assistant is visible and enabled). When this property is applied to the **Balloon** object, the Assistant is animated only while the balloon is displayed. Read/write <u>MsoAnimationType</u>.

MsoAnimationType can be one of these MsoAnimationType constants.

**msoAnimationAppear msoAnimationBeginSpeaking** msoAnimationCharacterSuccessMajor msoAnimationCheckingSomething **msoAnimationDisappear msoAnimationEmptyTrash msoAnimationGestureDown msoAnimationGestureLeft msoAnimationGestureRight msoAnimationGestureUp msoAnimationGetArtsy** msoAnimationGetAttentionMajor msoAnimationGetAttentionMinor **msoAnimationGetTechy msoAnimationGetWizardy** msoAnimationGoodbye **msoAnimationGreeting msoAnimationIdle** msoAnimationListensToComputer **msoAnimationLookDown** msoAnimationLookDownLeft msoAnimationLookDownRight

msoAnimationLookLeft msoAnimationLookRight msoAnimationLookUp msoAnimationLookUpLeft msoAnimationLookUpRight msoAnimationPrinting msoAnimationRestPose msoAnimationSaving msoAnimationSearching msoAnimationSendingMail msoAnimationThinking 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 an obvious animation. However, all **MsoAnimationType** constants are valid for all Assistants. Note that different constants may produce the same animation.

The following **MsoAnimationType** constants represent animations that repeat the specified action until the Assistant is dismissed or until the **Animation** property is reset with another animation:

msoAnimationCheckingSomething	
	msoAnimationThinking
msoAnimationGetTechy	
	msoAnimationWorkingAtSomething
msoAnimationListensToComputer	
	msoAnimationWritingNotingSomething
msoAnimationSearching	

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

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

# **Application Property**

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

#### expression. Application

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

This example returns the name of the application in which the command bar named Standard was created and displays this result in a message box.

Set Appobj = CommandBars("Standard").Application
MsgBox Appobj

# **AssignedTo Property**

Returns or sets the **SharedWorkspaceMember** to whom the task is assigned. Read/write **String**.

expression.AssignedTo()

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

### Remarks

The optional **AssignedTo** property of a shared workspace task returns the display name or **Name** property of the **SharedWorkspaceMember** to whom the task is assigned. However, use the **DomainName** property of a **SharedWorkspaceMember** to set this property.

The following example uses the **AssignedTo** property to reassign all shared workspace tasks currently assigned to one workspace member to a different member, using the member's **Name** when reading the property and **DomainName** when writing a new value for it.

```
Dim swsTask As Office.SharedWorkspaceTask
For Each swsTask In ActiveWorkbook.SharedWorkspace.Tasks
    Debug.Print swsTask.AssignedTo
    If swsTask.AssignedTo = "New Employee" Then
        swsTask.AssignedTo = "MYCOMPANY\user123"
    End If
Next
Set swsTask = Nothing
```

# **AssistWithAlerts Property**

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

**True** if the Office Assistant balloon delivers application alerts when the Office Assistant is visible. Read/write **Boolean**.

#### Remarks

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

If this property is set to **False**, the application displays alerts in dialog boxes.

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

```
With Assistant
    .On = True
    .Visible = True
    .AssistWithHelp = True
    .Animation = msoAnimationGetAttentionMajor
End With
```

# AssistWithHelp Property

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

**True** if the Office Assistant appears when the user presses the F1 key to display Help. Read/write **Boolean**.

**Note** This property is obsolete but has been retained for compatibility with code written in previous versions of Microsoft Office.

#### Remarks

The **AssistWithHelp** property corresponds to the **Respond to F1 key** option under **Use the Office Assistant** on the **Options** tab in the **Office Assistant** dialog box.

If this property is set to **False**, the **Help Topics** dialog box appears instead of the Office Assistant.

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

```
With Assistant
    .On = True
    .Visible = True
    .AssistWithHelp = True
    .AssistWithAlerts = True
    .Animation = msoAnimationGetAttentionMajor
End With
```

# **AssistWithWizards Property**

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

**True** if the Office Assistant provides online Help with wizards. Read/write **Boolean**.

### Remarks

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

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

Assistant.**AssistWithWizards** = True

# **AttachCertificate Property**

**True** if the digital certificate that corresponds to the specified **Signature** object is attached to the document. Read/write **Boolean**.

#### expression.AttachCertificate

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

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

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



## **BalloonError Property**

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

MsoBalloonErrorType can be one of these MsoBalloonErrorType constants. **msoBalloonErrorBadCharacter** The balloon contains an ASCII control character other than CR or LF and less than 32.

**msoBalloonErrorBadPictureRef** The balloon contains a graphic that couldn't be displayed because the file doesn't exist or because the graphic isn't a valid .BMP or .WMF file.

**msoBalloonErrorBadReference** The balloon contains an unrecognized or unsupported reference.

**msoBalloonErrorButtonlessModal** The balloon you attempted to display is modal, but it contains no buttons. The balloon won't be shown because it can't be dismissed.

**msoBalloonErrorButtonModeless** The balloon you attempted to display is modeless, contains buttons, and has no procedure assigned to the **Callback** property. The balloon won't be shown because a callback procedure is required for modeless balloons.

**msoBalloonErrorCharNotTopmostForModal** The modal balloon was requested by an application that isn't the active application. Microsoft Office renders balloons for the active (topmost) application only.

**msoBalloonErrorCOMFailure** The balloon could not be displayed because of a COM failure.

msoBalloonErrorNone No error was encountered.

**msoBalloonErrorOther** The balloon won't appear because some other error occurred, such as another modal balloon is already active.

**msoBalloonErrorOutOfMemory** The balloon won't appear because there is insufficient memory.

**msoBalloonErrorTooBig** The balloon is too big to appear on the screen.

**msoBalloonErrorTooManyControls** The balloon contains more than twenty controls (check boxes or labels).

This example creates a balloon that generates an error. The error is generated because the balloon is created without a way to dismiss it: the button type is set to **msoButtonSetNone** and the default balloon mode is **msoModeModal**, resulting in a buttonless, modal balloon. Note that there's no way to dismiss a buttonless modal balloon.

```
With Application.Assistant
With .NewBalloon
    .Heading = "This will never show."
    .Text = "Imagine a balloon here."
    .Button = msoButtonSetNone
    .Show
End With
.Visible = True
If .BalloonError = msoBalloonErrorButtonlessModal Then
    MsgBox "You need a button to dismiss the balloon."
End If
End With
```



## **BalloonType Property**

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

Returns or sets the type of balloon the Office Assistant uses. When you create a **Balloon** object, this property is initially set to **msoBalloonTypeButtons**. Read/write **MsoBalloonType**.

MsoBalloonType can be one of these MsoBalloonType constants. msoBalloonTypeBullets msoBalloonTypeButtons msoBalloonTypeNumbers

This example creates an instruction balloon that explains how to select a printer. The balloon is modeless, so the user can follow the instructions in the balloon and keep the balloon visible as he or she works.

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



# **BeginGroup Property**

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

This example begins a new group with the last control on the active menu bar.

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



## **BuiltIn Property**

**True** if the specified <u>command bar</u> or <u>command bar control</u> is a built-in command bar or control of the container application. **False** if it's a custom command bar or control, or if it's a built-in control whose **OnAction** property has been set. Read-only **Boolean**.

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

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



## **BuiltInFace Property**

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

This example determines whether the face of the first control on the command bar named "Custom" is its built-in button face. If it is, the example copies the button face to the Clipboard.



## **Button Property**

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

Returns or sets the type of button displayed at the bottom of the Office Assistant balloon. When you create a **Balloon** object, this property is initially set to **msoButtonSetOK**. Read/write **MsoButtonSetType**.

MsoButtonSetType can be one of these MsoButtonSetType constants.

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

This example displays a balloon that contains a heading, text, three region choices, and two command buttons (**OK** and **Cancel**).

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

## **ButtonName Property**

Sets or returns a **String** representing the text that is displayed on the action button of a file dialog box. By default, this property is set to the standard text for the type of file dialog box. For example, in the case of the Open dialog box, the property is set to "Open" by default. This string is limited to fifty-one characters. Read/write.

#### expression.ButtonName

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

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

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

End Sub

# **Callback Property**

Sets the name of the procedure to run from a modeless balloon. Read/write **String**.

### Remarks

The procedure you specify for the **Callback** property must be written to receive either two or three arguments, depending on what you use the property with. If you use the **Callback** property with a wizard, you must write the procedure to receive two arguments: a long integer that represents the **msoBalloonButtonType** value of the button that the user clicked, and a long integer that uniquely identifies the balloon. If you use the **Callback** property with a modeless balloon, you must write the procedure to receive three arguments: the **Balloon** object that called the procedure; a long integer that represents the **msoBalloonButtonType** value of the button the user clicked; and a long integer that uniquely identifies the balloon that called the procedure, as denoted in the balloon's **Private** property.

The callback procedure must contain at least one condition under which the **Close** method is applied to the **Balloon** object that is passed to it; otherwise, the modeless balloon cannot be dismissed.

If you specify a procedure that is stored in a separate class module, you must include the module name in the value assigned to the **Callback** property (for example, "Sheet1.MyCallback).

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

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



# **Caption Property**

Returns or sets the caption text for a <u>command bar control</u>. Read/write **String**.

**Note** A control's caption is also displayed as its default ScreenTip.

This example adds a command bar control with a spelling checker button face to a custom command bar, and then it sets the caption to "Spelling checker."

```
Set myBar = CommandBars.Add(Name:="Custom", _
Position:=msoBarTop, Temporary:=True)
myBar.Visible = True
Set myControl = myBar.Controls _
.Add(Type:=msoControlButton, Id:=2)
With myControl
    .DescriptionText = "Starts the spelling checker"
    .Caption = "Spelling checker"
End With
```

# **Checkboxes Property**

Returns the **BalloonCheckboxes** collection that represents all the check boxes contained in the specified balloon. Read-only.

For information about returning a single member of a collection, see <u>Returning</u> <u>an Object from a Collection</u>.

This example creates a balloon with a heading, text, and three region choices. When the user clicks **OK** in the balloon, data for the selected region or regions is printed.

```
With Assistant.NewBalloon
    .Heading = "Regional Sales Data"
    .Text = "Select the region(s) you want to print."
    For i = 1 To 3
        .CheckBoxes(i).Text = "Region " & i
    Next
    .Button = msoButtonSetOkCancel
    If .Show = msoBalloonButtonOK Then
        dataPrinted = 0
        For i = 1 To 3
            If .CheckBoxes(i).Checked = True Then
                ' Code to print region data.
                dataPrinted = dataPrinted + 1
                MsgBox "Region " & i & " data printed."
            End If
        Next
        If dataPrinted = 0 Then MsgBox "No data printed."
    End If
End With
```

# **Checked Property**

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

**True** if the specified check box in the Office Assistant balloon is checked. Read/write **Boolean**.

This example creates a balloon with a heading, text, and three region choices. When the user clicks **OK** in the balloon, data for the selected region or regions is printed.

```
With Assistant.NewBalloon
    .Heading = "Regional Sales Data"
    .Text = "Select the region(s) you want to print."
    For i = 1 To 3
        .CheckBoxes(i).Text = "Region " & i
    Next
    .Button = msoButtonSetOkCancel
    If .Show = msoBalloonButtonOK Then
        dataPrinted = 0
        For i = 1 To 3
            If .CheckBoxes(i).Checked = True Then
                ' Code to print region data.
                dataPrinted = dataPrinted + 1
                MsgBox "Region " & i & " data printed."
            End If
        Next
        If dataPrinted = 0 Then MsgBox "No data printed."
    End If
End With
```

# **Column Property**

Returns or sets a **String** that represents the name of the field in the mail merge data source to use in the filter. Read/write.

#### expression.Column

*expression* Required. An expression that returns one of the objects in the Applies To list.

The following example changes an existing filter to remove from the mail merge all records that do not have a Region field equal to "WA".

```
Sub SetQueryCriterion()
    Dim appOffice As Office.OfficeDataSourceObject
    Dim intItem As Integer
    Set appOffice = Application.OfficeDataSourceObject
    appOffice.Open bstrConnect:="DRIVER=SQL Server; SERVER=ServerName
        "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
   With appOffice.Filters
        For intItem = 1 To .Count
            With .Item(intItem)
                If .Column = "Region" Then
                    .Comparison = msoFilterComparisonNotEqual
                    .CompareTo = "WA"
                    If .Conjunction = "Or" Then .Conjunction = "And"
                End If
            End With
        Next intItem
    End With
End Sub
```

# **Columns Property**

Returns an **ODSOColumns** object that represents the fields in a data source.

#### expression.Columns

*expression* Required. An expression that returns one of the objects in the Applies To list.

The following example displays the field names in the data source attached to the active publication.

```
Sub ShowFieldNames()
   Dim appOffice As OfficeDataSourceObject
   Dim intCount As Integer
   Set appOffice = Application.OfficeDataSourceObject
   appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName
      "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
   With appOffice.Columns
      For intCount = 1 To .Count
           MsgBox "Field Name: " & .Item(intCount).Name
           Next
   End With
End Sub
```



## **CommandBar Property**

Returns a **CommandBar** object that represents the menu displayed by the specified pop-up control. Read-only.

This example sets the variable fourthLevel to the fourth control on the command bar named "Drawing."

```
Set fourthLevel = CommandBars("Drawing") _
.Controls(1).CommandBar.Controls(4)
```

## **CommandBars Property**

Returns a **<u>CommandBars</u>** collection.

#### expression.CommandBars

*expression* Required. An expression that returns one of the objects in the Applies To list.

The following example return the **CommandBars** collection from the **MsoEnvelope** object in Microsoft Word.

Dim cbars As CommandBars
Set cbars = Application.ActiveDocument.MailEnvelope.Commandbars

## **Comments Property**

Returns any optional comments associated with the specified version of the shared document. Read-only **String**.

expression.Comments()

*expression* Required. An expression that returns a **DocumentLibraryVersion** object.

### Remarks

A user can attach version comments through the document library user interface when checking in a document that was previously checked out.

The following example lists comments and other properties for each version of a shared document.

```
Dim dlvVersions As Office.DocumentLibraryVersions
Dim dlvVersion As Office.DocumentLibraryVersion
Dim strVersionInfo As String
Set dlvVersions = ActiveDocument.DocumentLibraryVersions
If dlvVersions.IsVersioningEnabled Then
    strVersionInfo = "This document has " & _
        dlvVersions.Count & " versions: " & vbCrLf
    For Each dlvVersion In dlvVersions
        strVersionInfo = strVersionInfo &
             " - Version #: " & dlvVersion.Index & vbCrLf & _

    Modified by: " & dlvVersion.ModifiedBy & vbCrLf
    Modified on: " & dlvVersion.Modified & vbCrLf &

             ш
             н
               - Comments: " & dlvVersion.Comments & vbCrLf
             п
    Next
Else
    strVersionInfo = "Versioning not enabled for this document."
End If
MsgBox strVersionInfo, vbInformation + vbOKOnly, "Version Inform
Set dlvVersion = Nothing
Set dlvVersions = Nothing
```

# **CompareTo Property**

Returns or sets a **String** that represents the text to compare in the query filter criterion. Read/write.

#### expression.CompareTo

*expression* Required. An expression that returns one of the objects in the Applies To list.

The following example changes an existing filter to remove from the mail merge all records that do not have a Region field equal to "WA".

```
Sub SetQueryCriterion()
    Dim appOffice As Office.OfficeDataSourceObject
    Dim intItem As Integer
    Set appOffice = Application.OfficeDataSourceObject
    appOffice.Open bstrConnect:="DRIVER=SQL Server; SERVER=ServerName
        "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
   With appOffice.Filters
        For intItem = 1 To .Count
            With .Item(intItem)
                If .Column = "Region" Then
                    .Comparison = msoFilterComparisonNotEqual
                    .CompareTo = "WA"
                    If .Conjunction = "Or" Then .Conjunction = "And"
                End If
            End With
        Next intItem
    End With
End Sub
```



## **Comparison Property**

Returns or sets an **MsoFilterComparison** constant that represents how to compare the **Column** and **CompareTo** properties. Read/write.

MsoFilterComparison can be one of these MsoFilterComparison constants. msoFilterComparisonEqual msoFilterComparisonGreaterThan msoFilterComparisonGreaterThanEqual msoFilterComparisonIsBlank msoFilterComparisonIsNotBlank msoFilterComparisonLessThan msoFilterComparisonLessThanEqual msoFilterComparisonNotContains msoFilterComparisonNotEqual

#### expression.Comparison

*expression* Required. An expression that returns one of the objects in the Applies To list.

The following example changes an existing filter to remove from the mail merge all records that do not have a Region field equal to "WA".

```
Sub SetQueryCriterion()
    Dim appOffice As Office.OfficeDataSourceObject
    Dim intItem As Integer
    Set appOffice = Application.OfficeDataSourceObject
    appOffice.Open bstrConnect:="DRIVER=SQL Server; SERVER=ServerName
        "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
   With appOffice.Filters
        For intItem = 1 To .Count
            With .Item(intItem)
                If .Column = "Region" Then
                    .Comparison = msoFilterComparisonNotEqual
                    .CompareTo = "WA"
                    If .Conjunction = "Or" Then .Conjunction = "And"
                End If
            End With
        Next intItem
    End With
End Sub
```



# **Condition Property**

Some of the content in this topic may not be applicable to some languages.

Returns the condition of the specified search criteria. Read-only MsoCondition.

MsoCondition can be one of these MsoCondition constants. msoConditionAnyNumberBetween **msoConditionAnytime msoConditionAnytimeBetween msoConditionAtLeast** msoConditionAtMost **msoConditionBeginsWith** msoConditionDoesNotEqual **msoConditionEndsWith msoConditionEquals** msoConditionFileTypeAllFiles **msoConditionFileTypeBinders** msoConditionFileTypeCalendarItem msoConditionFileTypeContactItem msoConditionFileTypeDatabases msoConditionFileTypeDataConnectionFiles msoConditionFileTypeDesignerFiles msoConditionFileTypeEPaperFiles msoConditionFileTypeExcelWorkbooks msoConditionFileTypeJournalItem msoConditionFileTypeMailItem msoConditionFileTypeNoteItem msoConditionFileTypeOfficeFiles msoConditionFileTypeOutlookItems msoConditionFileTypePhotoDrawFiles msoConditionFileTypePowerPointPresentations

msoConditionFileTypeProjectFiles msoConditionFileTypePublisherFiles msoConditionFileTypeTaskItem msoConditionFileTypeTemplates msoConditionFileTypeVisioDocuments msoConditionFileTypeWebPages msoConditionFileTypeWordDocuments msoConditionFreeText msoConditionIncludes msoConditionIncludesFormsOf msoConditionIncludesNearEachOther msoConditionIncludesPhrase msoConditionInTheLast msoConditionInTheNext **msoConditionIsExactly** msoConditionIsNo msoConditionIsNot **msoConditionIsYes** msoConditionLastMonth msoConditionLastWeek msoConditionLessThan msoConditionMoreThan msoConditionNextMonth msoConditionNextWeek msoConditionOn msoConditionOnOrAfter msoConditionOnOrBefore **msoConditionThisMonth msoConditionThisWeek msoConditionToday msoConditionTomorrow msoConditionYesterday** 

This example returns the condition value for search criteria for the first property test.

```
With Application.FileSearch.PropertyTests(1)
MsgBox "The condition you've set is: " & .Condition
End With
```



# **Conjunction Property**

Returns or sets an **MsoFilterConjunction** constant that represents how a filter criterion relates to other filter criteria in the **ODSOFilters** object. Read/write.

MsoFilterConjunction can be one of these MsoFilterConjunction constants. **msoFilterConjunctionAnd msoFilterConjunctionOr** 

expression.Conjunction

*expression* Required. An expression that returns one of the objects in the Applies To list.

The following example changes an existing filter to remove from the mail merge all records that do not have a Region field equal to "WA".

```
Sub SetQueryCriterion()
    Dim appOffice As Office.OfficeDataSourceObject
    Dim intItem As Integer
    Set appOffice = Application.OfficeDataSourceObject
    appOffice.Open bstrConnect:="DRIVER=SQL Server; SERVER=ServerName
        "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
   With appOffice.Filters
        For intItem = 1 To .Count
            With .Item(intItem)
                If .Column = "Region" Then
                    .Comparison = msoFilterComparisonNotEqual
                    .CompareTo = "WA"
                    If .Conjunction = "Or" Then .Conjunction = "And"
                End If
            End With
        Next intItem
    End With
End Sub
```

# **Connect Property**

Returns or sets the state of the connection for the specified **COMAddIn** object. Read/write **Boolean**.

The **Connect** property returns **True** if the add-in is active; it returns **False** if the add-in is inactive. An active add-in is registered and connected; an inactive add-in is registered but not currently connected.

The following example displays a message box that indicates whether COM add-in one is registered and currently connected.

If Application.COMAddIns(1).Connect Then
 MsgBox "The add-in is connected."
Else
 MsgBox "The add-in is not connected."
End If

# **Connected Property**

Returns a **Boolean** value that indicates whether or not the active document is currently saved in and connected to a shared workspace. Read-only **Boolean**.

expression.Connected

*expression* Required. An expression that returns a **SharedWorkspace** object.

Use the **Disconnect** method of the **SharedWorkspace** object to disconnect the local copy of the active document from the shared workspace. Use the **RemoveDocument** method to remove the document from the shared workspace.

The following example checks the **Connected** property to determine whether the active document is already saved in a shared workspace.

If ActiveWorkbook.SharedWorkspace.Connected Then
 MsgBox "This document is already saved in a shared workspace
End If



# **Connector Property**

Returns the connector between two similar property test values. The default value is **msoConnectorAnd**. Read-only **MsoConnector**.

MsoConnector can be one of these MsoConnector constants. msoConnectorAnd msoConnectorOr

A connector specifies whether two similar search criteria will be combined to form one property test (as with **msoConnectorAnd**) or treated independently (as with **msoConnectorOr**).

This example displays a message that describes how the search criteria will be evaluated in a file search.

```
With Application.FileSearch.PropertyTests(1)
If .Connector = msoConnectorAnd Then
    MsgBox "All search criteria will be combined."
Else
    MsgBox "Criteria will be treated independently"
End If
End With
```

# **ConnectString Property**

Returns or sets a **String** that represents the connection to the specified mail merge data source. Read/write.

#### expression.ConnectString

*expression* Required. An expression that returns one of the objects in the Applies To list.

This example checks if the connection string contains the characters ODSOOutlook and displays a message accordingly.

```
Sub VerifyCorrectDataSource()
   Dim appOffice As OfficeDataSourceObject
   Set appOffice = Application.OfficeDataSourceObject
   appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName
      "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
   If InStr(appOffice.ConnectString, "ODSOOutlook") > 0 Then
      MsgBox "Your Outlook address book is used as the data source
   Else
      MsgBox "Your Outlook address book is not used as the data sc
End If
```

End Sub



# **Context Property**

Returns or sets a string that determines where a <u>command bar</u> will be saved. The string is defined and interpreted by the application. Read/write **String**.

You can set the **Context** property only for custom command bars. This property will fail if the application doesn't recognize the context string, or if the application doesn't support changing context strings programmatically.

This example displays a message box containing the context string for the command bar named "Custom". This example works in Microsoft Word and other applications that support the **Context** property.

```
Set myBar = CommandBars _
.Add(Name:="Custom", Position:=msoBarTop, _
Temporary:=True)
With myBar
.Controls.Add Type:=msoControlButton, ID:=2
.Visible = True
End With
MsgBox (myBar.Context)
```

# **Controls Property**

Returns a <u>**CommandBarControls</u>** object that represents all the controls on a command bar or pop-up control. Read-only.</u>

For information about returning a single member of a collection, see **<u>Returning</u>** <u>**an Object from a Collection**</u>.

This example adds a combo box control to the command bar named "Custom" and fills the list with two items. The example also sets the number of line items, the width of the combo box, and an empty default for the combo box.

```
Set myControl = CommandBars("Custom").Controls _
.Add(Type:=msoControlComboBox, Before:=1)
With myControl
.AddItem Text:="First Item", Index:=1
.AddItem Text:="Second Item", Index:=2
.DropDownLines = 3
.DropDownWidth = 75
.ListHeaderCount = 0
End With
```

# **Count Property**

Some of the content in this topic may not be applicable to some languages.

Returns or sets a **Long** indicating the number of items in the specified collection. Read/write **Long** for the **BalloonCheckboxes** and **BalloonLabels** objects; readonly **Long** for all other objects in the Applies To list.

#### expression.Count

*expression* Required. An expression that returns one of the above objects.

For the **CommandBars** collection, the count includes only menu bars, toolbars, and shortcut menus.

For the **Permission** object, the **Count** property returns 0 (zero) if permissions are not enabled on the active document. When permissions are enabled, the **Count** property always returns at least 1, representing the permissions of the document author.

For the **Scripts** collection, the count returned is the number of script blocks in the specified document. In Microsoft Word, Scripts.Count returns the total number of inline and floating script anchors combined.

This example uses the **Count** property to display the number of command bars in the **CommandBars** collection.

This example uses the **Count** property to display the number of check boxes in the Office Assistant balloon.

```
With Assistant.NewBalloon
    .CheckBoxes(1).Text = "First Choice"
    .CheckBoxes(2).Text = "Second Choice"
    .Text = "You have the following " _
    & .CheckBoxes.Count & " choices."
    .Show
End With
```

This example displays the number of custom document properties in the active document.

```
MsgBox ("There are " & _
ActiveDocument.CustomDocumentProperties.Count & _
" custom document properties in the " & _
"active document.")
```

# **CreatedBy Property**

Returns the friendly name of the member who created the shared workspace object. Read-only **String**.

expression.CreatedBy

*expression* Required. An expression that returns one of the objects in the Applies To list.

The **CreatedBy** property returns the friendly name stored in the **Name** property of the **DWSMember** object.

The **SharedWorkspaceFolder** and **SharedWorkspaceMember** objects do not have a **CreatedBy** property.

The following example lists files in the shared workspace that were created by users other than the creator of the workspace.

```
Dim swsFile As Office.SharedWorkspaceFile
Dim swsOwner As Office.SharedWorkspaceMember
Dim strMemberFiles As String
Set swsOwner = ActiveWorkbook.SharedWorkspace.Members(1)
For Each swsFile In ActiveWorkbook.SharedWorkspace.Files
If swsFile.CreatedBy <> swsOwner.Name Then
strMemberFiles = strMemberFiles & swsFile.URL & vbCrLf
End If
Next
MsgBox "These files were created by other users:" & ______
vbCrLf & strMemberFiles, _______
vbInformation + vbOKOnly, "Files Created by Other Users"
Set swsOwner = Nothing
Set swsFile = Nothing
```

# **CreatedDate Property**

Returns the date and time when the shared workspace object was created. Readonly **Variant**.

#### expression.CreatedDate

*expression* Required. An expression that returns one of the objects in the Applies To list.

The **SharedWorkspaceFolder** and **SharedWorkspaceMember** objects do not have a **CreatedDate** property.

The following example returns a list of shared workspace files whose date and time created is earlier than today.

```
Dim swsFile As Office.SharedWorkspaceFile
Dim dtmMidnight As Date
Dim dtmFileDate As Date
Dim strOlderFiles As String
dtmMidnight = CDate(FormatDateTime(Now, vbShortDate) & " 12:00:6
For Each swsFile In ActiveWorkbook.SharedWorkspace.Files
    dtmFileDate = swsFile.CreatedDate
    If dtmFileDate < dtmMidnight Then
       strOlderFiles = strOlderFiles & swsFile.URL & vbCrLf
    End If
Next
MsgBox "Files older than today: " & vbCrLf & strOlderFiles, _
    vbInformation + vbOKOnly, "Older Files"
Set swsFile = Nothing
```

# **Creator Property**

Returns a 32-bit integer that indicates the application in which the specified object was created. For example, if the object was created in Microsoft Word, this property returns 1297307460, which represents the string "MSWD"; in Microsoft Excel, this property returns 1480803660. This value can also be represented by the constant **wdCreatorCode** in Word, or **xlCreatorCode** in Excel. Read-only **Long**.

#### expression.Creator

*expression* Required. An expression that returns one of the above objects.

The **Creator** property was primarily designed to be used on the Macintosh, where each application has a four-character creator code. For example, Microsoft Word has the creator code MSWD. For additional information about this property, consult the language reference Help included with Microsoft Office Macintosh Edition.

The **Creator** property always returns the numeric identifier for the active application, just as the **Application** property always returns the name of the active applicatin in string form. Used the **CreatedBy** property of the **SharedWorkspaceFile**, **SharedWorkspaceFolder**, **SharedWorkspaceLink**, and **SharedWorkspaceTask** objects to return the name of the individual who created those objects. Use document properties to return information about the authors of Office documents.

This example displays a message about the creator of myObject.

Set myObject = ActiveDocument
If myObject.Creator = wdCreatorCode Then
 MsgBox "This is a Microsoft Word object"
Else
 MsgBox "This is not a Microsoft Word object"
End If

### **DataSource Property**

Returns or sets a **String** that represents the name of the attached data source. Read/write.

#### expression.DataSource

*expression* Required. An expression that returns one of the objects in the Applies To list.

The following example sets the name of the data source if the name is blank.

```
Sub SetAndReturnDataSourceName()
    Dim appOffice As OfficeDataSourceObject
    Set appOffice = Application.OfficeDataSourceObject
    With appOffice
    .Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName;" &
        "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employee
        If .DataSource = "" Then
        .DataSource = "Northwind"
        MsgBox .DataSource
        End If
        End With
End Sub
```



## **Description Property**

Description property as it applies to the **COMAddIn** object.

Returns or sets a descriptive **String** value for the specified **COMAddin** object. Read/write.

#### expression. Description

*expression* Required. An expression that returns a **COMAddin** object.

Description property as it applies to the **FileDialogFilter** object.

Returns the description of each **Filter** object as a **String** value. The description is the text that is displayed in the file dialog box. Read-only.

#### expression. **Description**

*expression* Required. An expression that returns a **FileDialogFilter** object.

Description property as it applies to the **SharedWorkspaceLink** and **SharedWorkspaceTask** objects.

Returns or sets a descriptive **String** value for the specified **SharedWorkspaceLink** or **SharedWorkspaceTask** object. Read/write.

#### *expression*.**Description**

*expression* Required. An expression that returns a **SharedWorkspaceLink** or **SharedWorkspaceTask** object.

### Remarks

The **Description** property is optional when a new shared workspace link or task is created, and may be empty.

As it applies to the **COMAddIn** object.

The following example displays the description text of the Microsoft Accessibility COM add-in for drawing.

```
MsgBox "The description of this " & _
    "COMAddIn is """ & Application.COMAddIns. _
    Item("msodraa9.ShapeSelect"). _
    Description & """
```

As it applies to the **FileDialogFilter** object.

The following example iterates through the default filters of the SaveAs dialog box and displays the description of each filter that includes a Microsoft Excel file. The **Extensions** property is used to find the appropriate filter objects.

```
Sub Main()
    'Declare a variable as a FileDialogFilters collection.
    Dim fdfs As FileDialogFilters
    'Declare a variable as a FileDialogFilter object.
    Dim fdf As FileDialogFilter
    'Set the FileDialogFilters collection variable to
    'the FileDialogFilters collection of the SaveAs dialog box.
    Set fdfs = Application.FileDialog(msoFileDialogSaveAs).Filters
    'Iterate through the description and extensions of each
    'default filter in the SaveAs dialog box.
    For Each fdf In fdfs
        'Display the description of filters that include
        'Microsoft Excel files.
        If InStr(1, fdf.Extensions, "xls", vbTextCompare) > 0 Then
            MsgBox "Filter description: " & fdf. Description
        End If
    Next fdf
```

End Sub



## **DescriptionText Property**

Returns or sets the description for a <u>command bar control</u>. The description is not displayed to the user, but it can be useful for documenting the behavior of the control for other developers. Read/write **String**.

### Remarks

This property is used for Balloon Help on the Macintosh.

This example adds a control to a custom command bar, including a description of the control's behavior.

```
Set myBar = CommandBars.Add("Custom", msoBarTop, , True)
myBar.Visible = True
Set myControl = myBar.Controls _
    .Add(Type:=msoControlButton, ID:= _
    CommandBars("Standard").Controls("Paste").ID)
With myControl
    .DescriptionText = "Pastes the contents of the Clipboard"
    .Caption = "Paste"
End With
```



## **DialogType Property**

Returns an **MsoFileDialogType** constant representing the type of file dialog box that the **FileDialog** object is set to display. Read-only.

MsoFileDialogType can be one of these MsoFileDialogType constants. msoFileDialogFilePicker msoFileDialogFolderPicker msoFileDialogOpen msoFileDialogSaveAs

expression.DialogType

*expression* Required. An expression that returns one of the objects in the Applies To list.

The following example takes a **FileDialog** object of an unknown type and runs the **Execute** method if it is a SaveAs dialog box or an Open dialog box.

Sub DisplayAndExecuteFileDialog(ByRef fd As FileDialog)

```
'Use a With...End With block to reference the FileDialog object.
With fd
  'If the user presses the action button...
  If .Show = -1 Then
    'Use the DialogType property to determine whether to
    'use the Execute method.
    Select Case .DialogType
        Case msoFileDialogOpen, msoFileDialogSaveAs: .Execut
        'Do nothing otherwise.
        Case Else
        End Select
    'If the user presses Cancel...
    Else
    End If
End With
```

End Sub

# DisableAskAQuestionDropdown Property

**True** if the Answer Wizard dropdown menu is enabled. Read/write **Boolean**.

#### expression.DisableAskAQuestionDropdown

*expression* Required. An expression that returns one of the objects in the Applies To list.

The following example toggles the **DisableAskAQuestionDropdown** property.

```
Sub ToggleQuestionDropdown()
With Application.CommandBars
If .DisableAskAQuestionDropdown = True Then
.DisableAskAQuestionDropdown = False
Else
.DisableAskAQuestionDropdown = True
End If
End With
End Sub
```

## **DisableCustomize Property**

**True** if toolbar customization is disabled. Read/write **Boolean**.

#### expression. DisableCustomize

*expression* Required. An expression that returns one of the objects in the Applies To list.

The following example toggles the **DisableCustomize** property.

```
Sub ToggleCustomize()
With Application.CommandBars
If .DisableCustomize = True Then
.DisableCustomize = False
Else
.DisableCustomize = True
End If
End With
End Sub
```

## **DisplayFonts Property**

**True** if the font names in the **Font** box are displayed in their actual fonts. Read/write **Boolean**.

This example sets three options for all command bars in Microsoft Office, including custom command bars and the controls on them.

```
With CommandBars
.LargeButtons = True
.DisplayFonts = True
.AdaptiveMenus = True
End With
```



# **DisplayKeysInTooltips Property**

**True** if shortcut keys are displayed in the ToolTips for each <u>command bar</u> <u>control</u>. Read/write **Boolean**.

### Remarks

To display shortcut keys in ToolTips, you must also set the **<u>DisplayTooltips</u>** property to **True**.

This example sets options for all command bars in Microsoft Office.

```
With CommandBars
    .LargeButtons = True
    .DisplayTooltips = True
    .DisplayKeysInTooltips = True
    .MenuAnimationStyle = msoMenuAnimationUnfold
End With
```



## **DisplayTooltips Property**

**True** if ScreenTips are displayed whenever the user positions the pointer over <u>command bar controls</u>. Read/write **Boolean**.

### Remarks

Setting the **DisplayTooltips** property in a container application immediately affects every command bar in every running Microsoft Office application, and in every Office application opened after the property is set.

This example displays large controls and ToolTips on all command bars.

Set allBars = CommandBars

allBars.LargeButtons = True
allBars.DisplayTooltips = True

### **DocumentAuthor Property**

Returns or sets the name in email form of the author of the active document. Read/write **String**.

#### expression.DocumentAuthor

*expression* Required. An expression that returns a **Permission** object.

### Remarks

The **DocumentAuthor** property returns or sets the author of the active document. The author always has non-expiring owner rights to the document, whether owner permission is granted explicitly (through a **UserPermission** object) or not.

The **DocumentAuthor** property can only be changed to a different account that has been certified through the permissions user interface to open restricted content on the local machine. In most cases, users who have a single Windows account can only choose between their Windows and their Passport identities.

If the user's Microsoft Windows and Passport identities use the same email address, then use the format passport:someone@example.com to specify the Passport identity as the **DocumentAuthor** property.

The following example displays information about the permissions settings of the active document, including the document author.

```
Dim irmPermission As Office.Permission
Dim strIRMInfo As String
Set irmPermission = ActiveWorkbook.Permission
If irmPermission. Enabled Then
    strIRMInfo = "Permissions are enabled on this document." & v
    strIRMInfo = strIRMInfo & " View in trusted browser: " &
        irmPermission.EnableTrustedBrowser & vbCrLf &
        " Document author: " & irmPermission.DocumentAuthor & vb
" Users with rights: " & irmPermission.Count & vbCrLf &
        " Cache licenses locally: " & irmPermission.StoreLicense
        " Request permission URL: " & irmPermission.RequestPermi
    If irmPermission.PermissionFromPolicy Then
        strIRMInfo = strIRMInfo & " Permissions applied from pol
           Policy name: " & irmPermission.PolicyName & vbCrLf &
        " Policy description: " & irmPermission.PolicyDescripti
    Else
        strIRMInfo = strIRMInfo & " Default permissions applied.
    End If
Else
    strIRMInfo = "Permissions are NOT enabled on this document."
End If
MsgBox strIRMInfo, vbInformation + vbOKOnly, "IRM Information"
Set irmPermission = Nothing
```

### **DomainName Property**

Returns the domain and user name of the specified **SharedWorkspaceMember** in the format domain\user. Read-only **String**.

#### expression.DomainName

*expression* Required. An expression that returns a **SharedWorkspaceMember** object.

The following example extracts the domain name from the **DomainName** property of each shared workspace member and lists members who belong to the "MyCompany" domain.

```
Dim swsMember As Office.SharedWorkspaceMember
Dim strDomain As String
Dim strMemberList As String
For Each swsMember In ActiveWorkbook.SharedWorkspace.Members
    strDomain = UCase(Left(swsMember.DomainName, _
        InStr(swsMember.DomainName, "\") - 1))
    If strDomain = "MYCOMPANY" Then
        strMemberList = strMemberList & swsMember.Name & vbCrLf
    End If
Next
MsgBox strMemberList, vbInformation + vbOKOnly, _
    "Members in the MYCOMPANY Domain"
Set swsMember = Nothing
```



### **DropDownLines Property**

Returns or sets the number of lines in a command bar <u>combo box control</u>. The combo box control must be a custom control and it must be a drop-down list box or a combo box. Read/write **Long**.

**Note** An error occurs if you attempt to set this property for a combo box control that's an edit box or a built-in combo box control.

If this property is set to 0 (zero), the number of lines in the control will be based on the number of items in the list.

This example adds a combo box control containing two items to the command bar named "Custom". The example also sets the number of line items, the width of the combo box, and an empty default for the combo box.

```
Set myBar = CommandBars("Custom")
Set myControl = myBar.Controls.Add(Type:=msoControlComboBox, Id:=1)
With myControl
    .AddItem Text:="First Item", Index:=1
    .AddItem "Second Item", 2
    .DropDownLines = 3
    .DropDownWidth = 75
    .ListHeaderCount = 0
End With
```



# **DropDownWidth Property**

Returns or sets the width (in pixels) of the list for the specified command bar <u>combo box control</u>. Read/write **Long**.

**Note** An error occurs if you attempt to set this property for a built-in control.

If this property is set to -1, the width of the list is based on the length of the longest item in the combo box list. If this property is set to 0, the width of the list is based on the width of the control.

This example adds a combo box control containing two items to the command bar named "Custom". The example also sets the number of line items, the width of the combo box, and an empty default for the combo box.

```
Set myBar = CommandBars("Custom")
Set myControl = myBar.Controls.Add(Type:=msoControlComboBox, Id:=1)
With myControl
    .AddItem "First Item", 1
    .AddItem "Second Item", 2
    .DropDownLines = 3
    .DropDownWidth = 75
    .ListHeaderCount = 0
End With
```

# **DueDate Property**

Returns or sets the optional due date and time of a **SharedWorkspaceTask** object. Read/write **Variant**.

expression.DueDate()

*expression* Required. An expression that returns a **SharedWorkspaceTask** object.

The following example sets the **DueDate** of all tasks in a shared workspace to 12:00 noon on December 31, 2005 and uploads these changes to the server using the **Save** method.

```
Dim swsTask As Office.SharedWorkspaceTask
Const dtmNewDueDate As Date = #12/31/2005 12:00:00 PM#
For Each swsTask In ActiveWorkbook.SharedWorkspace.Tasks
    swsTask.DueDate = dtmNewDueDate
    swsTask.Save
Next
Set swsTask = Nothing
```

# **Email Property**

Returns the email name of the specified **SharedWorkspaceMember** in the format user@domain.com. Read-only **String**.

#### expression.Email

*expression* Required. An expression that returns a **SharedWorkspaceMember** object.

The following example extracts the email domain name from the **Email** property of each shared workspace member and lists members who have email addresses at the "mycompany.com" domain.

```
Dim swsMember As Office.SharedWorkspaceMember
Dim strEmailDomain As String
Dim strMemberList As String
For Each swsMember In ActiveWorkbook.SharedWorkspace.Members
    strEmailDomain = LCase(Right(swsMember.Email, _
        Len(swsMember.Email) - InStr(swsMember.Email, "@")))
    If strEmailDomain = "mycompany.com" Then
        strMemberList = strMemberList & swsMember.Email & vbCrLf
    End If
Next
MsgBox strMemberList, vbInformation + vbOKOnly, _
    "Members with mycompany.com Email"
Set swsMember = Nothing
```



# **Enabled Property**

Enabled property as it applies to the **CommandBar**, **CommandBarButton**, **CommandBarComboBox**, and **CommandBarControl** objects.

**True** if the specified <u>command bar</u> or <u>command bar control</u> is enabled. Read/write **Boolean**.

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 as it applies to the **Permission** object.

Returns or sets a **Boolean** value that indicates whether permissions are enabled on the active document. Read/write **Boolean**.

#### expression.Enabled

*expression* Required. An expression that returns a **Permission** object.

Use the **Enabled** property to determine whether permissions are restricted on the active document, and to enable or disable permissions. Set **Enabled** to **False** to disable permissions and to remove all users, other than the document author, and their permissions.

When permissions are disabled, the **Count** property of the **Permission** object returns 0 (zero); however, when permissions are re-enabled, the permissions of the document author remain intact.

As it applies to the **CommandBar**, **CommandBarButton**, **CommandBarComboBox**, and **CommandBarControl** objects.

This example adjusts the command bars according to the user level specified by user. If user is "Level 1," the command bar named "VB Custom Bar" is displayed. If user is any other value, the built-in Visual Basic command bar is reset to its default state and the command bar named "VB Custom Bar" is disabled.

```
Set myBar = CommandBars _
    .Add(Name:="VB Custom Bar", Position:=msoBarTop, _
    Temporary:=True)
With myBar
    .Controls.Add Type:=msoControlButton, ID:=2
    .Visible = True
End With
If user = "Level 1" Then
    myBar.Visible = True
Else
    CommandBars("Visual Basic").Reset
    myBar.Enabled = False
End If
```

This example adds two command bar buttons to the command bar named "Custom". The first control is disabled; the second control is enabled by default.

```
Set myBar = CommandBars("Custom")
With myBar
    .Controls.Add Type:=msoControlButton, Id:=3
    .Controls(1).Enabled = False
    .Controls.Add Type:=msoControlButton, Id:=3
End With
myBar.Visible = True
```

As it applies to the **Permission** object.

The following example checks the **Enabled** property to determine whether permissions are restricted on the active document.

```
Dim irmPermission As Office.Permission
Set irmPermission = ActiveWorkbook.Permission
Select Case irmPermission.Enabled
Case True
    MsgBox "Permissions are restricted on this document." &
    "There are " & irmPermission.Count & " users with rights
    vbInformation + vbOKOnly, "IRM Information"
Case False
    MsgBox "Permissions are NOT restricted on this document.
    vbInformation + vbOKOnly, "IRM Information"
End Select
Set irmPermission = Nothing
```

# **EnableTrustedBrowser Property**

Returns or sets the option that allows a user to view a document with restricted permissions in a web browser if the user does not have the appropriate client application installed. Read/write **Boolean**. Default is **False**.

#### expression. EnableTrustedBrowser

*expression* Required. An expression that returns a **Permission** object.

The **EnableTrustedBrowser** property, when **True**, allows a user who tries to open a Office Word 2003 document with restricted permissions, for example, but who does not have the Office Word 2003 application installed, to view the document in Microsoft Internet Explorer or another browser that supports this functionality. This property corresponds to the checkbox **Allow users with earlier versions of Office to read with browsers supporting Information Rights Management. (Increases file size.)** in the user interface.

The following example reports whether the active document has restricted permissions and, if so, whether the **EnableTrustedBrowser** property is also enabled.

```
Dim irmPermission As Office.Permission
Set irmPermission = ActiveWorkbook.Permission
Select Case irmPermission.Enabled
   Case True
        Select Case irmPermission.EnableTrustedBrowser
            Case True
                MsgBox "Permissions are restricted on this docum
                    vbCrLf & "Viewing in trusted browser is also
                    vbInformation + vbOKOnly, "IRM Information"
            Case False
                MsgBox "Permissions are restricted on this docum
                    vbCrLf & "However, viewing in trusted browse
                        vbInformation + vbOKOnly, "IRM Informati
        End Select
   Case False
       MsqBox "Permissions are NOT restricted on this document.
            vbInformation + vbOKOnly, "IRM Information"
End Select
Set irmPermission = Nothing
```



# **ErrorType Property**

Returns a constant which indicates the type of the most recent document synchronization error. Read-only **MsoSyncErrorType**.

MsoSyncErrorType can be one of the following msoSyncErrorType constants. msoSyncErrorNone (0) msoSyncErrorUnauthorizedUser (1) msoSyncErrorCouldNotConnect (2) msoSyncErrorOutOfSpace (3) msoSyncErrorFileNotFound (4) msoSyncErrorFileTooLarge (5) msoSyncErrorFileInUse (6) msoSyncErrorVirusUpload (7) msoSyncErrorVirusDownload (8) msoSyncErrorUnknownUpload (9) msoSyncErrorUnknownDownload (10) msoSyncErrorCouldNotOpen (11) msoSyncErrorCouldNotUpdate (12) msoSyncErrorCouldNotCompare (13) msoSyncErrorCouldNotResolve (14) msoSyncErrorNoNetwork (15) msoSyncErrorUnknown (16)

```
expression.ErrorType()
```

*expression* Required. An expression that returns a **Sync** object.

Use the **ErrorType** property to determine the type of the most recent document synchronization error.

Not all document synchronization problems raise trappable run-time errors. After performing an operation using the **Sync** object, it's a good idea to check the **Status** property; if the **Status** property is **msoSyncStatusError**, check the **ErrorType** property for additional information on the type of error that has occurred.

The following example checks the **Status** property, then uses the **ErrorType** property to display the type of sync error that has occurred.

```
Dim objSync As Office.Sync
Dim strError As String
Set objSync = ActiveDocument.Sync
If objSync.Status = msoSyncStatusError Then
    strError = "A sync error has occurred." & vbCrLf & _
        "Error type: " & objSync.ErrorType
Else
    strError = "No sync error has occurred."
End If
MsgBox strError, vbInformation + vbOKOnly, "Error Information"
Set objSync = Nothing
```

# **ExpirationDate Property**

Returns or sets the optional expiration date of the permissions on the active document assigned to the user associated with the specified **UserPermission** object. Read/write **Variant**.

#### expression. Expiration Date

*expression* Required. An expression that returns a **UserPermission** object.

The **UserPermission** object associates a set of permissions on the active document with a single user and an optional expiration date. The **ExpirationDate** property returns or sets the optional expiration date of the specified **UserPermission** object using the local time zone.

The following example extends the expiration date of the second user's permissions on the active document by an additional six months.

```
Dim irmPermission As Office.Permission
Dim irmUserPerm As Office.UserPermission
Dim dtmOldDate As Date, dtmNewDate As Date
Set irmPermission = ActiveWorkbook.Permission
Set irmUserPerm = irmPermission.Item(2)
dtmOldDate = irmUserPerm.ExpirationDate
dtmNewDate = DateAdd("m", 6, dtmOldDate)
irmUserPerm.ExpirationDate = dtmNewDate
MsgBox "Permissions expiration for " & irmUserPerm.UserId & vbCr
        "extended from " & dtmOldDate & " to " & dtmNewDate, _
        vbInformation + vbOKOnly, "IRM Information"
Set irmUserPerm = Nothing
Set irmPermission = Nothing
```

# **ExpireDate Property**

Returns a **Variant** representing the date on which the digital signature that corresponds to the **Signature** object will expire. Read-only.

#### expression. ExpireDate

*expression* Required. An expression that returns one of the objects in the Applies To list.

The following example prompts the user to select a digital signature with which to sign the active document in Microsoft Word. To use this example, open a document in Word and call this function. The function will test to make sure that the digital signature that the user selects will not expire in less than 12 months. If it will expire, the certificate isn't attached.

```
Function AddSignature() As Boolean
    On Error GoTo Error Handler
    Dim sig As Signature
    'Display the dialog box that lets the
    'user select a digital signature.
    'If the user selects a signature, then
    'it is added to the Signatures
    'collection. If the user doesn't, then
    'an error is returned.
    Set sig = ActiveDocument.Signatures.Add
    'Test to make sure that the new Signature object
    'doesn't expire too soon. This expression calculates
    'the number of months until the Signature object expires.
    If DateDiff("m", sig.SignDate, sig.ExpireDate) < 12 Then</pre>
        MsgBox "This Certificate will expire in less than 1 year." &
        "Please use a newer certificate."
        AddSignature = False
        sig.Delete
    Else
        AddSignature = True
    End If
    'Commit all signatures in the SignatureSet collection to the dis
    ActiveDocument.Signatures.Commit
    Exit Function
Error Handler:
    AddSignature = False
    MsgBox "Action cancelled."
```

```
End Function
```

# **Extended Property**

Sets or returns attributes added to the <SCRIPT> tag, with the exception of the LANGUAGE and ID attributes. Read/write **String**.

Attributes are separated by spaces, the same as in HTML. You cannot pass the LANGUAGE attribute or the ID attribute by using the **Extended** property.

The Microsoft Office host application doesn't provide any means of checking the syntax of passed attributes.

If you pass the LANGUAGE attribute in the **Extended** property, the <SCRIPT> tag receives two language settings, which causes a conflict.

If you pass an ID attribute in the **Extended** property and no ID has been set through either the ID parameter of the **Add** method or the **Id** property of the **Script** object, the ID is exported correctly.

This example checks the **Extended** property to ensure that no additional attributes have been added to the first script in worksheet one in the active workbook.

# **Extensions Property**

Returns a **String** value containing the extensions that determine which files are displayed in a file dialog box for each **Filter** object. Read-only.

#### expression. Extensions

*expression* Required. An expression that returns one of the objects in the Applies To list.

The following example iterates through the default filters of the SaveAs dialog box and displays the description of each filter that includes a Microsoft Excel file. The **Extensions** property is used to find the appropriate filter objects.

```
Sub Main()
    'Declare a variable as a FileDialogFilters collection.
    Dim fdfs As FileDialogFilters
    'Declare a variable as a FileDialogFilter object.
    Dim fdf As FileDialogFilter
    'Set the FileDialogFilters collection variable to
    'the FileDialogFilters collection of the SaveAs dialog box.
    Set fdfs = Application.FileDialog(msoFileDialogSaveAs).Filters
    'Iterate through the description and extensions of each
    'default filter in the SaveAs dialog box.
    For Each fdf In fdfs
        'Display the description of filters that include
        'Microsoft Excel files.
        If InStr(1, fdf.Extensions, "xls", vbTextCompare) > 0 Then
            MsgBox "Description of filter: " & fdf.Description
        End If
    Next fdf
End Sub
```



# **FaceId Property**

Returns or sets the Id number for the face of a command bar <u>button control</u>. Read/write **Long**.

The **FaceId** property dictates the look, not the function, of a command bar button. The **Id** property of the **CommandBarControl** object determines the function of the button.

The value of the **FaceId** property for a command bar button with a custom face is 0 (zero).

This example adds a command bar button to a custom command bar. Clicking this button is equivalent to clicking the **Open** command on the **File** menu because the ID number is 23, yet the button has the same button face as the built-in **Charting** button.

```
Set newBar = CommandBars.Add(Name:="Custom2", _
    Position:=msoBarTop, Temporary:=True)
newBar.Visible = True
Set con = newBar.Controls.Add(Type:=msoControlButton, Id:=23)
con.FaceId = 17
```

## **FeatureTips Property**

Some of the content in this topic may not be applicable to some languages.

**True** if the Office Assistant provides information about using application features more effectively. Read/write **Boolean**.

The **FeatureTips** property corresponds to the **Using features more effectively** check box on the **Options** tab in the **Assistant** dialog box.

This example allows the Office Assistant to provide information about using application features more effectively.

Assistant.**FeatureTips** = True

## **FileName Property**

Some of the content in this topic may not be applicable to some languages.

**Assistant** object: Returns or sets the path and file name for the active Office Assistant. Read/write **String**.

**FileSearch** object: Returns or sets the name of the file to look for during a file search. The name of the file may include the \* (asterisk) or ? (question mark) wildcards. Use the question mark wildcard to match any single character. For example, type **gr**?**y** to match both "gray" and "grey." Use the asterisk wildcard to match any number of characters. For example, type **\*.txt** to find all files that have the .TXT extension. Read/write **String**.

#### expression.FileName

*expression* Required. An expression that returns one of the objects in the Applies To list.

This example searches for all files located in the My Documents folder that begin with "cmd" and have a file name extension. The example displays the name and location of each found file.

```
Set fs = Application.FileSearch
With fs
.LookIn = "C:\My Documents"
.FileName = "cmd*.*"
If .Execute > 0 Then
        MsgBox "There were " & .FoundFiles.Count & _
        " file(s) found."
    For i = 1 To .FoundFiles.Count
        MsgBox .FoundFiles(i)
        Next i
    Else
        MsgBox "There were no files found."
    End If
End With
```



## **Files Property**

Files property as it applies to the **AnswerWizard** object.

Returns an <u>AnswerWizardFiles</u> collection that represents the list of files available to the current AnswerWizard. Read-only.

For information about returning a single member of a collection, see <u>Returning</u> <u>an Object from a Collection</u>.

Files property as it applies to the SharedWorkspace object.

Returns a **SharedWorkspaceFiles** collection that represents the list of files stored in the document library associated with the current shared workspace. Read-only.

As it applies to the **AnswerWizard** object.

This example resets the file list for the current AnswerWizard and then displays both the file count and the file names in a message box.

```
Dim customAnswerWizard As AnswerWizard
Dim strFileList As String
Dim intCounter As Integer
Dim intNumFiles As Integer
Set customAnswerWizard = Application.AnswerWizard
intCounter = 1
customAnswerWizard.ResetFileList
strFileList = ""
intNumFiles = customAnswerWizard.Files.Count
For intCounter = 1 To (intNumFiles)
    strFileList = strFileList & _
    customAnswerWizard.Files.Item(intCounter) & Chr(13)
Next
MsgBox "There are " & customAnswerWizard.Files.Count & _
    " files avaialble through this AnswerWizard: " & _
    Chr(13) & strFileList
```

As it applies to the **SharedWorkspace** object.

The following example lists the files saved in the current shared workspace.

```
Dim swsFiles As Office.SharedWorkspaceFiles
Set swsFiles = ActiveWorkbook.SharedWorkspace.Files
MsgBox "There are " & swsFiles.Count & _
    "file(s)
    vbInformation + vbOKOnly, _
    "Collection Information"
Set swsFiles = Nothing
```



## **FileType Property**

Returns or sets the type of file to look for during a file search. Read/write

#### **MsoFileType**.

MsoFileType can be one of these MsoFileType constants. **msoFileTypeAllFiles msoFileTypeBinders** msoFileTypeCalendarItem msoFileTypeContactItem msoFileTypeCustom **msoFileTypeDatabases msoFileTypeDataConnectionFiles msoFileTypeDesignerFiles** msoFileTypeDocumentImagingFiles msoFileTypeExcelWorkbooks **msoFileTypeJournalItem** msoFileTypeMailItem msoFileTypeNoteItem msoFileTypeOfficeFiles **msoFileTypeOutlookItems** msoFileTypePhotoDrawFiles msoFileTypePowerPointPresentations **msoFileTypeProjectFiles msoFileTypePublisherFiles msoFileTypeTaskItem msoFileTypeTemplates msoFileTypeVisioFiles msoFileTypeWebPages** msoFileTypeWordDocuments

### expression.FileType

*expression* Required. An expression that returns one of the objects in the Applies To list.

The constant **msoFileTypeOfficeFiles** includes all files with any of the following extensions: \*.doc, \*.xls, \*.ppt, \*.pps, \*.obd, \*.mdb, \*.mpd, \*.dot, \*.xlt, \*.pot, \*.obt, \*.htm, or \*.html.

This example searches for all Binder files located in the My Documents folder. The example displays a message box that contains the name and location of each file that's found.

```
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
```

# **FileTypes Property**

Returns a **<u>FileTypes</u>** collection.

expression.FileTypes

*expression* Required. An expression that returns one of the objects in the Applies To list.

The following example searches for all HTML and Microsoft Excel files on the C:\ drive.

```
Sub SearchForFiles()
    'Declare a variable to act as a generic counter.
    Dim lngCount As Long
    'Use a With...End With block to reference the
    'FileSearch object.
    With Application.FileSearch
        'Clear all the parameters of the previous searches.
        'This method doesn't clear the LookIn property or
        'the SearchFolders collection.
        .NewSearch
        'Setting the FileType property clears the
        'FileTypes collection and sets the first
        'item in the collection to the file type
        'defined by the FileType property.
        .FileType = msoFileTypeWebPages
        'Add a second item to the FileTypes collection.
        .FileTypes.Add msoFileTypeExcelWorkbooks
        'Display the number of FileTypes objects in the collection.
        MsgBox "You are about to search for " & .FileTypes.Count & _
            " file types."
        'Set up the search to look in all subfolders on the C:\ driv
        .LookIn = "C: \
        .SearchSubFolders = True
        'Execute the search and test to see if any files
        'were found.
        If .Execute <> 0 Then
            'Display the number of files found.
            MsgBox "Files found: " & .FoundFiles.Count
            'Loop through the list of found files and
            'display the path of each one in a message box.
            For lngCount = 1 To .FoundFiles.Count
```

## **FilterIndex Property**

Returns or sets a **Long** indicating the default file filter of a file dialog box. The default filter determines which types of files are displayed when the file dialog box is first opened. Read/write.

#### expression.FilterIndex

*expression* Required. An expression that returns one of the objects in the Applies To list.

If you try to set this property to a number greater than the number of filters, the last available filter will be selected.

The following example displays a File Picker dialog box using the **FileDialog** object and displays each selected file in a message box. This example also demonstrates how to add a new file filter and how to make it the default filter.

```
Sub Main()
    'Declare a variable as a FileDialog object.
    Dim fd As FileDialog
    'Create a FileDialog object as a File Picker dialog box.
    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    'Declare a variable to contain the path
    'of each selected item. Even though the path is a String,
    'the variable must be a Variant because For Each...Next
    'routines only work with Variants and Objects.
    Dim vrtSelectedItem As Variant
    'Use a With...End With block to reference the FileDialog object.
   With fd
        'Add a filter that includes GIF and JPEG images and make it
        .Filters.Add "Images", "*.gif; *.jpg; *.jpeg", 2
        'Sets the initial file filter to number 2.
        .FilterIndex = 2
        'Use the Show method to display the File Picker dialog box a
        'If the user presses the action button...
        If .Show = -1 Then
            'Step through each string in the FileDialogSelectedItems
            For Each vrtSelectedItem In .SelectedItems
                'vrtSelectedItem is a String that contains the path
                'You can use any file I/O functions that you want to
                'This example simply displays the path in a message
                MsgBox "Selected item's path: " & vrtSelectedItem
            Next vrtSelectedItem
        'If the user presses Cancel...
        Else
        End If
    End With
```

```
'Set the object variable to Nothing.
Set fd = Nothing
```

End Sub

## **Filters Property**

Returns a **<u>FileDialogFilters</u>** collection.

expression.Filters

*expression* Required. An expression that returns one of the objects in the Applies To list.

The following example displays a File Picker dialog box using the **FileDialog** object and displays each selected file in a message box. The example also adds a new file filter called "Images."

```
Sub Main()
    'Declare a variable as a FileDialog object.
    Dim fd As FileDialog
    'Create a FileDialog object as a File Picker dialog.
    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    'Declare a variable to contain the path
    'of each selected item. Even though the path is a String,
    'the variable must be a Variant because For Each...Next
    'routines only work with Variants and Objects.
    Dim vrtSelectedItem As Variant
    'Use a With...End With block to reference the FileDialog object.
   With fd
        'Add a filter that includes GIF and JPEG images and make it
        .Filters.Add "Images", "*.gif; *.jpg; *.jpeg", 1
        'Use the Show method to display the File Picker dialog box a
        'If the user presses the action button...
        If .Show = -1 Then
            'Step through each string in the FileDialogSelectedItems
            For Each vrtSelectedItem In .SelectedItems
                'vrtSelectedItem is a String that contains the path
                'You can use any file I/O functions that you want to
                'This example simply displays the path in a message
                MsgBox "Selected item's path: " & vrtSelectedItem
            Next vrtSelectedItem
        'If the user presses Cancel...
        Else
        End If
    End With
    'Set the object variable to Nothing.
    Set fd = Nothing
```

End Sub

# **FixedWidthFont Property**

Sets or returns the fixed-width font setting in the host application. Read/write **String**.

When you set the **FixedWidthFont** property, the host application does not check the value for validity.

This example sets the fixed-width font and fixed-width font size for the English/Western European/Other Latin Script character set in the active application.

```
Application.DefaultWebOptions. _
Fonts(msoCharacterSetEnglishWesternEuropeanOtherLatinScript) _
.FixedWidthFont = "System"
Application.DefaultWebOptions. _
Fonts(msoCharacterSetEnglishWesternEuropeanOtherLatinScript) _
.FixedWidthFontSize = 12
```

# **FixedWidthFontSize Property**

Sets or returns the fixed-width font size setting in the host application, in points. Read/write **Single**.

When you set the **FixedWidthFontSize** property, the host application does not check the value for validity. If you enter an invalid value, such as a nonnumber, the host application sets the size to 0 points. You can enter half-point sizes; if you enter other fractional point sizes, they are rounded up or down to the nearest half-point.

This example sets the fixed-width font and fixed-width font size for the English/Western European/Other Latin Script character set in the active application.

```
Application.DefaultWebOptions. _
Fonts(msoCharacterSetEnglishWesternEuropeanOtherLatinScript) _
.FixedWidthFont = "System"
Application.DefaultWebOptions. _
Fonts(msoCharacterSetEnglishWesternEuropeanOtherLatinScript) _
.FixedWidthFontSize = 12
```

## **FolderName Property**

Returns the name of a subfolder within the main document library folder of a shared workspace. Read-only **String**.

#### expression.FolderName()

*expression* Required. An expression that returns a **SharedWorkspaceFolder** object.

The **FolderName** property returns the subfolder name in the format parentfolder/subfolder. For example, if the shared workspace contains a folder named "Supporting Documents", the **FolderName** property returns Shared Documents/Supporting Documents.

The following example displays the number of subfolders in the shared workspace and their names.

```
Dim swsFolder As Office.SharedWorkspaceFolder
Dim strFolderInfo As String
strFolderInfo = "The shared workspace contains " & ______
ActiveWorkbook.SharedWorkspace.Folders.Count & " folder(s)."
If ActiveWorkbook.SharedWorkspace.Folders.Count > 0 Then
For Each swsFolder In ActiveWorkbook.SharedWorkspace.Folders
strFolderInfo = strFolderInfo & swsFolder.FolderName & v
Next
End If
MsgBox strFolderInfo, vbInformation + vbOKOnly, ______
"Folders in Shared Workspace"
Set swsFolder = Nothing
```

## **Folders Property**

Returns a **SharedWorkspaceFolders** collection that represents the list of subfolders in the document library associated with the current shared workspace. Read-only.

expression.Folders()

*expression* Required. An expression that returns a **SharedWorkspace** object.

The **SharedWorkspaceFolders** collection does not include the root document library folder itself, which by default is named "Shared Documents".

The following example lists the subfolders in the current shared workspace.

Dim swsFolders As Office.SharedWorkspaceFolders
Set swsFolders = ActiveWorkbook.SharedWorkspace.Folders
MsgBox "There are " & swsFolders.Count & \_
 "folder(s) in the current shared workspace.", \_
 vbInformation + vbOKOnly, \_
 "Collection Information"
Set swsFolders = Nothing

### **FoundFiles Property**

Returns a **FoundFiles** object that contains the names of all the files found during a search. Read-only.

This example steps through the list of files found during a search and displays the path for each file.

With Application.FileSearch
For i = 1 To .FoundFiles.Count
 MsgBox .FoundFiles(i)
Next I
End With

### **GuessHelp Property**

Some of the content in this topic may not be applicable to some languages.

**True** if the Office Assistant balloon presents a list of Help topics based on keywords the user selects before clicking the Assistant window or pressing F1. Read/write **Boolean**.

**Note** This property is obsolete but has been retained for compatibility with code written in previous versions of Microsoft Office.

The **GuessHelp** property corresponds to the **Guess help topics** option under **Use the Office Assistant** on the **Options** tab in the **Office Assistant** dialog box.

This example allows the Office Assistant to guess at Help topics.

Assistant.**GuessHelp** = True

## **Guid Property**

Returns the globally unique class identifier (GUID) for the specified **COMAddIn** object. Read-only **String**.

The following example displays the ProgID and GUID for COM add-in one in a message box.

MsgBox "My ProgID is " & \_ Application.COMAddIns(1).ProgID & \_ " and my GUID is " & \_ Application.COMAddIns(1).Guid

### **Heading Property**

Some of the content in this topic may not be applicable to some languages.

Returns or sets the heading that appears in the Office Assistant balloon. Read/write **String**.

You can specify a graphic to be displayed in the balloon heading by using the following syntax: {*type location sizing\_factor*}, where *type* is bmp (bitmap) or wmf (Windows metafile), *location* is the resource ID or the path and file name, and *sizing\_factor* specifies the width of the wmf (omitted for bmp).

The balloon heading also supports underlined text and text that has one of the 16 system palette colors applied to it. To display underlined text in a heading, use the syntax {*ul*} or {*ul* 1}; use {*ul* 0} to turn underlining off. To change the color of heading text, precede the text string with the character sequence {*cf number*}, where *number* is one of the system color numbers listed in the following table.

System color number	Color
0	Black
1	Dark red
2	Dark green
3	Dark yellow
4	Dark blue
5	Dark magenta
6	Dark cyan
7	Light gray
248	Medium gray
249	Red
250	Green
251	Yellow
252	Blue
253	Magenta
254	Cyan
255	White

If you specify a number other than one of the preceding system color numbers, the heading text is black.

This example displays a balloon with a heading, text, and three region choices.

```
With Assistant.NewBalloon
    .Button = msoButtonSetOkCancel
    .Heading = "Regional Sales Data"
    .Text = "Select a region"
    For i = 1 To 3
        .CheckBoxes(i).Text = "Region " & i
    Next
    .Show
End With
```



## **Height Property**

Returns or sets the height of a <u>command bar control</u> or <u>command bar</u>. Read/write **Long**.

expression.Height

*expression* Required. An expression that returns one of the above objects.

This example adds a custom control to the command bar named Custom. The example sets the height of the custom control to twice the height of the command bar and sets the control's width to 50 pixels. Notice how the command bar automatically resizes itself to accommodate the control.

```
Set myBar = CommandBars("Custom")
barHeight = myBar.Height
Set myControl = myBar.Controls _
    .Add(Type:=msoControlButton, _
    Id:= CommandBars("Standard").Controls("Save").Id, _
    Temporary:=True)
With myControl
    .Height = barHeight * 2
    .Width = 50
End With
myBar.Visible = True
```



### HelpContextId Property

Returns or sets the Help context Id number for the Help topic attached to the <u>command bar control</u>. Read/write **Long**.

To use this property, you must also set the **<u>HelpFile</u>** property. Help topics respond to Shift+F1.

This example adds a custom command bar with a combo box that tracks stock data. The example also specifies the Help topic to be displayed for the combo box when the user presses SHIFT+F1.

```
Set myBar = CommandBars _
    .Add(Name:="Custom", Position:=msoBarTop, _
    Temporary:=True)
With myBar
    .Controls.Add Type:=msoControlComboBox, ID:=1
    .Visible = True
End With
With CommandBars("Custom").Controls(1)
    .AddItem "Get Stock Quote", 1
    .AddItem "View Chart", 2
    .AddItem "View Fundamentals", 3
    .AddItem "View News", 4
    .Caption = "Stock Data"
    .DescriptionText = "View Data For Stock"
    .HelpFile = "C:\corphelp\custom.hlp"
    .HelpContextID = 47
End With
```



## **HelpFile Property**

Returns or sets the file name for the Help topic attached to the <u>command bar</u> <u>control</u>. Read/write **String**.

To use this property, you must also set the <u>HelpContextID</u> property. Help topics respond to the user pressing SHIFT+F1.

This example adds a custom command bar with a combo box that tracks stock data. The example also specifies the Help topic to be displayed for the combo box when the user presses SHIFT+F1.

```
Set myBar = CommandBars _
    .Add(Name:="Custom", Position:=msoBarTop, _
    Temporary:=True)
With myBar
    .Controls.Add Type:=msoControlComboBox, ID:=1
    .Visible = True
End With
With CommandBars("Custom").Controls(1)
    .AddItem "Get Stock Quote", 1
    .AddItem "View Chart", 2
    .AddItem "View Fundamentals", 3
    .AddItem "View News", 4
    .Caption = "Stock Data"
    .DescriptionText = "View Data For Stock"
    .HelpFile = "C:\corphelp\custom.hlp"
    .HelpContextID = 47
End With
```

## **HighPriorityTips Property**

Some of the content in this topic may not be applicable to some languages.

**True** if the Office Assistant displays high-priority tips. Read/write **Boolean**.

The **HighPriorityTips** property corresponds to the **Only show high priority tips** option under **Show tips about** on the **Options** tab in the **Office Assistant** dialog box.

This example sets the Office Assistant to display high-priority tips.

Assistant.**HighPriorityTips** = True

### **HTMLProjectItems Property**

Returns the **HTMLProjectItems** collection that is included in the specified HTML project. Read-only **HTMLProjectItems**.

This example returns the number of items in the **HTMLProjectItems** collection in the HTML project in the active workbook.

intCount = ActiveWorkbook.HTMLProject.HTMLProjectItems.Count



# HyperlinkType Property

Sets or returns the type of hyperlink associated with the specified command bar button. Read/write **MsoCommandBarButtonHyperlinkType**.

MsoCommandBarButtonHyperlinkType can be one of these MsoCommandBarButtonHyperlinkType constants. **msoCommandBarButtonHyperlinkInsertPicture msoCommandBarButtonHyperlinkNone msoCommandBarButtonHyperlinkOpen** 

This example checks the **HyperlinkType** property for the specified command bar button on the command bar named "Custom.". If **HyperlinkType** is set to **msoCommandBarButtonHyperlinkNone**, the example sets the property to **msoCommandBarButtonHyperlinkOpen** and sets the URL to www.microsoft.com.

```
Set myBar = CommandBars ______.Add(Name:="Custom", Position:=msoBarTop, ______
Temporary:=True)
Set myButton = myBar.Controls.Add(Type:=msoControlButton)
With myButton
.FaceId = 277
.HyperlinkType = msoCommandBarButtonHyperlinkNone
End With
If myButton.HyperlinkType <> _______
msoCommandBarButtonHyperlinkOpen Then
myButton.HyperlinkType = _______msoCommandBarButtonHyperlinkOpen
myButton.TooltipText = "www.microsoft.com"
End If
```



### **Icon Property**

Some of the content in this topic may not be applicable to some languages.

Returns or sets the type of icon that appears in the upper-left portion of the Office Assistant balloon. Read/write **MsoIconType**.

MsoIconType can be one of these MsoIconType constants. msoIconAlert msoIconAlertCritical msoIconAlertInfo msoIconAlertQuery msoIconAlertWarning msoIconNone

msoIconTip

This example creates a balloon with an "Alert" icon that instructs the user to select a printer.

```
With Assistant.NewBalloon
    .Heading = "Select A Printer"
    .Text = "You must select a printer before printing."
    .Icon = msoIconAlert
    .CheckBoxes(1).Text = "Local printer"
    .CheckBoxes(2).Text = "Network printer"
    .Show
End With
```



## **Id Property**

Id property as it applies to the **CommandBarButton**, **CommandBarComboBox**, and **CommandBarControl** objects.

Returns the ID for a built-in <u>command bar control</u>. Read-only **Long**.

expression.Id

*expression* Required. An expression that returns one of the above objects.

### Remarks

A control's ID determines the built-in action for that control. The value of the **Id** property for all custom controls is 1.

Id property as it applies to the **Script** object.

Sets or returns the ID of a **Script** object. Read/write **String**.

expression.Id

*expression* Required. An expression that returns a **Script** object.

## Remarks

The ID returned is the ID attribute of the <SCRIPT> tag in HTML. If there's no ID attribute specified in the <SCRIPT> tag, the **Id** property returns an empty string.

**Id** specifies an SGML identifier used for naming elements. Valid identifiers include any string that begins with a letter and is composed of alphanumeric characters; the string can also include the underscore character ( \_ ).

The ID must be unique within the HTML document.

As it applies to the **CommandBarButton**, **CommandBarComboBox**, and **CommandBarControl** objects.

This example changes the button face of the first control on the command bar named "Custom2" if the button's ID value is less than 25.

```
Set ctrl = CommandBars("Custom").Controls(1)
With ctrl
    If .Id < 25 Then
        .FaceId = 17
        .Tag = "Changed control"
    End If
End With</pre>
```

The following example changes the caption of every control on the toolbar named "Standard" to the current value of the **Id** property for that control.

For Each ctl In CommandBars("Standard").Controls
 ctl.Caption = CStr(ctl.Id)
Next ctl

As it applies to the **Script** object.

This example sets the **Id** property of the first script in worksheet one of the active workbook to a new value.

```
ActiveWorkbook.Worksheets(1).Scripts(1).Id = "UpdatedScriptName"
```

# **Index Property**

Returns a **Long** representing the index number for an object in the collection. Read-only.

expression.Index

## Remarks

The position of the first command bar control is 1. Separators are not counted in the **CommandBarControls** collection.

This example searches the command bar named "Custom2" for a control with an Id value of 23. If such a control is found and the index number of the control is greater than 5, the control will be positioned as the first control on the command bar.

```
Set myBar = CommandBars("Custom2")
Set ctrl1 = myBar.FindControl(Id:=23)
If ctrl1.Index > 5 Then
    ctrl1.Move before:=1
End If
```

# **InitialFileName Property**

Set or returns a **String** representing the path and/or file name that is initially displayed in a file dialog box. Read/write.

#### expression.InitialFileName

### Remarks

You can use the '\*' and '?' wildcard characters when specifying the file name but not when specifying the path. The '\*' represents any number of consecutive characters and the '?' represents a single character. For example,

```
.InitialFileName = "c:\c*s.txt" will return both "charts.txt" and "checkregister.txt."
```

If you specify a path and no file name, then all files that are allowed by the file filter will appear in the dialog box.

If you specify a file that exists in the initial folder, then only that file will appear in the dialog box.

If you specify a file name that doesn't exist in the initial folder, then the dialog box will contain no files. The type of file that you specify in the **InitialFileName** property will override the file filter settings.

If you specify an invalid path, the last-used path is used. A message will warn users when an invalid path is used.

Setting this property to a string longer than 256 characters will cause a run-time error.

The following example displays a File Picker dialog box using the **FileDialog** object and displays each selected file in a message box.

```
Sub Main()
    'Declare a variable as a FileDialog object
    Dim fd As FileDialog
    'Create a FileDialog object as a File Picker dialog box.
    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    'Declare a variable to contain the path
    'of each selected item. Even though the path is a String,
    'the variable must be a Variant because For Each...Next
    'routines only work with Variants and Objects.
    Dim vrtSelectedItem As Variant
    'Use a With...End With block to reference the FileDialog object.
   With fd
        'Set the initial path to the C:\ drive.
        .InitialFileName = "C:\"
        'Use the Show method to display the File Picker dialog box a
        'If the user presses the action button...
        If .Show = -1 Then
            'Step through each string in the FileDialogSelectedItems
            For Each vrtSelectedItem In .SelectedItems
                'vrtSelectedItem is a String that contains the path
                'You can use any file I/O functions that you want to
                'This example simply displays the path in a message
                MsgBox "Selected item's path: " & vrtSelectedItem
            Next vrtSelectedItem
        'If the user presses Cancel...
        Else
        End If
    End With
    'Set the object variable to Nothing.
    Set fd = Nothing
```

End Sub



# **InitialView Property**

Returns or sets an **MsoFileDialogView** constant representing the initial presentation of files and folders in a file dialog box. Read/write.

MsoFileDialogView can be one of these MsoFileDialogView constants.

msoFileDialogViewDetails

msoFileDialogViewLargeIcons

msoFileDialogViewList

msoFileDialogViewPreview

msoFileDialogViewProperties

msoFileDialogViewSmallIcons

**msoFileDialogViewThumbnail** This constant is only available in conjunction with Microsoft Windows 2000 or Microsoft Windows Millennium Edition, or later.

**msoFileDialogViewWebView** Not available. If you select this constant, the default view will be used.

#### expression.InitialView

The following example displays a File Picker dialog box in details view using the **FileDialog** object and displays each selected file in a message box.

```
Sub Main()
    'Declare a variable as a FileDialog object.
    Dim fd As FileDialog
    'Create a FileDialog object as a File Picker dialog box.
    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    'Declare a variable to contain the path
    'of each selected item. Even though the path is a String,
    'the variable must be a Variant because For Each...Next
    'routines only work with Variants and Objects.
    Dim vrtSelectedItem As Variant
    'Use a With...End With block to reference the FileDialog object.
   With fd
        'Set the initial view to the details view.
        .InitialView = msoFileDialogViewDetails
        'Use the Show method to display the File Picker dialog box a
        'If the user presses the action button...
        If .Show = -1 Then
            'Step through each string in the FileDialogSelectedItems
            For Each vrtSelectedItem In .SelectedItems
                'vrtSelectedItem is a String that contains the path
                'You can use any file I/O functions that you want to
                'This example simply displays the path in a message
                MsgBox "Selected item's path: " & vrtSelectedItem
            Next vrtSelectedItem
        'If the user presses Cancel...
        Else
        End If
    End With
    'Set the object variable to Nothing.
    Set fd = Nothing
```

End Sub

## **Introduction Property**

Sets or returns the introductory text that is included with a document that is sent using the **MsoEnvelope** object. The introductory text is included at the top of the document in the e-mail. Read/write **String**.

#### expression.Introduction

The following example sends the active Microsoft Word document as an e-mail to the e-mail address that you pass to the subroutine.

Sub SendMail(ByVal strRecipient As String) 'Use a With...End With block to reference the MsoEnvelope object With Application.ActiveDocument.MailEnvelope 'Add some introductory text before the body of the e-mail. .Introduction = "Please read this and send me your comments. 'Return a MailItem object that you can use to send the docum With .Item 'All of the mail item settings are saved with the docume 'When you add a recipient to the Recipients collection 'or change other properties these settings will persist. .Recipients.Add strRecipient .Subject = "Here is the document." 'The body of this message will be 'the content of the active document. .Send End With End With End Sub

# **IsCertificateExpired Property**

**True** if the digital certificate that corresponds to the **Signature** object has expired. Read-only **Boolean**.

#### expression.IsCertificateExpired

The following example prompts the user to select a digital signature with which to sign the active document in Microsoft Word. To use this example, open a document in Word and pass this function the name of a certificate issuer and the name of a certificate signer that match the **Issued By** and **Issued To** fields of a digital certificate in the **Digital Certificates** dialog box. This example will test to make sure that the digital signature that the user selects meets certain criteria, such as not having expired, before the new signature is committed to the disk.

```
Function AddSignature(ByVal strIssuer As String,
    strSigner As String) As Boolean
    On Error GoTo Error_Handler
    Dim sig As Signature
    'Display the dialog box that lets the
    'user select a digital signature.
    'If the user selects a signature, then
    'it is added to the Signatures
    'collection. If the user doesn't, then
    'an error is returned.
    Set sig = ActiveDocument.Signatures.Add
    'Test several properties before commiting the Signature object t
    If sig.Issuer = strIssuer And
        sig.Signer = strSigner And _
        sig.IsCertificateExpired = False And _
        sig.IsCertificateRevoked = False And _
        siq.IsValid = True Then
        MsgBox "Signed"
        AddSignature = True
    'Otherwise, remove the Signature object from the SignatureSet co
    Else
        sig.Delete
        MsgBox "Not signed"
        AddSignature = False
    End If
    'Commit all signatures in the SignatureSet collection to the dis
    ActiveDocument.Signatures.Commit
```

Exit Function

```
Error_Handler:
AddSignature = False
MsgBox "Action cancelled."
End Function
```

# IsCertificateRevoked Property

**True** if the digital certificate that corresponds to the **Signature** object has been revoked by the issuer of the certificate. Read-only **Boolean**.

#### expression.IsCertificateRevoked

The following example prompts the user to select a digital signature with which to sign the active document in Microsoft Word. To use this example, open a document in Word and pass this function the name of a certificate issuer and the name of a certificate signer that match the **Issued By** and **Issued To** fields of a digital certificate in the **Digital Certificates** dialog box. This example will test to make sure that the digital signature that the user selects meets certain criteria, such as not having expired, before the new signature is committed to the disk.

```
Function AddSignature(ByVal strIssuer As String,
    strSigner As String) As Boolean
    On Error GoTo Error_Handler
    Dim sig As Signature
    'Display the dialog box that lets the
    'user select a digital signature.
    'If the user selects a signature, then
    'it is added to the Signatures
    'collection. If the user doesn't, then
    'an error is returned.
    Set sig = ActiveDocument.Signatures.Add
    'Test several properties before commiting the Signature object t
    If sig.Issuer = strIssuer And
        sig.Signer = strSigner And _
        sig.IsCertificateExpired = False And _
        sig.IsCertificateRevoked = False And _
        siq.IsValid = True Then
        MsgBox "Signed"
        AddSignature = True
    'Otherwise, remove the Signature object from the SignatureSet co
    Else
        sig.Delete
        MsgBox "Not signed"
        AddSignature = False
    End If
    'Commit all signatures in the SignatureSet collection to the dis
    ActiveDocument.Signatures.Commit
```

Exit Function

```
Error_Handler:
AddSignature = False
MsgBox "Action cancelled."
End Function
```

# **IsOpen Property**

**True** if the specified HTML project item is open in the Microsoft Script Editor. Read-only **Boolean**.

This example determines whether the specified HTML project item is open. and if it is, the example then loads script from the specified file.

```
If ActiveWorkbook.HTMLProject.HTMLProjectItems _
    .Item(1).IsOpen Then
    ActiveWorkbook.HTMLProject.HTMLProjectItems _
    .Item(1).LoadFromFile ("C:\MyScript.txt")
Else
    MsgBox "The HTMLProjectItem is not open."
End If
```

# **IsPriorityDropped Property**

**True** if the control is currently dropped from the menu or toolbar based on usage statistics and layout space. (Note that this is not the same as the control's visibility, as set by the **Visible** property.) A control with **Visible** set to **True**, will not be immediately visible on a **Personalized Menu** or **Toolbar** if **IsPriorityDropped** is **True**. Read-only **Boolean**.

### Remarks

To determine when to set **IsPriorityDropped** to **True** for a specific menu item, Microsoft Office maintains a total count of the number of times the menu item was used and a record of the number of different application sessions in which the user has used another menu item in the same menu as this menu item, without using the specific menu item. When this value reaches certain threshold values, the count is decremented. When the count reaches zero, **IsPriorityDropped** is set to **True**. Programmers cannot set the session value, the threshold value, or the **IsPriorityDropped** property. Programmers can, however, use the **AdaptiveMenus** property to disable adaptive menus for specific menus

in an application.

To determine when to set **IsPriorityDropped** to **True** for a specific toolbar control, Office maintains a list of the order in which all the controls on that toolbar were last executed. A toolbar will always show as many controls as it has space to show, in the order of most recently used to least recently used. Controls with **Priority** set to 1 will always be shown and the toolbar will wrap rows, if necessary, to show these controls. Programmers can use the **Priority** property to ensure that specific toolbar controls are always shown, or to reposition toolbars so that they have enough space to display all of their controls.

You can use the following table to predict the number of sessions for which a menu item on a Personalized Menu will remain visible before the menu item's **IsPriorityDropped** property is set to **True**.

Number of uses of the command bar control	Number of sessions of the application
0, 1	3
2	6
3	9
4, 5	12
6–8	17
9–13	23
14–24	29
25 or more	31

This example checks the **IsPriorityDropped** property for the first control on the command bar named "Custom." If **IsPriorityDropped** is **True**, the example sets the **AdaptiveMenus** property to **False**, restoring the dropped command bar.

If CommandBars("Custom").Controls(1).IsPriorityDropped Then CommandBars.AdaptiveMenus = True End If

## **Issuer Property**

Returns a **String** representing the name of the issuer of the digital certificate that corresponds to the **Signature** object. Read-only.

#### expression.Issuer

The following example prompts the user to select a digital signature with which to sign the active document in Microsoft Word. To use this example, open a document in Word and pass this function the name of a certificate issuer and the name of a certificate signer that match the **Issued By** and **Issued To** fields of a digital certificate in the **Digital Certificates** dialog box. This example will test to make sure that the digital signature that the user selects meets certain criteria, such as not having expired, before the new signature is committed to the disk.

```
Function AddSignature(ByVal strIssuer As String,
    strSigner As String) As Boolean
    On Error GoTo Error_Handler
    Dim sig As Signature
    'Display the dialog box that lets the
    'user select a digital signature.
    'If the user selects a signature, then
    'it is added to the Signatures
    'collection. If the user doesn't, then
    'an error is returned.
    Set sig = ActiveDocument.Signatures.Add
    'Test several properties before commiting the Signature object t
    If sig.Issuer = strIssuer And
        sig.Signer = strSigner And
        sig.IsCertificateExpired = False And _
        sig.IsCertificateRevoked = False And _
        siq.IsValid = True Then
        MsgBox "Signed"
        AddSignature = True
    'Otherwise, remove the Signature object from the SignatureSet co
    Else
        sig.Delete
        MsgBox "Not signed"
        AddSignature = False
    End If
    'Commit all signatures in the SignatureSet collection to the dis
    ActiveDocument.Signatures.Commit
```

Exit Function

```
Error_Handler:
AddSignature = False
MsgBox "Action cancelled."
End Function
```

# **IsValid Property**

**True** if the digital signature that corresponds to the **Signature** object is a valid signature. A signature may be invalid for several reasons ranging from its certificate having expired to changes in the document that contains it. Read-only **Boolean**.

expression.IsValid

The following example prompts the user to select a digital certificate with which to sign the active document in Microsoft Word. To use this example, open a document in Word and pass this function the name of a certificate issuer and the name of a certificate signer that match the **Issued By** and **Issued To** fields of a digital certificate in the **Digital Certificates** dialog box. This example will test to make sure that the digital certificate that the user selects meets certain criteria, such as not having expired, before the new signature is committed to the disk.

```
Function AddSignature(ByVal strIssuer As String,
    strSigner As String) As Boolean
    On Error GoTo Error_Handler
    Dim sig As Signature
    'Display the dialog box that lets the
    'user select a digital certificate.
    'If the user selects a certificate,
    'then a signature is created and
    'it is added to the Signatures
    'collection. If the user doesn't, then
    'an error is returned.
    Set sig = ActiveDocument.Signatures.Add
    'Test several properties before commiting the Signature object t
    If sig.Issuer = strIssuer And
        sig.Signer = strSigner And _
        sig.IsCertificateExpired = False And _
        sig.IsCertificateRevoked = False And _
        sig.IsValid = True Then
        MsgBox "Signed"
        AddSignature = True
    'Otherwise, remove the Signature object from the SignatureSet co
    Else
        sig.Delete
        MsqBox "Not signed"
        AddSignature = False
    End If
```

'Commit all signatures in the SignatureSet collection to the dis ActiveDocument.Signatures.Commit

```
Exit Function
Error_Handler:
AddSignature = False
MsgBox "Action cancelled."
End Function
```

# **IsVersioningEnabled Property**

Returns a **Boolean** value that indicates whether the document library in which the active document is saved on the server is configured to create a backup copy, or version, each time the file is edited on the Web site. Read-only **Boolean**.

#### expression.IsVersioningEnabled()

*expression* Required. An expression that returns a **DocumentLibraryVersions** object.

## Remarks

Versioning is enabled or disabled on the document library and not on individual documents. Therefore the value of the **IsVersioningEnabled** property depends on the document library in which the document is saved.

The following example displays the number of saved versions of the active document, if versioning is enabled.

```
Dim dlvVersions As Office.DocumentLibraryVersions
Set dlvVersions = ActiveDocument.DocumentLibraryVersions
If dlvVersions.IsVersioningEnabled Then
    MsgBox "This document has " & dlvVersions.Count & _
    " saved versions.", vbInformation + vbOKOnly, _
    "Version Information"
Else
    MsgBox "Versioning is not enabled for this document.", _
    vbInformation + vbOKOnly, "No Versioning"
End If
Set dlvVersions = Nothing
```



# **Item Property**

Item property as it applies to the **AnswerWizardFiles** and **FoundFiles** objects.

Returns a file name string from an **AnswerWizardFiles** collection, or a file name from the list of file names represented by the **FoundFiles** object, respectively. Read-only **String**.

```
expression.Item(Index)
```

*expression* Required. An expression that returns one of the above objects.

*Index* Required **Long**. The index number of the Answer Wizard file name string, or the file name, to be returned.

Item property as it applies to the **Assistant**, **BalloonCheckbox**, **BalloonLabel**, and **FileDialog** objects.

Returns the text associated with an object. Read-only **String**.

### expression.Item

*expression* Required. An expression that returns one of the above objects.

Item property as it applies to the **BalloonCheckboxes** and **BalloonLabels** objects.

Returns a **BalloonCheckBox** or **BalloonLabel** object.

expression.Item(Index)

*expression* Required. An expression that returns one of the above objects.

*Index* Required **Long**. The index number of the check box or label to be returned.

Item property as it applies to the **CommandBars** object.

Returns a **CommandBar** object from the **CommandBars** collection.

## expression.Item(Index)

*expression* Required. An expression that returns a **CommandBars** object.

*Index* Required **Variant**. The name or index number of the object to be returned.

Item property as it applies to the **CommandBarControls** object.

Returns a **CommandBarControl** object from the **CommandBarControls** collection.

expression.Item(Index)

*expression* Required. An expression that returns a **CommandBarControls** object.

*Index* Required **Variant**. The name or index number of the object to be returned.

Item property as it applies to the **DocumentLibraryVersions** object.

Returns a **DocumentLibraryVersion** object from the **DocumentLibraryVersions** collection.

expression.Item(Index)

*expression* Required. An expression that returns a **DocumentLibraryVersions** object.

*Index* Required **Long**. The index number of the **DocumentLibraryVersion** returned.

Item property as it applies to the **DocumentProperties** object.

Returns a **DocumentProperty** object from the **DocumentProperties** collection.

## expression.Item(Index)

*expression* Required. An expression that returns a **DocumentProperties** object.

*Index* Required **Variant**. The name or index number of the document property returned.

Item property as it applies to the **FileTypes** collection.

Returns a value that indicates which file type will be searched for by the **Execute** method of the **FileSearch** object. Read-only **MsoFileType**.

MsoFileType can be one of these MsoFileType constants. **msoFileTypeAllFiles msoFileTypeBinders** msoFileTypeCalendarItem **msoFileTypeContactItem msoFileTypeDatabases msoFileTypeDataConnectionFiles msoFileTypeDesignerFiles** msoFileTypeDocumentImagingFiles msoFileTypeExcelWorkbooks **msoFileTypeJournalItem msoFileTypeMailItem msoFileTypeNoteItem** msoFileTypeOfficeFiles **msoFileTypeOutlookItems msoFileTypePhotoDrawFiles** msoFileTypePowerPointPresentations **msoFileTypeProjectFiles msoFileTypePublisherFiles msoFileTypeTaskItem** 

msoFileTypeTemplates msoFileTypeVisioFiles msoFileTypeWebPages msoFileTypeWordDocuments

expression.Item(Index)

*expression* Required. An expression that returns a **FileTypes** collection.

*Index* Required **Long**. The index number of the object to be returned.

Item property as it applies to the **MsoEnvelope** object.

Returns a **MailItem** object that can be used to send the document as an e-mail.

expression.Item

*expression* Required. An expression that returns an **MsoEnvelope** object.

Item property as it applies to the **Permission** object.

Returns a **UserPermission** object that is a member of the **Permission** collection. The **UserPermission** object associates a set of permissions on the active document with a single user and an optional expiration date.

expression.Item(Index)

*expression* Required. An item that returns a **Permission** object.

*Index* Required **Variant**. The numeric index of the **UserPermission** in the **Permission** collection, or the email address of the user whose set of permissions on the active document is to be returned.

Item property as it applies to the **PropertyTests** object.

Returns a **PropertyTest** object from the **PropertyTests** collection.

expression.Item(Index)

*expression* Required. An expression that returns a **PropertyTests** object.

*Index* Required **Long**. The index number of the property test to be returned.

Item property as it applies to the **ScopeFolders** and **SearchFolders** objects.

Returns a **ScopeFolder** object that represents a subfolder of the parent object.

expression.Item(Index)

*expression* Required. An expression that returns one of the above objects.

*Index* Required **Long**. Determines which subfolder to return.

Item property as it applies to the **SearchScopes** object.

Returns a **SearchScope** object that corresponds to an area in which to perform a file search, such as local drives or Microsoft Outlook folders.

## expression.Item(Index)

*expression* Required. An expression that returns a **SearchScopes** object.

*Index* Required **Long**. Determines which **SearchScope** object to return.

Item property as it applies to the **SharedWorkspaceFiles** object.

Returns a **SharedWorkspaceFile** object from the Files collection of the shared workspace.

expression.Item(Index)

*expression* Required. The specified **SharedWorkspaceFiles** collection.

*Index* Required **Long**. Returns the **SharedWorkspaceFile** at the position specified. *Index* does not correspond to the order in which the items are displayed in the Shared Workspace pane, and is not affected by re-sorting the display.

Item property as it applies to the **SharedWorkspaceFolders** object.

Returns a **SharedWorkspaceFolder** object from the Folders collection of the shared workspace.

## expression.Item(Index)

*expression* Required. The specified **SharedWorkspaceFolders** collection.

*Index* Required **Long**. Returns the **SharedWorkspaceFolder** at the position specified. *Index* does not correspond to the order in which the items are displayed in the Shared Workspace pane, and is not affected by re-sorting the display.

Item property as it applies to the **SharedWorkspaceLinks** object.

Returns a **SharedWorkspaceLink** object from the Links collection of the shared workspace.

## expression.Item(Index)

*expression* Required. The specified **SharedWorkspaceLinks** collection.

*Index* Required **Long**. Returns the **SharedWorkspaceLink** at the position specified. *Index* does not correspond to the order in which the items are displayed in the Shared Workspace pane, and is not affected by re-sorting the display.

Item property as it applies to the **SharedWorkspaceMembers** object.

Returns a **SharedWorkspaceMember** object from the Members collection of the shared workspace.

expression.Item(Index)

*expression* Required. The specified **SharedWorkspaceMembers** collection.

*Index* Required **Long**. Returns the **SharedWorkspaceMember** at the position specified. *Index* does not correspond to the order in which the items are

displayed in the Shared Workspace pane.

Item property as it applies to the **SharedWorkspaceTasks** object.

Returns a **SharedWorkspaceTask** object from the Tasks collection of the shared workspace.

## expression.Item(Index)

*expression* Required. The specified **SharedWorkspaceTasks** collection.

*Index* Required **Long**. Returns the **SharedWorkspaceTask** at the position specified. *Index* does not correspond to the order in which the items are displayed in the Shared Workspace pane, and is not affected by re-sorting the display.

Item property as it applies to the **SignatureSet** object.

Returns a **Signature** object that corresponds to one of the digital signatures with which the document is currently signed.

## expression.Item(iSig)

*expression* Required. An expression that returns a **SignatureSet** object.

*iSig* Required **Long**. Determines which **Signature** object to return.

Item property as it applies to the **WebPageFonts** object.

Returns a **WebPageFont** object from the **WebPageFonts** collection for a particular value of **MsoCharacterSet**.

```
expression.Item(Index)
```

*expression* Required. An expression that returns one of the above objects.

*Index* Required <u>MsoCharacterSet</u>. The specified character set.

MsoCharacterSet can be one of these MsoCharacterSet constants.

msoCharacterSetArabic msoCharacterSetCyrillic msoCharacterSetEnglishWesternEuropeanOtherLatinScript msoCharacterSetGreek msoCharacterSetHebrew msoCharacterSetJapanese msoCharacterSetJapanese msoCharacterSetKorean msoCharacterSetKorean msoCharacterSetThilingualUnicode msoCharacterSetSimplifiedChinese msoCharacterSetThai msoCharacterSetThai

As it applies to the **AnswerWizardFiles** and **FoundFiles** objects.

This example resets the file list for the current Answer Wizard and displays both the file count and the file names in a message box, using the **Item** property to return the file names.

```
Dim customAnswerWizard As AnswerWizard
Dim strFileList As String
Dim intCounter As Integer
Dim intNumFiles As Integer
Set customAnswerWizard = Application.AnswerWizard
intCounter = 1
customAnswerWizard.ResetFileList
strFileList = ""
intNumFiles = customAnswerWizard.Files.Count
For intCounter = 1 To (intNumFiles)
    strFileList = strFileList & _
    customAnswerWizard.Files.Item(intCounter) & Chr(13)
Next
MsgBox "There are " & customAnswerWizard.Files.Count & _
    " files avaialble through this AnswerWizard: " &
    Chr(13) & strFileList
```

As it applies to the **BalloonCheckboxes** and **BalloonLabels** objects.

**Item** is the default member of the object or collection. The following two statements both assign to lblText the text of the first label in the **Balloon** object assigned to myBalloon.

```
lblText = myBalloon.Labels(1).Item
lblText = myBalloon.Labels(1)
```

As it applies to the **CommandBars** object.

**Item** is the default member of the object or collection. The following two statements both assign a **CommandBar** object to cmdBar.

```
Set cmdBar = CommandBars.Item("Standard")
Set cmdBar = CommandBars("Standard")
```

As it applies to the **MsoEnvelope** object.

The following example sends the active Microsoft Word document as an e-mail to the e-mail address that you pass to the subroutine.

```
Sub SendMail(ByVal strRecipient As String)
    'Use a With...End With block to reference the msoEnvelope object
   With Application.ActiveDocument.MailEnvelope
        'Add some introductory text before the body of the e-mail me
        .Introduction = "Please read this and send me your comments.
        'Return a MailItem object that you can use to send the docum
        With .Item
            'All of the mail item settings are saved with the docume
            'When you add a recipient to the Recipients collection
            'or change other properties these settings will persist.
            .Recipients.Add strRecipient
            .Subject = "Here is the document."
            'The body of this message will be
            'the content of the active document.
            .Send
        End With
    End With
End Sub
```

As it applies to the **WebPageFonts** object.

The following example uses the **Item** property to set myFont to the **WebPageFont** object for the English/Western European/Other Latin Script character set in the active application.

```
Dim myFont As WebPageFont
Set myFont = _
Application.DefaultWebOptions.Fonts. _
Item(msoCharacterSetEnglishWesternEuropeanOtherLatinScript)
```

# ItemCountExceeded Property

Returns a **Boolean** value that indicates whether the number of items in the collection has exceeded the 99 that can be displayed in the **Shared Workspace** task pane. Read-only **Boolean**.

### expression.Connected

*expression* Required. An expression that returns one of the objects in the Applies To list.

## Remarks

The **Shared Workspace** task pane can only display 99 shared workspace files and folders, links, members, or tasks. If more than 99 items are added to any of these collections:

- the corresponding tab of the **Shared Workspace** task pane will stop displaying the list of items and will display a link to the shared workspace site web page instead;
- the collection will no longer be populated locally and its **Count** property will return 0 (zero).

Furthermore, once the **ItemCountExceeded** property returns **True** for one of the collections listed above, the developer can no longer remedy the situation programmatically by deleting items from the collection to reduce the count below 99, because the collection is no longer populated.

The **ItemCountExceeded** property of the **SharedWorkspaceFiles** and the **SharedWorkspaceFolders** collections returns **True** when the combined **Count** of files and folders exceeds 99, since both lists are combined and displayed together on the **Documents** tab of the **Shared Workspace** task pane.

The following example checks the **Count** property of the **SharedWorkspaceLinks** collection. If **Count** returns 0 (zero), it checks the **ItemCountExceeded** property to determine whether in fact the shared workspace has no saved links, or whether it has more than 99 and the links collection has been cleared.

```
ActiveWorkbook.SharedWorkspace.Refresh
If ActiveWorkbook.SharedWorkspace.Links.Count = 0 Then
    If ActiveWorkbook.SharedWorkspace.Links.ItemCountExceeded Th
        MsgBox "More than 99 links in shared workspace.", _
            vbInformation + vbOKOnly, "Item Count Exceeded"
    Else
        MsgBox "No links in shared workspace.", _
        vbInformation + vbOKOnly, "No Links"
    End If
End If
```

# **KeyboardShortcutTips Property**

Some of the content in this topic may not be applicable to some languages.

**True** if the Office Assistant displays Help about keyboard shortcuts. Read/write **Boolean**.

## Remarks

The **KeyboardShortcutTips** property corresponds to the **Keyboard shortcuts** option in the **Show tips about** section on the **Options** tab in the **Office Assistant** dialog box.

This example sets the Office Assistant to provide Help information about keyboard shortcuts.

Assistant.KeyboardShortcutTips = True

# **Labels Property**

Some of the content in this topic may not be applicable to some languages.

Returns a **BalloonLabels** collection that represents the button labels, number labels, and bullet labels contained in the specified Office Assistant balloon. Read-only.

For information about returning a single member of a collection, see <u>Returning</u> <u>an Object from a Collection</u>.

This example creates a balloon containing three choices. The variable x is set to the return value of the **Show** method, which will be 1, 2 or 3, depending on the label the user clicks. In the example, a message box displays the value of the variable x, but you can pass the value to another procedure or you can use the value in a **Select Case** statement.

```
Set b = Assistant.NewBalloon
With b
    .Heading = "This is my heading"
    .Text = "Select one of these things:"
    .Labels(1).Text = "Choice One"
    .Labels(2).Text = "Choice Two"
    .Labels(3).Text = "Choice Three"
    x = .Show
End With
MsgBox x
```



# Language Property

Specifies or returns the scripting language of the active script. Read/write **MsoScriptLanguage**.

MsoScriptLanguage can be one of these MsoScriptLanguage constants. msoScriptLanguageASP msoScriptLanguageJava msoScriptLanguageOther msoScriptLanguageVisualBasic

## Remarks

The **MsoScriptLanguage** constants used with the **Language** property are also used in the **Language** parameter in the **Add** method of the **Scripts** collection.

This example checks the **Language** property to ensure that the first script in worksheet one in the active workbook is written in VBScript.

```
If ActiveWorkbook.Worksheets(1).Scripts(1).Language <> _
    msoScriptLanguageVisualBasic Then
    MsgBox "Language is not set to VBScript."
End If
```



# LanguageID Property

Returns the locale identifier (LCID) for the install language, the user interface language, or the Help language. Read-only **Long**.

## expression.LanguageID(Id)

*expression* Required. An expression that returns a **LanguageSettings** object.

Id Required MsoAppLanguageID.

#### MsoAppLanguageID constant

msoLanguageIDExeMode

### Meaning

The language mode that the application is using. This setting applies only to Microsoft Excel and Microsoft Access. This setting affects languages that can be displayed and edited, available language-specific features, number styles, currency settings, and so forth.

If none of the languages supported by Excel and Access are used, the host application will not be configured to support right-to-left and East Asian languages. The supported languages are as follows:

- Arabic
- Farsi
- Hebrew
- Japanese
- Korean
- Simplified Chinese
- Traditional Chinese
- Urdu
- Yiddish

### msoLanguageIDHelp

The language used for online Help.

	The language settings used by Microsoft Office to
	set up defaults. For example, Microsoft Word
msoLanguageIDInstall	uses this setting to determine the layout of
	toolbars and the default types of bullets and
	numbers on a language-by-language basis.
msoLanguageIDUI	The language used by the host application's user interface.
msoLanguageIDUIPrevious	The language setting for the user interface when a given computer was last rebooted. A program or add-in can use this to determine whether the user interface language has changed.

This Microsoft Excel example checks the **LanguageID** property settings for the user interface and execution mode to verify that they are set to the same LCID. The example returns an error if there is a discrepancy.

If Application.LanguageSettings.LanguageID(msoLanguageIDExeMode) \_
 <> Application.LanguageSettings.LanguageID(msoLanguageIDUI) \_
 Then MsgBox "The user interface language and execution " & \_
 "mode are different."



# LanguagePreferredForEditing Property

Returns **True** if the value for the **msoLanguageID** constant has been identified in the Windows registry as a preferred language for editing. Read-only **Boolean**.

expression.LanguagePreferredForEditing(lid)

*expression* Required. An expression that returns one of the objects in the Applies To list.

*lid* Required <u>MsoLanguageID</u>.

MsoLanguageID can be one of these MsoLanguageID constants.

msoLanguageIDAfrikaans msoLanguageIDAlbanian msoLanguageIDAmharic msoLanguageIDArabic msoLanguageIDArabicAlgeria msoLanguageIDArabicBahrain msoLanguageIDArabicEgypt msoLanguageIDArabicIraq msoLanguageIDArabicJordan msoLanguageIDArabicKuwait msoLanguageIDArabicLebanon msoLanguageIDArabicLibya msoLanguageIDArabicMorocco msoLanguageIDArabicOman msoLanguageIDArabicQatar msoLanguageIDArabicSyria msoLanguageIDArabicTunisia **msoLanguageIDArabicUAE** 

msoLanguageIDArabicYemen **msoLanguageIDArmenian** msoLanguageIDAssamese msoLanguageIDAzeriCyrillic msoLanguageIDAzeriLatin **msoLanguageIDBasque** msoLanguageIDBelgianDutch msoLanguageIDBelgianFrench **msoLanguageIDBengali** msoLanguageIDBrazilianPortuguese **msoLanguageIDBulgarian** msoLanguageIDBurmese msoLanguageIDByelorussian **msoLanguageIDCatalan** msoLanguageIDCherokee msoLanguageIDChineseHongKong msoLanguageIDChineseMacao msoLanguageIDChineseSingapore **msoLanguageIDCroatian** msoLanguageIDCzech msoLanguageIDDanish msoLanguageIDDutch msoLanguageIDEnglishAUS msoLanguageIDEnglishBelize msoLanguageIDEnglishCanadian msoLanguageIDEnglishCaribbean msoLanguageIDEnglishIreland msoLanguageIDEnglishJamaica msoLanguageIDEnglishNewZealand msoLanguageIDEnglishPhilippines msoLanguageIDEnglishSouthAfrica msoLanguageIDEnglishTrinidad msoLanguageIDEnglishUK

msoLanguageIDEnglishUS msoLanguageIDEnglishZimbabwe **msoLanguageIDEstonian** msoLanguageIDFaeroese msoLanguageIDFarsi **msoLanguageIDFinnish** msoLanguageIDFrench msoLanguageIDFrenchCameroon msoLanguageIDFrenchCanadian msoLanguageIDFrenchCotedIvoire msoLanguageIDFrenchLuxembourg msoLanguageIDFrenchMali msoLanguageIDFrenchMonaco msoLanguageIDFrenchReunion msoLanguageIDFrenchSenegal msoLanguageIDFrenchWestIndies msoLanguageIDFrenchZaire msoLanguageIDFrisianNetherlands msoLanguageIDGaelicIreland msoLanguageIDGaelicScotland msoLanguageIDGalician **msoLanguageIDGeorgian** msoLanguageIDGerman msoLanguageIDGermanAustria msoLanguageIDGermanLiechtenstein msoLanguageIDGermanLuxembourg **msoLanguageIDGreek msoLanguageIDGujarati msoLanguageIDHebrew msoLanguageIDHindi** msoLanguageIDHungarian msoLanguageIDIcelandic msoLanguageIDIndonesian

msoLanguageIDInuktitut **msoLanguageIDItalian** msoLanguageIDJapanese msoLanguageIDKannada msoLanguageIDKashmiri **msoLanguageIDKazakh** msoLanguageIDKhmer **msoLanguageIDKirghiz** msoLanguageIDKonkani msoLanguageIDKorean **msoLanguageIDLao msoLanguageIDLatvian** msoLanguageIDLithuanian msoLanguageIDMacedonian msoLanguageIDMalayalam msoLanguageIDMalayBruneiDarussalam msoLanguageIDMalaysian msoLanguageIDMaltese msoLanguageIDManipuri msoLanguageIDMarathi msoLanguageIDMexicanSpanish msoLanguageIDMixed msoLanguageIDMongolian msoLanguageIDNepali msoLanguageIDNone msoLanguageIDNoProofing msoLanguageIDNorwegianBokmol msoLanguageIDNorwegianNynorsk msoLanguageIDOriya msoLanguageIDOromo **msoLanguageIDPolish** msoLanguageIDPortuguese msoLanguageIDPunjabi

msoLanguageIDRhaetoRomanic **msoLanguageIDRomanian** msoLanguageIDRomanianMoldova **msoLanguageIDRussian** msoLanguageIDRussianMoldova msoLanguageIDSamiLappish msoLanguageIDSanskrit msoLanguageIDSerbianCyrillic msoLanguageIDSerbianLatin msoLanguageIDSesotho msoLanguageIDSimplifiedChinese msoLanguageIDSindhi msoLanguageIDSlovak msoLanguageIDSlovenian msoLanguageIDSorbian **msoLanguageIDSpanish** msoLanguageIDSpanishArgentina msoLanguageIDSpanishBolivia msoLanguageIDSpanishChile msoLanguageIDSpanishColombia msoLanguageIDSpanishCostaRica msoLanguageIDSpanishDominicanRepublic msoLanguageIDSpanishEcuador msoLanguageIDSpanishElSalvador msoLanguageIDSpanishGuatemala msoLanguageIDSpanishHonduras msoLanguageIDSpanishModernSort msoLanguageIDSpanishNicaragua msoLanguageIDSpanishPanama msoLanguageIDSpanishParaguay msoLanguageIDSpanishPeru msoLanguageIDSpanishPuertoRico msoLanguageIDSpanishUruguay

msoLanguageIDSpanishVenezuela **msoLanguageIDSutu** msoLanguageIDSwahili msoLanguageIDSwedish msoLanguageIDSwedishFinland msoLanguageIDSwissFrench msoLanguageIDSwissGerman msoLanguageIDSwissItalian msoLanguageIDTajik msoLanguageIDTamil **msoLanguageIDTatar msoLanguageIDTelugu msoLanguageIDThai msoLanguageIDTibetan** msoLanguageIDTigrignaEritrea msoLanguageIDTigrignaEthiopic msoLanguageIDTraditionalChinese msoLanguageIDTsonga msoLanguageIDTswana **msoLanguageIDTurkish** msoLanguageIDTurkmen msoLanguageIDUkrainian msoLanguageIDUrdu msoLanguageIDUzbekCyrillic msoLanguageIDUzbekLatin msoLanguageIDVenda msoLanguageIDVietnamese msoLanguageIDWelsh msoLanguageIDXhosa msoLanguageIDZulu

# Remarks

You must test all valid **msoLanguageID** values to enumerate the set of preferred languages.

This example displays a message if U.S. English is a preferred editing language.

```
If Application.LanguageSettings. _
LanguagePreferredForEditing(msoLanguageIDEnglishUS) Then
MsgBox "One of the preferred editing languages is US English."
End If
```

### **LargeButtons Property**

**True** if the toolbar buttons displayed are larger than normal size. Read/write **Boolean**.

This example toggles the display size of toolbar buttons on all command bars.

```
Set allBars = CommandBars
If allBars.LargeButtons Then
        allBars.LargeButtons = False
Else
        allBars.LargeButtons = True
End If
```



### **LastModified Property**

Returns or sets a constant that represents the amount of time since the specified file was last modified and saved. The default value is **msoLastModifiedAnyTime**. Read/write **MsoLastModified**.

MsoLastModified can be one of these MsoLastModified constants. msoLastModifiedAnyTime msoLastModifiedLastMonth msoLastModifiedThisMonth msoLastModifiedThisWeek msoLastModifiedToday msoLastModifiedYesterday

This example sets options for a file search. The files this search returns were modified yesterday and are located in the C:\My Documents folder or in one of its subfolders.

```
Set fs = Application.FileSearch
With fs
    .LookIn = "C:\My Documents"
    .SearchSubFolders = True
    .LastModified = msoLastModifiedYesterday
End With
```

### LastRefreshed Property

Returns the date and time when the **Refresh** method was most recently called. Read-only **Variant**.

expression.LastRefreshed

*expression* Required. An expression that returns a **SharedWorkspace** object.

The **LastRefreshed** property raises an error if the **Refresh** method has never been called.

The following example refreshes the shared workspace if it has not been refreshed in the last 3 minutes. The example also handles the error condition where the workspace has not yet been refreshed.

err\_NeverRefreshed:

ActiveWorkbook.SharedWorkspace.Refresh

### LastSyncTime Property

Returns the date and time when the local copy of the active document was last synchronized with the server copy. Read-only **Variant**.

expression.LastSyncTime

*expression* Required. An expression that returns a **Sync** object.

Use the **LastSyncTime** property to determine how much time has elapsed since the local copy of the active document was last synchronized with the server copy. Check the **Status** property to determine whether the local copy and the server copy are out of sync.

If the active document is not configured for synchronization between the local copy and the server copy, the **LastSyncTime** property raises a run-time error.

The following example alerts the user and displays the sync status if more than 24 hours have elapsed since the **LastSyncTime**.

```
Dim objSync As Office.Sync
Dim dtmLastSync As Date
Dim strStatus As String
Set objSync = ActiveDocument.Sync
dtmLastSync = CDate(objSync.LastSyncTime)
If DateDiff("h", dtmLastSync, Now) > 24 Then
strStatus = "Document has not been synced " & _
" within the last 24 hours." & vbCrLf & _
"Document status: " & objSync.Status
MsgBox strStatus, vbInformation + vbOKOnly, "Error Informati
End If
Set objSync = Nothing
```



# **Left Property**

Some of the content in this topic may not be applicable to some languages.

Left property as it applies to the Assistant and CommandBar objects.

Sets or returns the horizontal position of the Office Assistant window (in points), or the distance (in pixels) of the <u>command bar</u>, from the left edge of the specified object relative to the screen. Read/write **Long**.

#### expression.Left

*expression* Required. An expression that returns one of the above objects.

Left property as it applies to the **CommandBarButton**, **CommandBarComboBox**, **CommandBarControl**, and **CommandBarPopup** objects.

Set or returns the horizontal position of the specified <u>command bar control</u> (in pixels) relative to the left edge of the screen. Returns the distance from the left side of the docking area. Read-only **Long**.

#### expression.Left

*expression* Required. An expression that returns one of the above objects.

As it applies to the **Assistant** and **CommandBar** objects.

This example displays the Office Assistant and moves it to the specified position within the application window.

```
With Assistant
.Visible = True
.Left = 300
.Top = 300
End With
```

This example moves the command bar named Custom from its docked position along the top of the window to the left edge of the window.

```
Set myBar = CommandBars("Custom")
With myBar
    .Position = 1
    .RowIndex = 2
    .Left = 0
End With
```

# **Links Property**

Returns a **SharedWorkspaceLinks** collection that represents the list of links saved in the current shared workspace. Read-only.

expression.Links()

*expression* Required. An expression that returns a **SharedWorkspace** object.

The following example lists the links saved in the current shared workspace.

```
Dim swsLinks As Office.SharedWorkspaceLinks
  Set swsLinks = ActiveWorkbook.SharedWorkspace.Links
  MsgBox "There are " & swsLinks.Count & _
        " link(s) in the current shared workspace.", _
        vbInformation + vbOKOnly, _
        "Collection Information"
   Set swsLinks = Nothing
```

# **LinkSource Property**

Returns or sets the source of a linked custom document property. Read/write **String**.

This property applies only to custom document properties; you cannot use it with built-in document properties.

The source of the specified link is defined by the container application.

Setting the **LinkSource** property sets the **LinkToContent** property to **True**.

This example displays the linked status of a custom document property. For the example to work, dp must be a valid **DocumentProperty** object.

```
Sub DisplayLinkStatus(dp As DocumentProperty)
Dim stat As String, tf As String
If dp.LinkToContent Then
    tf = ""
Else
    tf = "not "
End If
stat = "This property is " & tf & "linked"
If dp.LinkToContent Then
    stat = stat + Chr(13) & "The link source is " & dp.LinkSourc
End If
MsgBox stat
End Sub
```

# LinkToContent Property

**True** if the value of the custom document property is linked to the content of the container document. **False** if the value is static. Read/write **Boolean**.

This property applies only to custom document properties. For built-in document properties, the value of this property is **False**.

Use the **LinkSource** property to set the source for the specified linked property. Setting the **LinkSource** property sets the **LinkToContent** property to **True**.

This example displays the linked status of the custom document property. For the example to work, dp must be a valid **DocumentProperty** object.

```
Sub DisplayLinkStatus(dp As DocumentProperty)
Dim stat As String, tf As String
If dp.LinkToContent Then
    tf = ""
Else
    tf = "not "
End If
stat = "This property is " & tf & "linked"
If dp.LinkToContent Then
    stat = stat + Chr(13) & "The link source is " & dp.LinkSourc
End If
MsgBox stat
End Sub
```



# **List Property**

Returns or sets an item in the command bar <u>combo box control</u>. Read/write **String**.

**Note** This property is read-only for built-in combo box controls.

#### expression.List(Index)

*expression* Required. An expression that returns a **CommandBarComboBox** object.

*Index* Required **Long**. The list item to be set.

This example checks the fourth list item in the combo box control whose caption is "Stock Data" on the command bar named "Custom." If the item isn't "View News," the example displays a message advising the user that the combo box may be damaged and asks the user to reinstall the application.

```
Set myBar = CommandBars
    .Add(Name:="Custom", Position:=msoBarTop, _
    Temporary:=True)
With myBar
    .Controls.Add Type:=msoControlComboBox, ID:=1
    .Visible = True
End With
With CommandBars("Custom").Controls(1)
    .AddItem "Get Stock Quote", 1
    .AddItem "View Chart", 2
    .AddItem "View Fundamentals", 3
    .AddItem "View News", 4
    .Caption = "Stock Data"
    .DescriptionText = "View Data For Stock"
End With
If CommandBars("Custom").Controls(1).List(4) _
     <> "View News" Then
MsgBox ("Stock Data appears to be damaged." & _
     " Please reinstall application.")
End If
```



# **ListCount Property**

Returns the number of list items in a command bar <u>combo box control</u>. Readonly **Long**.

This example checks the number of items in the combo box on the command bar named "Custom." If there aren't three items in the list that the procedure produces, the example displays a message advising the user that the combo box may be damaged and asks the user to reinstall the application.

```
Set myBar = CommandBars
    .Add(Name:="Custom", Position:=msoBarTop, _
    Temporary:=True)
With myBar
    .Controls.Add Type:=msoControlComboBox, ID:=1
    .Visible = True
End With
With CommandBars("Custom").Controls(1)
    .AddItem "Get Stock Quote", 1
    .AddItem "View Chart", 2
    .AddItem "View Fundamentals", 3
    .Caption = "Stock Data"
    .DescriptionText = "View Data For Stock"
End With
If CommandBars("Custom").Controls(1).ListCount _
     <> 4 Then
MsgBox ("ComboBox appears to be damaged." & _
     " Please reinstall.")
End If
```



## ListHeaderCount Property

Returns or sets the number of list items in a command bar <u>combo box control</u> that appears above the separator line. Read/write **Long**.

**Note** This property is read-only for built-in combo box controls.

A **ListHeaderCount** property value of – 1 indicates that there's no separator line in the combo box control.

This example adds a combo box control to the command bar named "Custom" and then adds two items to the combo box. The example uses the **ListHeaderCount** property to display a separator line between First Item and Second Item in the combo box. The example also sets the number of line items, the width of the combo box, and an empty default for the combo box.

```
Set myBar = CommandBars("Custom")
Set myControl = myBar.Controls.Add(Type:=msoControlComboBox)
With myControl
    .AddItem Text:="First Item", Index:=1
    .AddItem Text:="Second Item", Index:=2
    .DropDownLines = 3
    .DropDownWidth = 75
    .ListHeaderCount = 1
End With
```



# **ListIndex Property**

Returns or sets the index number of the selected item in the list portion of the command bar <u>combo box control</u>. If nothing is selected in the list, this property returns zero. Read/write **Long**.

**Note** This property fails when applied to controls other than list controls.

Setting the **ListIndex** property causes the specified control to select the given item and execute the appropriate action in the application.

This example uses the **ListIndex** property to determine the correct subroutine to run, based on the selection in the combo box on the command bar named "My Custom Bar." Because the procedure uses **ListIndex**, the text in the combo box can be anything.

```
Sub processSelection()
Dim userChoice As Long
userChoice = CommandBars("My Custom Bar").Controls(1).ListIndex
Select Case userChoice
    Case 1
        chartcourse
    Case 2
        displaygraph
    Case Else
        MsgBox ("Invalid choice. Please choose again.")
    End Select
End Sub
```



# **Location Property**

Returns the location of the script anchor in the specified HTML document. Read-only **MsoScriptLocation**.

MsoScriptLocation can be one of these MsoScriptLocation constants. msoScriptLocationInBody msoScriptLocationInHead

Script tags in an HTML document can appear anywhere between the <HTML> tags in the document. In Microsoft Word, Excel, or PowerPoint, only the script anchors located between the <BODY> tags are visible. Additional HTML script that appears before or after the <BODY> tags is stored in but isn't visible to the user.

The **Scripts** collection contains all of the script anchors that appear in the document, whether inside or outside of the main body of the document. Using the **Location** argument of the **Add** method, you can insert script anchors within the <HEAD> and <BODY> tags in the HTML document. You can also use the **Location** property to determine where a particular script anchor is stored within the document.

This example checks the **Location** property of the first script in worksheet one in the active workbook and displays the location in a message box.

```
If ActiveWorkbook.Worksheets(1).Scripts(1).Location = 1 Then
    MsgBox "The script in located in the header."
End If
If ActiveWorkbook.Worksheets(1).Scripts(1).Location = 2 Then
    MsgBox "The script in located in the body of the worksheet."
End If
```

## **LookIn Property**

Returns or sets the folder to be searched during the specified file search. Read/write **String**.

This example searches the My Documents folders for all files that begin with "Cmd" and displays the name and location of each file that's found.

```
Set fs = Application.FileSearch
With fs
.LookIn = "C:\My Documents"
.FileName = "cmd*.*"
If .Execute > 0 Then
    MsgBox "There were " & .FoundFiles.Count & _
    " file(s) found."
    For i = 1 To .FoundFiles.Count
        MsgBox .FoundFiles(i)
        Next i
    Else
        MsgBox "There were no files found."
    End If
End With
```

## **Mask Property**

Returns an **IPictureDisp** object representing the mask image of a **CommandBarButton** object. The mask image determines what parts of the button image are transparent.

expression.Mask

*expression* Required. An expression that returns a **CommandBarButton** object.

When you create an image that you plan on using as a mask image, all of the areas that you want to be transparent should be white, and all of the areas that you want to show should be black.

Always set the mask after you have set the picture for a **CommandBarButton** object.

The following example sets the image and mask of the first

**CommandBarButton** that the code returns. To make this work, create a mask image and a button image and sustitute the paths in the sample with the paths to your images.

```
Sub ChangeButtonImage()
   Dim picPicture As IPictureDisp
   Dim picMask As IPictureDisp
   Set picPicture = stdole.StdFunctions.LoadPicture( _
       "c:\images\picture.bmp")
   "c:\images\mask.bmp")
    'Reference the first button on the first command bar
   'using a With...End With block.
   With Application.CommandBars.FindControl(msoControlButton)
       'Change the button image.
       .Picture = picPicture
       'Use the second image to define the area of the
       'button that should be transparent.
       .Mask = picMask
   End With
End Sub
```

The following example gets the image and mask of the first

**CommandBarButton** that the code returns and outputs each of them to a file. To make this work, specify a path for the output files.

```
Sub GetButtonImageAndMask()
   Dim picPicture As IPictureDisp
   Dim picMask As IPictureDisp
   With Application.CommandBars.FindControl(msoControlButton)
      'Get the button image and mask of this CommandBarButton obje
      Set picPicture = .Picture
      Set picMask = .Mask
End With
   'Save the button image and mask in a folder.
   stdole.SavePicture picPicture, "c:\image.bmp"
```

stdole.SavePicture picMask, "c:\mask.bmp"
End Sub

# MatchAllWordForms Property

Some of the content in this topic may not be applicable to some languages.

**True** if the file search is expanded to include all forms of the specified word contained in the body of the file, or in the file's properties. Read/write **Boolean**.

This property is available only if the file Mswds\_en.lex has been installed and registered. Note that this file isn't installed as part of a Typical setup.

This example returns all files that contain the word "run," "running," "runs," or "ran" in the body of the file, or in the properties of the file. The **TextOrProperty** property sets the word to be matched, and limits the search to either the body of the file or the file properties.

```
With Application.FileSearch
    .NewSearch
    .LookIn = "C:\My Documents"
    .SearchSubFolders = True
    .TextOrProperty = "run"
    .MatchAllWordForms = True
    .FileType = msoFileTypeAllFiles
End With
```

# MatchTextExactly Property

Some of the content in this topic may not be applicable to some languages.

**True** if the specified file search will find only files whose body text or file properties contain the exact word or phrase that you've specified. Read/write **Boolean**.

This example searches the C:\My Documents folder and returns all files that contain the word "Run" either in the body text or in the file properties.

```
With Application.FileSearch
    .NewSearch
    .LookIn = "C:\My Documents"
    .TextOrProperty = "Run"
    .MatchTextExactly = True
    .FileType = msoFileTypeAllFiles
End With
```

## **Members Property**

Returns a **SharedWorkspaceMembers** collection that represents the list of members in the current shared workspace. Read-only.

expression.Members()

*expression* Required. An expression that returns a **SharedWorkspace** object.

The following example lists the members in the current shared workspace.

```
Dim swsMembers As Office.SharedWorkspaceMembers
Set swsMembers = ActiveWorkbook.SharedWorkspace.Members
MsgBox "There are " & swsMembers.Count & ______"
member(s) in the current shared workspace.", ______
vbInformation + vbOKOnly, ______"
Collection Information"
Set swsMembers = Nothing
```



# MenuAnimationStyle Property

Returns or sets the way a <u>command bar</u> is animated. Read/write <u>MsoMenuAnimation</u>.

MsoMenuAnimation can be one of these MsoMenuAnimation constants. msoMenuAnimationNone msoMenuAnimationRandom msoMenuAnimationSlide msoMenuAnimationUnfold

This example sets options for all command bars in Microsoft Office.

```
With CommandBars
    .LargeButtons = True
    .DisplayTooltips = True
    .DisplayKeysInTooltips = True
    .MenuAnimationStyle = msoMenuAnimationUnfold
End With
```



## **Mode Property**

Some of the content in this topic may not be applicable to some languages.

Returns or sets the modal behavior of the Office Assistant balloon. When you create a **Balloon** object, this property is initially set to **msoModeModal**. Read/write **MsoModeType**.

MsoModeType can be one of these MsoModeType constants. msoModeAutoDown msoModeModal msoModeModeless

If the **Mode** property for a balloon is set to **msoModeModeless**, the user can work in the application while the balloon is visible. If the property is set to **msoModeModal**, the user must dismiss the balloon before continuing to work in the application. If the property is set to **msoModeAutoDown**, the balloon is dismissed when the user clicks anywhere on the screen.

If the **Mode** property for a balloon is set to **msoModeModeless** and a value for the **Callback** property is not supplied, an error occurs. The **Close** method can only be used in the procedure specified by the **Callback** property when the **Mode** property is set to **msoModeModeless**.

This example creates a balloon with an alert icon that instructs the user to select a printer. Because the balloon is modeless, the user has access to printer commands while the balloon is visible. When the user clicks the **OK** button, the procedure specified in the **Callback** property is run.

```
With Assistant.NewBalloon
    .Heading = "Select A Printer"
    .Text = "You must select a printer before printing."
    .Icon = msoIconAlert
    .CheckBoxes(1).Text = "Local printer"
    .CheckBoxes(2).Text = "Network printer"
    .Mode = msoModeModeless
    .Callback = "ProcessPrinter"
    .Show
End With
```

# **Modified Property**

Returns the date and time at which the specified version of the shared document was last saved to the server. Read-only **Variant**.

#### expression.Modified

*expression* Required. An expression that returns a **DocumentLibraryVersion** object.

A new version is created on the server each time a user opens the document and is updated when the user saves changes; additional versions are not created each time the user saves changes to the open document. The **Modified** property of the active document version represents the last time the user saved changes to the open document.

The following example displays the **Modified** date and time along with other properties of each version of a shared document.

```
Dim dlvVersions As Office.DocumentLibraryVersions
Dim dlvVersion As Office.DocumentLibraryVersion
Dim strVersionInfo As String
Set dlvVersions = ActiveDocument.DocumentLibraryVersions
If dlvVersions.IsVersioningEnabled Then
    strVersionInfo = "This document has " & _
        dlvVersions.Count & " versions: " & vbCrLf
    For Each dlvVersion In dlvVersions
        strVersionInfo = strVersionInfo &
             " - Version #: " & dlvVersion.Index & vbCrLf & _

    Modified by: " & dlvVersion.ModifiedBy & vbCrLf
    Modified on: " & dlvVersion.Modified & vbCrLf &

             ш
             п
               - Comments: " & dlvVersion.Comments & vbCrLf
             н
    Next
Else
    strVersionInfo = "Versioning not enabled for this document."
End If
MsgBox strVersionInfo, vbInformation + vbOKOnly, "Version Inform
Set dlvVersion = Nothing
Set dlvVersions = Nothing
```



# **ModifiedBy Property**

As it applies to the **DocumentLibraryVersion** object.

Returns the name of the user who last saved the specified version of the shared document to the server. Read-only **String**.

#### expression.ModifiedBy

*expression* Required. An expression that returns a **DocumentLibraryVersion** object.

As it applies to the **SharedWorkspaceFile**, **SharedWorkspaceLink**, and **SharedWorkspaceTask** objects.

Returns the name of the user who last modified the object. Read-only **String**.

#### expression.ModifiedBy

*expression* Required. An expression that returns a **SharedWorkspaceFile**, **SharedWorkspaceLink**, or **SharedWorkspaceTask** object.

For shared workspace objects, the **ModifiedBy** property returns the friendly name stored in the **Name** property of the **DWSMember** object.

The **SharedWorkspaceFolder** and **SharedWorkspaceMember** objects do not have a **ModifiedBy** property.

As it applies to the **DocumentLibraryVersion** object.

The following example displays the **ModifiedBy** name along with other properties of each version of a shared document.

```
Dim dlvVersions As Office.DocumentLibraryVersions
Dim dlvVersion As Office.DocumentLibraryVersion
Dim strVersionInfo As String
Set dlvVersions = ActiveDocument.DocumentLibraryVersions
If dlvVersions.IsVersioningEnabled Then
    strVersionInfo = "This document has " & _
        dlvVersions.Count & " versions: " & vbCrLf
    For Each dlvVersion In dlvVersions
        strVersionInfo = strVersionInfo & _
             " - Version #: " & dlvVersion.Index & vbCrLf & _
             " - Modified by: " & dlvVersion.ModifiedBy & vbCrLf
" - Modified on: " & dlvVersion.Modified & vbCrLf &
             " - Comments: " & dlvVersion.Comments & vbCrLf
    Next
Else
    strVersionInfo = "Versioning not enabled for this document."
End If
MsgBox strVersionInfo, vbInformation + vbOKOnly, "Version Inform
Set dlvVersion = Nothing
Set dlvVersions = Nothing
```

As it applies to the **SharedWorkspaceFile**, **SharedWorkspaceLink**, and **SharedWorkspaceTask** objects.

The following example lists the files in a shared workspace that were last modified by users other than the creator of the workspace.

```
vbInformation + vbOKOnly, "Files Modified by Other Users"
Set swsOwner = Nothing
Set swsFile = Nothing
```

# **ModifiedDate Property**

Returns the date and time when the shared workspace object was last modified. Read-only **Variant**.

#### expression.ModifiedDate

*expression* Required. An expression that returns one of the objects in the Applies To list.

The **SharedWorkspaceFolder** and **SharedWorkspaceMember** objects do not have a **ModifiedDate** property.

The following example returns a list of shared workspace files whose date and time last modified is earlier than today.

```
Dim swsFile As Office.SharedWorkspaceFile
Dim dtmMidnight As Date
Dim dtmFileDate As Date
Dim strOlderFiles As String
dtmMidnight = CDate(FormatDateTime(Now, vbShortDate) & " 12:00:6
For Each swsFile In ActiveWorkbook.SharedWorkspace.Files
    dtmFileDate = swsFile.ModifiedDate
    If dtmFileDate < dtmMidnight Then
        strOlderFiles = strOlderFiles & swsFile.URL & vbCrLf
    End If
Next
MsgBox "Files not modified today: " & vbCrLf & strOlderFiles, _
    vbInformation + vbOKOnly, "Older Files"
Set swsFile = Nothing
```

# **MouseTips Property**

Some of the content in this topic may not be applicable to some languages.

**True** if the Office Assistant provides suggestions for using the mouse effectively. Read/write **Boolean**.

The **MouseTips** property corresponds to the **Using the mouse more effectively** option under **Show tips about** on the **Options** tab in the **Office Assistant** dialog box.

This example sets the Office Assistant to provide suggestions for using the mouse effectively.

Assistant.MouseTips = True

# MoveWhenInTheWay Property

Some of the content in this topic may not be applicable to some languages.

**True** if the Office Assistant window automatically moves when it's in the way of the user's work area. For example, the Assistant will move if it's in the way of dragging or dropping or in the way of keystroke entries. Read/write **Boolean**.

The default value is **True**. The **MoveWhenInTheWay** property corresponds to the **Move when in the way** option in the Use the **Office Assistant** section on the **Options** tab in the **Office Assistant** dialog box.

This example displays the Office Assistant in a specific location and it sets several options before making the Assistant visible and active.

```
With Assistant
   .On = True
   .Visible = True
   .Left = 400
   If Not MoveWhenInTheWay Then MoveWhenInTheWay = True
   .Animation = msoAnimationGetAttentionMajor
End With
```



### **Name Property**

Returns or sets the name of the specified object. Read/write **String** for the **CommandBar**, **DocumentProperty**, and **SharedWorkspace** objects; read-only **String** for all other objects.

Name property as it applies to the **CommandBar** object.

The local name of a built-in <u>command bar</u> is displayed in the title bar (when the command bar isn't docked) and in the list of available command bars— wherever that list is displayed in the container application.

For a built-in command bar, the **Name** property returns the command bar's U.S. English name. Use the **NameLocal** property to return the localized name.

If you change the value of the **LocalName** property for a custom command bar, the value of **Name** changes as well, and vice versa.

Name property as it applies to the **SharedWorkspaceMember** object.

The **Name** property of the **SharedWorkspaceMember** object returns the friendly name for the shared workspace member, as specified in the *DisplayName* argument of the **Add** method of the **SharedWorkspaceMembers** collection. The **SharedWorkspaceMember** object has other properties that return the member's **DomainName** or **Email**.

As it applies to the **CommandBar** object.

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
```

As it applies to the **DocumentProperty** object.

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



# **NameLocal Property**

Returns the name of a built-in <u>command bar</u> as it's displayed in the language version of the container application, or returns or sets the name of a custom command bar. Read/write **String**.

**Note** If you attempt to set this property for a built-in command bar, an error occurs.

The local name of a built-in command bar is displayed in the title bar (when the command bar isn't docked) and in the list of available command bars, wherever that list is displayed in the container application.

If you change the value of the **LocalName** property for a custom command bar, the value of **Name** changes as well, and vice versa.

This example displays the name and localized name of the first command bar in the container application.

With CommandBars(1) MsgBox "The name of the command bar is " & .Name MsgBox "The localized name of the command bar is " & .**NameLocal** End With

### **NewBalloon Property**

Some of the content in this topic may not be applicable to some languages.

Creates an Office Assistant balloon. Returns a **<u>Balloon</u>** object. Read-only.

This example creates a balloon with a heading, text, and three region choices, and then displays it.

```
With Assistant.NewBalloon
    .Heading = "Regional Sales Data"
    .Text = "Select one or more regions"
    For i = 1 To 3
        .CheckBoxes(i).Text = "Region " & i
        Next
        .Show
End With
```

### **Notes Property**

Returns or sets the optional notes associated with a shared workspace link. Read/write **String**.

expression.Notes

*expression* Required. An expression that returns a **SharedWorkspaceLink** object.

The following example creates a list of all the shared workspace links that contain the word "building" in the **Notes** field.

```
Dim strBuildingLinks As String
Dim swsLink As Office.SharedWorkspaceLink
For Each swsLink In ActiveWorkbook.SharedWorkspace.Links
    If InStr(swsLink.Notes, "building", vbTextCompare) > 0 Then
        strBuildingLinks = strBuildingLinks & swsLink.Descriptic
    End If
Next
MsgBox "Building Links: " & vbCrLf & strBuildingLinks, _
    vbInformation + vbOKOnly, "Building Links in Shared Workspac
```

# **Object Property**

Sets or returns the object that is the basis for the specified **COMAddIn** object. Read/write **Object**.

This property is primarily used for enabling one **COMAddIn** to communicate with another **COMAddIn**.

The following example returns the object associated with the COM add in msodraa9.ShapeSelect.

```
Dim objBaseObject As Object
Set objBaseObject = _
Application.COMAddIns.Item("msodraa9.ShapeSelect"). _
Object
```



# **OLEMenuGroup Property**

Returns or sets the menu group that the specified command bar <u>pop-up control</u> belongs to when the menu groups of the OLE server are merged with the menu groups of an OLE client (that is, when an object of the container application type is embedded in another application). Read/write <u>MsoOLEMenuGroup</u>.

MsoOLEMenuGroup can be one of these MsoOLEMenuGroup constants. msoOLEMenuGroupEdit msoOLEMenuGroupFile msoOLEMenuGroupHelp msoOLEMenuGroupObject msoOLEMenuGroupWindow

**Note** This property is read-only for built-in controls.

This property is intended to allow add-in applications to specify how their command bar controls will be represented in the Office application. If either the container or the server does not implement command bars, normal OLE menu merging will occur: the menu bar will be merged, as well as all the toolbars from the server, and none of the toolbars from the container. This property is relevant only for pop-up controls on the menu bar because menus are merged on the basis of their menu group category.

If both of the merging applications implement command bars, command bar controls are merged according to the **OLEUsage** property.

This example checks the **OLEMenuGroup** property of a new custom pop-up control on the command bar named "Custom" and sets the property to **msoOLEMenuGroupNone**.

Set myControl = CommandBars("Custom").Controls \_
.Add(Type:=msoControlPopup,Temporary:=False)
myControl.OLEMenuGroup = msoOLEMenuGroupNone



### **OLEUsage Property**

Returns or sets the OLE client and OLE server roles in which a <u>command bar</u> <u>control</u> will be used when two Microsoft Office applications are merged. Read/write <u>MsoControlOLEUsage</u>.

MsoControlOLEUsage can be one of these MsoControlOLEUsage constants. msoControlOLEUsageBoth msoControlOLEUsageClient msoControlOLEUsageNeither msoControlOLEUsageServer

This property is intended to allow you to specify how individual add-in applications' command bar controls will be represented in one Office application when it is merged with another Office application. If both the client and server implement command bars, the command bar controls are embedded in the client control by control. Custom controls marked as client-only (or neither client nor server) are dropped from the server, and controls marked as server-only (or neither server nor client) are dropped from the client. The remaining controls are merged.

If one of the merging applications isn't an Office application, normal OLE menu merging is used, which is controlled by the **OLEMenuGroup** property.

This example adds a new button to the command bar named Tools, and sets its **OLEUsage** property.

```
Set myControl = CommandBars("Tools").Controls _
    .Add(Type:=msoControlButton,Temporary:=True)
myControl.OLEUsage = msoControlOLEUsageNeither
```

# **On Property**

Some of the content in this topic may not be applicable to some languages.

**True** if the Office Assistant is enabled. Read/write **Boolean**.

This example disables the Office Assistant, displays a message box that asks the user whether the Assistant should be enabled, and enables the Assistant if the user clicks **Yes**. If the users enables the Assistant, the Assistant appears and performs the animation **msoAnimationGreeting**.

```
Assistant.On = False
If MsgBox("Enable Office Assistant?", _
    vbYesNo, "Assistant is Off") = vbYes Then
    Assistant.On = True
    Assistant.Visible = True
    Assistant.Animation = _
        msoAnimationGetAttentionMajor
End If
```



# **OnAction Property**

Returns or sets the name of a Visual Basic procedure that will run when the user clicks or changes the value of a <u>command bar control</u>. Read/write **String**.

**Note** The container application determines whether the value is a valid macro name.

This example adds a command bar control to the command bar named "Custom". The procedure named "MySub" will run each time the control is clicked.

```
Set myBar = CommandBars("Custom")
Set myControl = myBar.Controls _
    .Add(Type:=msocontrolButton)
With myControl
    .FaceId = 2
    .OnAction = "MySub"
End With
myBar.Visible = True
```

This example adds a command bar control to the command bar named "Custom". The COM add in named "FinanceAddIn" will run each time the control is clicked.

```
Set myBar = CommandBars("Custom")
Set myControl = myBar.Controls _
   .Add(Type:=msocontrolButton)
With myControl
   .FaceId = 2
   .OnAction = "!<FinanceAddIn>"
End With
myBar.Visible = True
```

### **Parameter Property**

Returns or sets a string that an application can use to execute a command. Read/write **String**.

If the specified parameter is set for a built-in control, the application can modify its default behavior if it can parse and use the new value. If the parameter is set for custom controls, it can be used to send information to Visual Basic procedures, or it can be used to hold information about the control (similar to a second **Tag** property value).

This example assigns a new parameter to a control and sets the focus to the new button.

```
Set myControl = CommandBars("Custom").Controls(4)
With myControl
    .Copy , 1
    .Parameter = "2"
    .SetFocus
End With
```

### **Parent Property**

Returns the **Parent** object for the specified object. Read-only.

This example displays the name of the parent object for a document property. You must pass a valid **DocumentProperty** object to the procedure.

Sub DisplayParent(dp as DocumentProperty) MsgBox dp.**Parent**.Name End Sub

### **Path Property**

Returns a **String** indicating the full path of a **ScopeFolder** object. Read-only.

#### expression.Path

*expression* Required. An expression that returns a **ScopeFolder** object.

The following example displays the root path of each directory in My Computer. To retrieve this information, the example first gets the **ScopeFolder** object at the root of My Computer. The path of this **ScopeFolder** will always be "\*". As with all **ScopeFolder** objects, the root object contains a **ScopeFolders** collection. This example loops through this **ScopeFolders** collection and displays the path of each **ScopeFolder** object in it. The paths of these **ScopeFolder** objects will be "A:\", "C:\", etc.

```
Sub DisplayRootScopeFolders()
    'Declare variables that reference a
    'SearchScope and a ScopeFolder object.
    Dim ss As SearchScope
    Dim sf As ScopeFolder
    'Use a With...End With block to reference the
    'FileSearch object.
    With Application.FileSearch
        'Loop through the SearchScopes collection
        'and display all of the root ScopeFolders collections in
        'the My Computer scope.
        For Each ss In .SearchScopes
            Select Case ss.Type
                Case msoSearchInMyComputer
                     'Loop through each ScopeFolder collections in
                     'the ScopeFolders collection of the
                    'SearchScope object and display the path.
                    For Each sf In ss.ScopeFolder.ScopeFolders
                        MsgBox "Path: " & sf.Path
                    Next sf
                Case Else
            End Select
        Next ss
    End With
End Sub
```



### **Permission Property**

Returns or sets a **Long** value representing the permissions on the active document assigned to the user associated with the specified **UserPermission** object. Read/write **Long**. The **Permission** property can be one or a combination of **MsoPermission** constants.

MsoPermission can be one of the following msoPermission constants.

```
msoPermissionView (1)
msoPermissionRead (1)
msoPermissionEdit (2)
msoPermissionSave (4)
msoPermissionExtract (8)
msoPermissionChange (15)
msoPermissionPrint (16)
msoPermissionObjectModel (32)
msoPermissionFullControl (64)
```

expression.Permission

expression Required. An expression that returns a UserPermission object.

The **UserPermission** object associates a set of permissions on the active document with a single user and an optional expiration date. The **Permission** property returns the set of user permissions determined by the specified **UserPermission** object. While some permissions granted through the user interface (such as **msoPermissionPrint**) apply to all users, you can use the **UserPermission** object to assign them on a per-user basis with per-user expiration dates.

- The **msoPermissionView** or **msoPermissionRead** option corresponds to the **Read...** option in the user interface.
- The msoPermissionExtract option corresponds to the Allow users with read access to copy content option in the user interface.
- The msoPermissionChange option corresponds to the Change... option in the user interface. The msoPermissionChange option represents the sum of msoPermissionView + msoPermissionEdit + msoPermissionSave + msoPermissionExtract and allows users to view, edit, copy, and save, but not print, the document.
- The **msoPermissionPrint** option corresponds to the **Print content** option in the user interface.
- The **msoPermissionObjectModel** option corresponds to the **Access content programmatically** option in the user interface and allows users to access the document programmatically through its object model. Users without **msoPermissionObjectModel** cannot use the object model to determine their own rights, since programmatic access is disabled.

The following example uses the bitwise **And** operator with the **Permission** property and an **msoPermission** constant to determine whether the second user has permission to save the active document.

## **PermissionFromPolicy Property**

Returns a **Boolean** value that indicates whether a permission policy has been applied to the active document. Read-only **Boolean**.

#### expression. Permission From Policy

*expression* Required. An expression that returns a **Permission** object.

Microsoft Office 2003 Information Rights Management supports the use of administrative permission policies which list users and groups and their document permissions. The **PermissionFromPolicy** property returns a Boolean value that indicates whether a permission policy was applied to the active document the last time permissions were enabled on the document.

The **PermissionFromPolicy** property always returns **False** when checked by a non-owner of the document, even when the user has object model permissions.

The following example displays permission policy information about the active document.

```
Dim irmPermission As Office.Permission
Dim strIRMInfo As String
Set irmPermission = ActiveWorkbook.Permission
If irmPermission. Enabled Then
    strIRMInfo = "Permissions are restricted on this document."
   If irmPermission.PermissionFromPolicy Then
        strIRMInfo = strIRMInfo & " Permissions applied from pol
           Policy name: " & irmPermission.PolicyName & vbCrLf &
        ....
           Policy description: " & irmPermission.PolicyDescripti
    Else
        strIRMInfo = strIRMInfo & " Default permissions applied.
               Default policy name: " & irmPermission.PolicyName
            п
               Default policy description: " & irmPermission.Pol
    End If
Else
    strIRMInfo = "Permission are NOT restricted on this document
End If
MsgBox strIRMInfo, vbInformation + vbOKOnly, "IRM Information"
Set irmPermission = Nothing
```

### **Picture Property**

Returns an **IPictureDisp** object representing the image of a **CommandBarButton** object.

expression.Picture

*expression* Required. An expression that returns a **CommandBarButton** object.

When you change the image on a button, you will also want to use the **Mask** property to set a mask image. The mask image determines which parts of the button image are transparent. Always set the mask after you have set the picture for a **CommandBarButton** object.

**Note** The images for the **View Microsoft** *Application* and **Insert** *Item* buttons on the **Standard** toolbar in the Visual Basic Editor cannot be changed.

The following example sets the image and mask of the first

**CommandBarButton** that the code returns. To make this work, create a mask image and a button image and substitute the paths in the sample with the paths to your images.

```
Sub ChangeButtonImage()
   Dim picPicture As IPictureDisp
   Dim picMask As IPictureDisp
   Set picPicture = stdole.StdFunctions.LoadPicture( _
       "c:\images\picture.bmp")
   "c:\images\mask.bmp")
    'Reference the first button on the first command bar
   'using a With...End With block.
   With Application.CommandBars.FindControl(msoControlButton)
       'Change the button image.
       .Picture = picPicture
       'Use the second image to define the area of the
       'button that should be transparent.
       .Mask = picMask
   End With
End Sub
```

The following example gets the image and mask of the first

**CommandBarButton** that the code returns and outputs each of them to a file. To make this work, specify a path for the output files.

```
Sub GetButtonImageAndMask()
   Dim picPicture As IPictureDisp
   Dim picMask As IPictureDisp
   With Application.CommandBars.FindControl(msoControlButton)
      'Get the button image and mask of this CommandBarButton obje
      Set picPicture = .Picture
      Set picMask = .Mask
End With
   'Save the button image and mask in a folder.
   stdole.SavePicture picPicture, "c:\image.bmp"
```

stdole.SavePicture picMask, "c:\mask.bmp"
End Sub

# **PolicyDescription Property**

Returns the description of the permissions policy applied to the active document. Read-only **String**.

#### expression.PolicyDescription()

*expression* Required. An expression that returns a **Permission** object.

Microsoft Office Information Rights Management supports the use of administrative permission policies which list users and groups and their document permissions. The **PolicyDescription** property returns the description of the policy applied to the active document, or a default value if a policy was not used.

The following example displays permission policy information about the active document.

```
Dim irmPermission As Office.Permission
Dim strIRMInfo As String
Set irmPermission = ActiveWorkbook.Permission
If irmPermission. Enabled Then
    strIRMInfo = "Permissions are restricted on this document."
   If irmPermission.PermissionFromPolicy Then
        strIRMInfo = strIRMInfo & " Permissions applied from pol
           Policy name: " & irmPermission.PolicyName & vbCrLf &
        ....
           Policy description: " & irmPermission.PolicyDescripti
    Else
        strIRMInfo = strIRMInfo & " Default permissions applied.
               Default policy name: " & irmPermission.PolicyName
            п
               Default policy description: " & irmPermission.Pol
    End If
Else
    strIRMInfo = "Permission are NOT restricted on this document
End If
MsgBox strIRMInfo, vbInformation + vbOKOnly, "IRM Information"
Set irmPermission = Nothing
```

### **PolicyName Property**

Returns the name of the permissions policy applied to the active document. Read-only **String**.

expression.PolicyName

*expression* Required. An expression that returns a **Permission** object.

Microsoft Office Information Rights Management supports the use of administrative permission policies which list users and groups and their document permissions. The **PolicyName** property returns the name of the policy applied to the active document, or a default value if a policy was not used.

The following example displays permission policy information about the active document.

```
Dim irmPermission As Office.Permission
Dim strIRMInfo As String
Set irmPermission = ActiveWorkbook.Permission
If irmPermission. Enabled Then
    strIRMInfo = "Permissions are restricted on this document."
   If irmPermission.PermissionFromPolicy Then
        strIRMInfo = strIRMInfo & " Permissions applied from pol
           Policy name: " & irmPermission.PolicyName & vbCrLf &
        ....
           Policy description: " & irmPermission.PolicyDescripti
    Else
        strIRMInfo = strIRMInfo & " Default permissions applied.
               Default policy name: " & irmPermission.PolicyName
            п
               Default policy description: " & irmPermission.Pol
    End If
Else
    strIRMInfo = "Permission are NOT restricted on this document
End If
MsgBox strIRMInfo, vbInformation + vbOKOnly, "IRM Information"
Set irmPermission = Nothing
```



### **Position Property**

Returns or sets the position of a <u>command bar</u>. Read/write <u>MsoBarPosition</u>.

MsoBarPosition can be one of these MsoBarPosition constants. msoBarBottom msoBarFloating msoBarLeft msoBarMenuBar msoBarPopup msoBarRight msoBarTop

This example steps through the collection of command bars, docking the custom command bars at the bottom of the application window and docking the built-in command bars at the top of the window.

```
For Each bar In CommandBars
   If bar.Visible = True Then
        If bar.BuiltIn Then
        bar.Position = msoBarTop
        Else
        bar.Position = msoBarBottom
        End If
   End If
   Next
```



# **Priority Property**

As it applies to the **CommandBarButton**, **CommandBarComboBox**, and **CommandBarControl** objects.

Returns or sets the priority of a <u>command bar control</u>. A control's priority determines whether the control can be dropped from a docked command bar if the command bar controls can't fit in a single row. Controls that can't fit in a single row drop off command bars from right to left. Read/write **Long**.

Valid priority numbers are 0 (zero) through 7 and the default value is 3. A priority of 1 means that the control cannot be dropped from a toolbar. Other priority values are ignored.

The **Priority** property is not used by command bar controls that are menu items.

As it applies to the **SharedWorkspaceTask** object.

Returns or sets the status of the specified shared workspace task. Read/write **msoSharedWorkspaceTaskPriority**.

MsoSharedWorkspaceTaskPriority can be one of the following msoSharedWorkspaceTaskPriority constants.

msoSharedWorkspaceTaskPriorityHigh (1) msoSharedWorkspaceTaskPriorityLow (3) msoSharedWorkspaceTaskPriorityNormal (2)

expression.Priority

*expression* Required. An expression that returns a **SharedWorkspaceTask** object.

The shared workspace task schema on the server can be customized. Customization of the schema may affect the task priority enumeration when the **Add** or **Save** method is called. **Priority** property values are mapped as follows:

- Downloaded value 1 is mapped to msoSharedWorkspaceTaskPriority 1 (msoSharedWorkspaceTaskPriorityHigh). Downloaded values 2 through N-1 are mapped to msoSharedWorkspaceTaskPriority 2 (msoSharedWorkspaceTaskPriorityNormal). Downloaded value N is mapped to msoSharedWorkspaceTaskPriority 3 (msoSharedWorkspaceTaskPriorityLow).
- Uploaded enumeration values 1 through 3 are mapped to schema values 1 through 3. If a user-specified value does not map to any value defined in the schema, the user-specified value is silently ignored and the **Status** property is not updated on the server.

As it applies to the **CommandBarButton**, **CommandBarComboBox**, and **CommandBarControl** object.

This example moves a control and assigns it a priority of 5 so that it will likely be dropped from the command bar if the controls don't all fit in one row.

```
Set allcontrols = CommandBars("Custom").Controls
For Each ctrl In allControls
    If ctrl.Type = msoControlComboBox Then
        With ctrl
        .Move Before:=7
        .Tag = "Selection box"
        .Priority = 5
        End With
        Exit For
End If
Next
```

As it applies to the SharedWorkspaceTask object.

The following example raises the priority of each task in the shared workspace by one step, unless the task priority is already set to High, and uploads the changes to the server. (High priority is the lowest value in the enumeration.)

```
Dim swsTask As Office.SharedWorkspaceTask
Dim lngTaskPriority As MsoSharedWorkspaceTaskPriority
For Each swsTask In ActiveWorkbook.SharedWorkspace.Tasks
lngTaskPriority = swsTask.Priority
If lngTaskPriority > msoSharedWorkspaceTaskPriorityHigh Then
swsTask.Priority = lngTaskPriority - 1
swsTask.Save
End If
Next
Set swsTask = Nothing
```

### **Private Property**

Some of the content in this topic may not be applicable to some languages.

Returns or sets an integer that identifies the Office Assistant balloon that initiated the callback procedure. Read/write **Long**.

This property is helpful if you run the same callback procedure from more than one balloon.

This example identifies the Office Assistant balloon by setting the **Private** property to 129.

```
Set printerOrientation = Assistant.NewBalloon
With printerOrientation .Heading = " Print portrait or landscape
.Text = "Click OK when you've selected the " & _
        "printer orientation."
.Labels(1).Text = "Portrait"
.Labels(2).Text = "Landscape"
.BalloonType = msoBalloonTypeButtons
.Mode = msoModeModeless
.Button = msoButtonSetOK
.Private = 129
.Callback = "PortraitOrLandscape "
.Show
End With
```

## **ProgId Property**

Returns the programmatic identifier (ProgID) for the specified **COMAddIn** object. Read-only **String**.

The following example displays the ProgID and GUID for COM add-in one in a message box.

MsgBox "My ProgID is " & \_ Application.COMAddIns(1).ProgID & \_ " and my GUID is " & \_ Application.COMAddIns(1).Guid

## **PropertyTests Property**

Returns the **<u>PropertyTests</u>** collection that represents all the search criteria for a file search. Read-only.

For information about returning a single member of a collection, see <u>Returning</u> <u>an Object from a Collection</u>.

This example displays all the search criteria for the first property test in the collection.

```
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
```

# **ProportionalFont Property**

Sets or returns the proportional font setting in the host application. Read/write **String**.

When you set the **ProportionalFont** property, the host application does not check the value for validity.

This example sets the proportional font and proportional font size for the English/Western European/Other Latin Script character set in the active application.

```
Application.DefaultWebOptions. _
Fonts(msoCharacterSetEnglishWesternEuropeanOtherLatinScript) _
.ProportionalFont = "Tahoma"
Application.DefaultWebOptions. _
Fonts(msoCharacterSetEnglishWesternEuropeanOtherLatinScript) _
.ProportionalFontSize = 14.5
```

# **ProportionalFontSize Property**

Sets or returns the proportional font size setting in the host application, in points. Read/write **Single**.

When you set the **ProportionalFontSize** property, the host application does not check the value for validity. If you enter an invalid value, such as a nonnumber, the host application sets the size to 0 points. You can enter half-point sizes; if you enter other fractional point sizes, they are rounded up or down to the nearest half-point.

This example sets the proportional font and proportional font size for the English/Western European/Other Latin Script character set in the active application.

```
Application.DefaultWebOptions. _
Fonts(msoCharacterSetEnglishWesternEuropeanOtherLatinScript) _
.ProportionalFont = "Tahoma"
Application.DefaultWebOptions. _
Fonts(msoCharacterSetEnglishWesternEuropeanOtherLatinScript) _
.ProportionalFontSize = 14.5
```



## **Protection Property**

Some of the content in this topic may not be applicable to some languages.

Returns or sets the way a <u>command bar</u> is protected from user customization. Read/write <u>MsoBarProtection</u>.

MsoBarProtection can be one of these MsoBarProtection constants. msoBarNoChangeDock msoBarNoChangeVisible msoBarNoCustomize msoBarNoHorizontalDock msoBarNoMove msoBarNoProtection msoBarNoResize msoBarNoVerticalDock

Using the constant **msoBarNoCustomize** prevents users from accessing the **Add or Remove Buttons** menu (this menu enables users to customize a toolbar).

This example steps through the collection of command bars to find the command bar named "Forms." If this command bar is found, it's docking state is protected and it's made visible.

```
foundFlag = False
For i = 1 To CommandBars.Count
    If CommandBars(i).Name = "Forms" Then
        CommandBars(i).Protection = msoBarNoChangeDock
        CommandBars(i).Visible = True
        foundFlag = True
    End If
Next
If Not foundFlag Then
        MsgBox "'Forms' command bar is not in the collection."
End If
```

# **Reduced Property**

Some of the content in this topic may not be applicable to some languages.

**True** if the Office Assistant window appears in its smaller size. Read/write **Boolean**.

This property is not used in Microsoft Office.

This example displays the Office Assistant in a specific location and it sets several options before making the Assistant visible.

```
With Assistant
   .Reduced = True
   .Left = 400
   .MoveWhenInTheWay = True
   .TipOfDay = True
   .Visible = True
   .Animation = msoAnimationGreeting
End With
```

# **RequestPermissionURL Property**

Returns or sets the file or web site URL to visit or the email address to contact for users who need additional permissions on the active document. Read/write **String**.

#### expression.RequestPermissionURL

*expression* Required. An expression that returns a **Permission** object.

The **RequestPermissionURL** setting corresponds to the **Users can request** additional permissions from option in the permissions user interface. Use the **RequestPermissionURL** property to specify a file, a web site, or an email contact from which users can request, or learn how to request, additional permissions on the active document, for example:

- A web address: http://companyserver/request\_permissions.asp
- A file: \\companyserver\share\requesting\_permissions.txt
- An email address: mailto:permissionsmgr@example.com? Subject=Request%20permissions

The following example displays information about the permissions settings of the active document, including the **RequestPermissionURL** setting.

```
Dim irmPermission As Office.Permission
Dim strIRMInfo As String
Set irmPermission = ActiveWorkbook.Permission
If irmPermission. Enabled Then
    strIRMInfo = "Permissions are restricted on this document."
    strIRMInfo = strIRMInfo & " View in trusted browser: " &
        irmPermission.EnableTrustedBrowser & vbCrLf &
        " Document author: " & irmPermission.DocumentAuthor & vb
        " Users with permissions: " & irmPermission.Count & vbCr
        " Cache licenses locally: " & irmPermission.StoreLicense
        " Request permission URL: " & irmPermission.RequestPermi
    If irmPermission.PermissionFromPolicy Then
        strIRMInfo = strIRMInfo & " Permissions applied from pol
           Policy name: " & irmPermission.PolicyName & vbCrLf &
        н
          Policy description: " & irmPermission.PolicyDescripti
    Else
        strIRMInfo = strIRMInfo & " Default permissions applied.
   End If
Else
    strIRMInfo = "Permissions are NOT restricted on this documen
End If
MsgBox strIRMInfo, vbInformation + vbOKOnly, "IRM Information"
Set irmPermission = Nothing
```

## **RowCount Property**

Returns a **Long** that represents the number of records in the specified data source. Read-only.

#### expression.RowCount

*expression* Required. An expression that returns one of the objects in the Applies To list.

This example adds a new filter that removes all records with a blank Region field and then applies the filter to the active publication.

```
Sub OfficeFilters()
Dim appOffice As OfficeDataSourceObject
Dim appFilters As ODSOFilters
Set appOffice = Application.OfficeDataSourceObject
appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName
    "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
Set appFilters = appOffice.Filters
MsgBox appOffice.RowCount
appFilters.Add Column:="Region", Comparison:=msoFilterComparison
    Conjunction:=msoFilterConjunctionAnd, bstrCompareTo:="WA"
appOffice.ApplyFilter
MsgBox appOffice.RowCount
```

End Sub



## **RowIndex Property**

Returns or sets the docking order of a <u>command bar</u> in relation to other command bars in the same docking area. Can be an integer greater than zero, or either of the following **MsoBarRow** constants: **msoBarRowFirst** or **msoBarRowLast**. Read/write **Long**.

Several command bars can share the same row index, and command bars with lower numbers are docked first. If two or more command bars share the same row index, the command bar most recently assigned will be displayed first in its group.

This example adjusts the position of the command bar named "Custom" by moving it to the left 110 pixels more than the default, and it makes this command bar the first to be docked by changing its row index to **msoBarRowFirst**.

```
Set myBar = CommandBars("Custom")
With myBar
    .RowIndex = msoBarRowFirst
    .Left = 140
End With
```

# **Unsupported Language Element**

You have requested Help for a language element that is not supported.

## **ScopeFolder Property**

Returns a **<u>ScopeFolder</u>** object.

expression.ScopeFolder

*expression* Required. An expression that returns one of the objects in the Applies To list.

The following example displays the root path of each directory in My Computer. To retrieve this information, the example first gets the **ScopeFolder** object at the root of My Computer. The path of this **ScopeFolder** will always be "\*". As with all **ScopeFolder** objects, the root object contains a **ScopeFolders** collection. This example loops through this **ScopeFolders** collection and displays the path of each **ScopeFolder** object in it. The paths of these **ScopeFolder** objects will be "A:\", "C:\", etc.

```
Sub DisplayRootScopeFolders()
    'Declare variables that reference a
    'SearchScope and a ScopeFolder object.
    Dim ss As SearchScope
    Dim sf As ScopeFolder
    'Use a With...End With block to reference the
    'FileSearch object.
    With Application.FileSearch
        'Loop through the SearchScopes collection
        'and display all of the root ScopeFolders collections in
        'the My Computer scope.
        For Each ss In .SearchScopes
            Select Case ss.Type
                Case msoSearchInMyComputer
                     'Loop through each ScopeFolder object in
                     'the ScopeFolders collection of the
                    'SearchScope object and display the path.
                    For Each sf In ss.ScopeFolder.ScopeFolders
                        MsgBox "Path: " & sf.Path
                    Next sf
                Case Else
            End Select
        Next ss
    End With
End Sub
```

## **ScopeFolders Property**

Returns a **<u>ScopeFolders</u>** collection. The items in this collection correspond to the subfolders of the parent **ScopeFolder** object.

#### expression.ScopeFolders

*expression* Required. An expression that returns one of the objects in the Applies To list.

The following example displays the root path of each directory in My Computer. To retrieve this information, the example first gets the **ScopeFolder** object at the root of My Computer. The path of this **ScopeFolder** will always be "\*". As with all **ScopeFolder** objects, the root object contains a **ScopeFolders** collection. This example loops through this **ScopeFolders** collection and displays the path of each **ScopeFolder** object in it. The paths of these **ScopeFolder** objects will be "A:\", "C:\", etc.

```
Sub DisplayRootScopeFolders()
    'Declare variables that reference a
    'SearchScope and a ScopeFolder object.
    Dim ss As SearchScope
    Dim sf As ScopeFolder
    'Use a With...End With block to reference the
    'FileSearch object.
    With Application.FileSearch
        'Loop through the SearchScopes collection
        'and display all of the root ScopeFolders collections in
        'the My Computer scope.
        For Each ss In .SearchScopes
            Select Case ss.Type
                Case msoSearchInMyComputer
                     'Loop through each ScopeFolder object in
                     'the ScopeFolders collection of the
                    'SearchScope object and display the path.
                    For Each sf In ss.ScopeFolder.ScopeFolders
                        MsgBox "Path: " & sf.Path
                    Next sf
                Case Else
            End Select
        Nextb ss
    End With
End Sub
```

# **ScriptText Property**

Sets or returns the text contained in a block of script. Read/write **String**.

The Microsoft Office host application doesn't check the syntax of the script. The **ScriptText** property is the default property for the **Script** object.

This example sets the text of the first script in worksheet one in the active workbook.

```
ActiveWorkbook.Worksheets(1).Scripts(1).ScriptText = _
    "MsgBox ""New ScriptText"""
```

## **SearchFolders Property**

Returns a **<u>SearchFolders</u>** collection.

#### expression.SearchFolders

*expression* Required. An expression that returns one of the objects in the Applies To list.

The following example displays the current number of **ScopeFolder** objects in the **SearchFolders** collection. See the **SearchFolders** collection topic for a more detailed example.

MsgBox "Number of ScopeFolder objects in the SearchFolders collectic Application.FileSearch.SearchFolders.Count

## **SearchScopes Property**

Returns a **<u>SearchScopes</u>** collection.

#### expression.SearchScopes

*expression* Required. An expression that returns one of the objects in the Applies To list.

The following example displays all of the currently available **SearchScope** objects in the **SearchScopes** collection.

```
Sub DisplayAvailableScopes()
    'Declare a variable that references a
    'SearchScope object.
    Dim ss As SearchScope
    'Use a With...End With block to reference the
    'FileSearch object.
   With Application.FileSearch
        'Loop through the SearchScopes collection
        For Each ss In .SearchScopes
            Select Case ss.Type
                Case msoSearchInMyComputer
                    MsgBox "My Computer is an available search scope
                Case msoSearchInMyNetworkPlaces
                    MsgBox "My Network Places is an available search
                Case msoSearchInOutlook
                    MsgBox "Outlook is an available search scope."
                Case msoSearchInCustom
                    MsqBox "A custom search scope is available."
                Case Else
                    MsqBox "Can't determine search scope."
            End Select
        Next ss
    End With
End Sub
```

## **SearchSubFolders Property**

**True** if the search includes all the subfolders in the folder specified by the **LookIn** property. Read/write **Boolean**.

This example searches the My Documents folder and all of its subfolders for all files whose names begin with "Cmd." The example also displays the name and location of each file that's found.

```
Set fs = Application.FileSearch
With fs
.LookIn = "C:\My Documents"
.SearchSubFolders = True
.FileName = "cmd*"
If .Execute() > 0 Then
    MsgBox "There were " & .FoundFiles.Count & _
    " file(s) found."
    For i = 1 To .FoundFiles.Count
        MsgBox .FoundFiles(i)
        Next i
Else
        MsgBox "There were no files found."
End If
End With
```

# SearchWhenProgramming Property

Some of the content in this topic may not be applicable to some languages.

**True** if the Office Assistant displays application and programming Help while the user is working in Visual Basic. Read/write **Boolean**.

The default value is **False**. **SearchWhenProgramming** property corresponds to the **Search for both product and programming help when programming** option in the Use the **Office Assistant** section on the **Options** tab in the **Office Assistant** dialog box.

This example allows the user to search both application and programming help while working in Visual Basic.

Assistant.**SearchWhenProgramming** = True

## **SecondValue Property**

Returns an optional second value property test (as in a range) for the file search. Read-only **Variant**.

This property is intended to be used to specify a range, and it can only be used with the **MsoCondition** constant **msoConditionAnyTimeBetween** or **msoConditionAnyNumberBetween**.

This example displays the second value of the search criteria (if it exists) in a dialog box. If the second value doesn't exist, the example displays another message.

```
With Application.FileSearch.PropertyTests(1)
If .SecondValue = "" Then
    MsgBox "You haven't specified a second value."
Else
    MsgBox "The second value you've set is: " _
    & .SecondValue
End If
End With
```

## **SelectedItems Property**

Returns a **FileDialogSelectedItems** collection. This collection contains a list of the paths of the files that a user selected from a file dialog box displayed using the **Show** method of the **FileDialog** object.

#### expression.SelectedItems

*expression* Required. An expression that returns one of the objects in the Applies To list.

The following example displays a File Picker dialog box using the **FileDialog** object and displays each selected file in a message box.

```
Sub Main()
    'Declare a variable as a FileDialog object.
    Dim fd As FileDialog
    'Create a FileDialog object as a File Picker dialog box.
    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    'Declare a variable to contain the path
    'of each selected item. Even though the path is a String,
    'the variable must be a Variant because For Each...Next
    'routines only work with Variants and Objects.
    Dim vrtSelectedItem As Variant
    'Use a With...End With block to reference the FileDialog object.
   With fd
        'Allow the user to select multiple files.
        .AllowMultiSelect = True
        'Use the Show method to display the File Picker dialog box a
        'If the user presses the action button...
        If .Show = -1 Then
            'Step through each string in the FileDialogSelectedItems
            For Each vrtSelectedItem In .SelectedItems
                'vrtSelectedItem is a String that contains the path
                'You can use any file I/O functions that you want to
                'This example simply displays the path in a message
                MsgBox "Selected item's path: " & vrtSelectedItem
            Next
        'If the user presses Cancel...
        Else
        End If
    End With
    'Set the object variable to Nothing.
    Set fd = Nothing
```

End Sub

# **Shape Property**

Returns a **Shape** object or **InlineShape** object, depending on the Microsoft Office host application. Read-only **Object**.

The **Shape** property returns a **Shape** object in Microsoft Excel and PowerPoint. In Word, the **Shape** property returns a **Shape** object if the script anchor is floating; if it's an inline anchor, however, this property returns an **InlineShape** object.

This example gets the shape associated with the first script in the **Scripts** collection and deletes it from worksheet one in the active workbook.

```
Dim objScriptShape As Object
Set objScriptShape = ______
ActiveWorkbook.Worksheets(1).Scripts(1).Shape
ObjScriptShape.Delete
```



# **ShortcutText Property**

Returns or sets the shortcut key text displayed next to a <u>button control</u> when the button appears on a menu, submenu, or shortcut menu. Read/write **String**.

You can set this property only for command bar buttons that contain an **OnAction** macro.

This example displays the shortcut text for the **Open** command (**File** menu) on the Microsoft Excel Worksheet menu bar in a message box.

```
MsgBox (CommandBars("Worksheet Menu Bar"). _
Controls("File").Controls("New...).ShortcutText)
```

# **SignDate Property**

Returns a **Variant** representing the date and time that the digital certificate corresponding to the **Signature** object was attached to the document. Read-only.

#### expression.SignDate

*expression* Required. An expression that returns one of the objects in the Applies To list.

The following example prompts the user to select a digital signature with which to sign the active document in Microsoft Word. To use this example, open a document in Word and call this function. The function will test to make sure that the digital signature that the user selects will not expire in less than 12 months. If it will expire, the certificate isn't attached.

```
Function AddSignature() As Boolean
    On Error GoTo Error Handler
    Dim sig As Signature
    'Display the dialog box that lets the
    'user select a digital signature.
    'If the user selects a signature, then
    'it is added to the Signatures
    'collection. If the user doesn't, then
    'an error is returned.
    Set sig = ActiveDocument.Signatures.Add
    'Test to make sure that the new Signature object
    'doesn't expire too soon. This expression calculates
    'the number of months until the Signature object expires.
    If DateDiff("m", sig.SignDate, sig.ExpireDate) < 12 Then</pre>
        MsgBox "This certificate will expire in less than 1 year." &
        "Please use a newer certificate."
        AddSignature = False
        sig.Delete
    Else
        AddSignature = True
    End If
    'Commit all signatures in the SignatureSet collection to the dis
    ActiveDocument.Signatures.Commit
    Exit Function
Error Handler:
    AddSignature = False
    MsgBox "Action cancelled."
```

```
End Function
```

# **Signer Property**

Returns a **String** representing the name of the person who attached the digital certificate that corresponds to the **Signature** object to the document. Read-only.

#### expression.Signer

*expression* Required. An expression that returns one of the objects in the Applies To list.

The following example prompts the user to select a digital signature with which to sign the active document in Microsoft Word. To use this example, open a document in Word and pass this function the name of a certificate issuer and the name of a certificate signer that match the **Issued By** and **Issued To** fields of a digital certificate in the **Digital Certificates** dialog box. This example will test to make sure that the digital signature that the user selects meets certain criteria, such as not having expired, before the new signature is committed to the disk.

```
Function AddSignature(ByVal strIssuer As String,
    strSigner As String) As Boolean
    On Error GoTo Error_Handler
    Dim sig As Signature
    'Display the dialog box that lets the
    'user select a digital signature.
    'If the user selects a signature, then
    'it is added to the Signatures
    'collection. If the user doesn't, then
    'an error is returned.
    Set sig = ActiveDocument.Signatures.Add
    'Test several properties before commiting the Signature object t
    If sig.Issuer = strIssuer And
        sig.Signer = strSigner And _
        sig.IsCertificateExpired = False And _
        sig.IsCertificateRevoked = False And _
        siq.IsValid = True Then
        MsgBox "Signed"
        AddSignature = True
    'Otherwise, remove the Signature object from the SignatureSet co
    Else
        sig.Delete
        MsgBox "Not signed"
        AddSignature = False
    End If
    'Commit all signatures in the SignatureSet collection to the dis
    ActiveDocument.Signatures.Commit
```

Exit Function

```
Error_Handler:
AddSignature = False
MsgBox "Action cancelled."
End Function
```

# **SolutionID Property**

Returns or sets the ID, often a globally unique identifier (GUID), which identifies the XML expansion pack attached to the active Microsoft Office Word 2003 document or Microsoft Office Excel 2003 workbook. Read/write **String**.

#### expression.SolutionID

*expression* Required. An expression that returns a <u>SmartDocument</u> object.

The **SolutionID** property returns an empty string or "None" when no XML expansion pack is attached to the active document.

Provide appropriate values for the **SolutionID** and **SolutionUrl** properties to attach an available XML expansion pack to the active document to transform it into a smart document without using the **PickSolution** method. Set the **SolutionID** and **SolutionUrl** properties to empty strings to remove the attached XML expansion pack.

The following example determines whether an XML expansion pack is attached to the active Excel workbook by checking the **SolutionID** property.

```
Dim objSmartDoc As Office.SmartDocument
Set objSmartDoc = ActiveWorkbook.SmartDocument
If objSmartDoc.SolutionID = "None" Or objSmartDoc.SolutionID = "
    MsgBox "No XML expansion pack attached."
Else
    MsgBox "Smart document Solution ID: " & _
        objSmartDoc.SolutionID
End If
Set objSmartDoc = Nothing
```



# **SolutionURL Property**

Returns or sets an <u>absolute URL</u> which provides the complete path to the XML expansion pack file attached to the active Microsoft Office Word 2003 document or Microsoft Office Excel 2003 workbook. Read/write **String**.

#### expression.SolutionURL

*expression* Required. An expression that returns a <u>SmartDocument</u> object.

The **SolutionUrl** property returns an empty string when no XML expansion pack is attached to the active document.

Provide appropriate values for the **SolutionID** and **SolutionUrl** properties to attach an available XML expansion pack to the active document and transform it into a smart document without using the **PickSolution** method. Set the **SolutionID** and **SolutionUrl** properties to empty strings to remove the attached XML expansion pack

The following example determines whether an XML expansion pack is attached to the active Word document, then displays the smart document's Solution URL.

```
Dim objSmartDoc As Office.SmartDocument
Set objSmartDoc = ActiveDocument.SmartDocument
If objSmartDoc.SolutionID = "None" Or objSmartDoc.SolutionID = "
    MsgBox "No XML expansion pack attached."
Else
    MsgBox "Smart document Solution URL: " & _
        objSmartDoc.SolutionURL
End If
Set objSmartDoc = Nothing
```

## **Sounds Property**

Some of the content in this topic may not be applicable to some languages.

**True** if the Office Assistant produces the sounds that correspond to animations. Read/write **Boolean**.

The default value is **True**. The **Sounds** property corresponds to the **Make sounds** option under **Use the Office Assistant** on the **Options** tab in the **Office Assistant** dialog box. If a sound card is not installed, this property has no effect.

This example displays and animates the Office Assistant and allows sound.

```
With Assistant

.Visible = True

.On = True

If Not Sounds Then Sounds = True

.Animation = msoAnimationGreeting

End With
```

# SourceURL Property

Designates the location of the public copy of a shared document to which changes should be published back after the document has been revised in a separate document workspace. Read-only **String**.

#### expression.SourceURL

*expression* Required. An expression that returns a **SharedWorkspace** object.

The MicrosoftWindows SharePoint Services user interface allows users to copy a public shared document from a document library to a separate document workspace for temporary collaboration purposes, then publish changes back to the original source location. The **SourceURL** property indicates the original source location of a document copy stored in a temporary workspace.



## **State Property**

State property as it applies to the **CommandBarButton** object.

Returns or sets the appearance of a command bar <u>button control</u>. Read/write <u>MsoButtonState</u>. However the **State** property of built-in command bar buttons is read-only.

MsoButtonState can be one of these MsoButtonState constants.

msoButtonDown msoButtonMixed msoButtonUp

expression.State

*expression* Required. An expression that returns a **CommandBarButton** object.

State property as it applies to the **HTMLProject** object.

Returns the current state of an **HTMLProject** object. Read-only **MsoHTMLProjectState**.

MsoHTMLProjectState can be one of these MsoHTMLProjectState constants. msoHTMLProjectStateDocumentLocked msoHTMLProjectStateDocumentProjectUnlocked msoHTMLProjectStateProjectLocked

expression.State

*expression* Required. An expression that returns an **HTMLProject** object.

As it applies to the **CommandBarButton** object.

This example creates a command bar named Custom and adds two buttons to it. The example then sets the button on the left to **msoButtonUp** and sets the button on the right to **msoButtonDown**.

```
Dim myBar As Office.CommandBar
Dim imgSource As Office.CommandBarButton
Dim myControl1 As Office.CommandBarButton
Dim myControl2 As Office.CommandBarButton
' Add new command bar.
Set myBar = CommandBars.Add(Name:="Custom", Position:=msoBarTop,
 Add 2 buttons to new command bar.
With myBar
    .Controls.Add Type:=msoControlButton
    .Controls.Add Type:=msoControlButton
    .Visible = True
End With
' Paste Bold button face and set State of first button.
Set myControl1 = myBar.Controls(1)
Set imgSource = CommandBars.FindControl(msoControlButton, 113)
imgSource.CopyFace
With myControl1
    .PasteFace
    .State = msoButtonUp
End With
' Paste Italics button face and set State of second button.
Set myControl2 = myBar.Controls(2)
Set imgSource = CommandBars.FindControl(msoControlButton, 114)
imgSource.CopyFace
With myControl2
    .PasteFace
    .State = msoButtonDown
End With
```



# **Status Property**

Status property as it applies to the **SharedWorkspaceTask** object.

Returns or sets the status of the specified shared workspace task. Read/write **msoSharedWorkspaceTaskStatus**.

MsoSharedWorkspaceTaskStatus can be one of these msoSharedWorkspaceTaskStatus constants. msoSharedWorkspaceTaskStatusComplete (3) msoSharedWorkspaceTaskStatusInProgress (1) msoSharedWorkspaceTaskStatusInProgress (1) msoSharedWorkspaceTaskStatusNotStarted (4) msoSharedWorkspaceTaskStatusWaiting (5)

#### expression.Status

*expression* Required. An expression that returns a **SharedWorkspaceTask** object.

The shared workspace task schema on the server can be customized. Customization of the schema may affect the task status enumeration when the **Add** or **Save** method is called. **Status** property values are mapped as follows:

- Downloaded values 1 through 5 are mapped to msoSharedWorkspaceTaskStatus enumeration values 1 through 5. Schema values beyond 5 are mapped to enumeration value 1 (msoSharedWorkspaceTaskStatusInProgress).
- Uploaded enumeration values 1 through 5 are mapped to schema values 1 through 5. If a user-specified value does not map to any value defined in the schema, the user-specified value is silently ignored and the **Status** property is not updated on the server.

Status property as it applies to the **Sync** object.

Returns the status of the synchronization of the local copy of the active document with the server copy. Read-only <u>MsoSyncStatusType</u>.

MsoSyncStatusType can be one of the following msoSyncStatusType constants. msoSyncStatusConflict (4) msoSyncStatusError (6)

```
msoSyncStatusLatest (1)
msoSyncStatusLocalChanges (3)
msoSyncStatusNewerAvailable (2)
```

```
msoSyncStatusNoSharedWorkspace (0)
```

```
msoSyncStatusSuspended (5)
```

#### expression.Status

expression Required. An expression that returns a Sync object.

Use the **Status** property to determine whether the local copy of the active document is synchronized with the shared server copy. Use the **GetUpdate** method to refresh the status. Use the following methods and properties when appropriate to respond to various status conditions:

- **msoSyncStatusConflict True** when both the local and the server copies have changes. Use the **ResolveConflict** method to resolve the differences.
- **msoSyncStatusError** Check the **ErrorType** property.
- **msoSyncStatusLocalChanges True** when only the local copy has changes. Use the **PutUpdate** method to save local changes to the server copy.
- **msoSyncStatusNewerAvailable True** when only the server copy has changes. Close and re-open the document to work with the latest copy from the server.
- **msoSyncStatusSuspended** Use the **Unsuspend** method to resume synchronization.

The **Status** property returns a single constant from the list in the following order of precedence:

- 1. msoSyncStatusNoSharedWorkspace
- 2. msoSyncStatusError
- 3. msoSyncStatusSuspended
- 4. msoSyncStatusConflict
- 5. msoSyncStatusNewerAvailable
- 6. msoSyncStatusLocalChanges
- 7. msoSyncStatusLatest

As it applies to the **SharedWorkspaceTask** object.

The following example displays a list of all tasks in the current shared workspace whose status is not set to Complete.

```
Dim swsTask As Office.SharedWorkspaceTask
Dim strTaskStatus As String
For Each swsTask In ActiveWorkbook.SharedWorkspace.Tasks
    If swsTask.Status <> msoSharedWorkspaceTaskStatusCompleted T
        strTaskStatus = strTaskStatus & swsTask.Title & vbCrLf
    End If
Next
MsgBox "The following tasks have not been completed:" & vbCrLf &
    strTaskStatus, vbInformation + vbOKOnly, "Incomplete Tasks"
Set swsTask = Nothing
```

As it applies to the **Sync** object.

The following example examines the **Status** property and takes an appropriate action to synchronize the local and server copies of the document if necessary.

```
Dim objSync As Office.Sync
Dim strStatus As String
Set objSync = ActiveDocument.Sync
If objSync.Status > msoSyncStatusNoSharedWorkspace Then
    Select Case objSync.Status
        Case msoSyncStatusConflict
            objSync.ResolveConflict msoSyncConflictMerge
            ActiveDocument.Save
            objSync.ResolveConflict msoSyncConflictClientWins
            strStatus = "Conflict resolved by merging changes."
        Case msoSyncStatusError
            strStatus = "Last error type: " & objSync.ErrorType
        Case msoSyncStatusLatest
            strStatus = "Document copies already in sync."
        Case msoSyncStatusLocalChanges
            objSync.PutUpdate
            strStatus = "Local changes saved to server."
        Case msoSyncStatusNewerAvailable
            strStatus = "Newer copy available on the server."
        Case msoSyncStatusSuspended
            objSync.Unsuspend
```

```
strStatus = "Synchronization resumed."
End Select
Else
strStatus = "Not a shared workspace document."
End If
MsgBox strStatus, vbInformation + vbOKOnly, "Sync Information"
Set objSync = Nothing
```

## **StoreLicenses Property**

Returns a Boolean value that indicates whether the user's license to view the active document should be cached to allow offline viewing when the user cannot connect to a rights management server. Read/write **Boolean**. Default is **True**.

#### expression.StoreLicenses

*expression* Required. An expression that returns a **Permission** object.

The **StoreLicenses** property corresponds to (and its value is the opposite of) the **Require a connection to verify a user's permission** option in the permissions user interface. When **StoreLicenses** is **False**, users other than the document owner must connect to the rights management server and acquire the license to work with the document each time they open it.

When content is protected using the Microsoft Office 2003 Information Rights Management service, the **StoreLicenses** property is always **True** and cannot be set to **False**.

The following example displays information about the permissions settings of the active document, including the **StoreLicenses** setting.

```
Dim irmPermission As Office.Permission
Dim strIRMInfo As String
Set irmPermission = ActiveWorkbook.Permission
If irmPermission. Enabled Then
    strIRMInfo = "Permissions are restricted on this document."
    strIRMInfo = strIRMInfo & " View in trusted browser: " &
        irmPermission.EnableTrustedBrowser & vbCrLf &
        " Document author: " & irmPermission.DocumentAuthor & vb
        " Users with permissions: " & irmPermission.Count & vbCr
        " Cache licenses locally: " & irmPermission.StoreLicense
        " Request permission URL: " & irmPermission.RequestPermi
    If irmPermission.PermissionFromPolicy Then
        strIRMInfo = strIRMInfo & " Permissions applied from pol
           Policy name: " & irmPermission.PolicyName & vbCrLf &
        " Policy description: " & irmPermission.PolicyDescripti
    Else
        strIRMInfo = strIRMInfo & " Custom permissions applied."
   End If
Else
    strIRMInfo = "Permissions are NOT restricted on this documen
End If
MsgBox strIRMInfo, vbInformation + vbOKOnly, "IRM Information"
Set irmPermission = Nothing
```



# **Style Property**

Style property as it applies to the **CommandBarButton** object.

Returns or sets the way a command bar <u>button control</u> is displayed. Read/write <u>MsoButtonStyle</u>.

MsoButtonStyle can be one of these MsoButtonStyle constants. msoButtonAutomatic msoButtonCaption msoButtonIconAndCaption msoButtonIconAndCaptionBelow msoButtonIconAndWrapCaptionBelow msoButtonIconAndWrapCaptionBelow

Style property as it applies to the **CommandBarComboBox** object.

Returns or sets the way a command bar <u>combo box control</u> is displayed. Can be either of the following **MsoComboStyle** constants: **msoComboLabel** or **msoComboNormal**. Read/write <u>MsoComboStyle</u>.

MsoComboStyle can be one of these MsoComboStyle constants. msoComboLabel msoComboNormal

This example creates a shortcut menu containing a button control and a combo box control and sets the style of each.

```
Set myBar = CommandBars ______.Add(Name:="Custom1", Position:=msoBarPopup, Temporary:=False)
With myBar
    .Controls.Add Type:=msoControlButton, Id:=3
    .Controls(1).Style = msoButtonCaption
    .Controls.Add Type:=msoControlComboBox
With .Controls(2)
    .Style = msoComboLabel
    .AddItem "vanilla"
    .AddItem "chocolate"
    .AddItem "cookie dough"
End With
End With
End With
myBar.ShowPopup
```

# **Table Property**

Returns a **String** that represents the name of the table within the data source file that contains the mail merge records. The returned value may be blank if the table name is unknown or not applicable to the current data source. Read-only.

#### expression.Table

*expression* Required. An expression that returns one of the objects in the Applies To list.

This example sets the name of the table if the table name is currently blank.



# **Tag Property**

Returns or sets information about the <u>command bar control</u>, such as data that can be used as an argument in procedures, or information that identifies the control. Read/write **String**.

expression.Tag

*expression* Required. An expression that returns one of the objects in the Applies To list.

To avoid duplicate calls of the same class when tiggered with events, define the **Tag** property unique to the events. The following example demonstrates this concept with two modules.

```
Public WithEvents oBtn As CommandBarButton
Private Sub oBtn_click(ByVal ctrl As Office.CommandBarButton, Cancel
    MsgBox "Clicked " & ctrl.Caption
End Sub
Dim oBtns As New Collection
Sub Use_Tag()
    Dim oEvt As CBtnEvent
    Set oBtns = Nothing
    For i = 1 To 5
        Set oEvt = New CBtnEvent
        Set oEvt.oBtn = Application.CommandBars("Worksheet Menu Bar"
        With oEvt.oBtn
            .Caption = "Btn" & i
            .Style = msoButtonCaption
            .Tag = "Hello" & i
        End With
        oBtns.Add oEvt
    Next
```

End Sub

This example sets the tag for the button on the custom command bar to "Spelling Button" and displays the tag in a message box.

CommandBars("Custom").Controls(1).Tag = "Spelling Button"
MsgBox (CommandBars("Custom").Controls(1).Tag)

# **Tasks Property**

Returns a **SharedWorkspaceTasks** collection that represents the list of tasks in the current shared workspace. Read-only.

expression.Tasks()

*expression* Required. An expression that returns a **SharedWorkspace** object.

The following example lists the tasks in the current shared workspace.

```
Dim swsTasks As Office.SharedWorkspaceTasks
Set swsTasks = ActiveWorkbook.SharedWorkspace.Tasks
MsgBox "There are " & swsTasks.Count & _
        " task(s) in the current shared workspace.", _
        vbInformation + vbOKOnly, _
        "Collection Information"
Set swsTasks = Nothing
```



## **Text Property**

Some of the content in this topic may not be applicable to some languages.

**BalloonLabel** or **BalloonCheckbox** object: Returns or sets the text displayed next to a check box or label in the Office Assistant balloon. Read/write **String**.

**Balloon** object: Returns or sets the text displayed after the heading but before the labels or check boxes in the Office Assistant balloon. Read/write **String**.

**CommandBarComboBox** object: Returns or sets the text in the display or edit portion of the command bar <u>combo box control</u>. Read/write **String**.

**HTMLProjectItem** object: Returns or sets the HTML text in the HTML editor. Read/write **String**.

expression.Text

*expression* Required. An expression that returns one of the objects in the Applies To list.

For the **Balloon**, **BalloonLabel**, and **BalloonCheckbox** objects, you can specify that a particular graphic be displayed by using the following syntax: {*type location sizing\_factor*}, where *type* is bmp (bitmap) or wmf (Windows metafile); *location* is the resource ID or the path and file name; and *sizing\_factor* denotes the width of the .wmf file (*sizing\_factor* is omitted for .bmp files).

The **Balloon** object also supports underlined text and text that has one of the 16 system palette colors applied to it. To display underlined text, use the syntax {*u*} or {*u*l 1}; use {*u*l 0} to turn underlining off. To change the color of text, precede the text string with the character sequence {*cf number*}, where *number* is one of the system color numbers listed in the following table.

System color number	Color
0	Black
1	Dark red
2	Dark green
3	Dark yellow
4	Dark blue
5	Dark magenta
6	Dark cyan
7	Light gray
248	Medium gray
249	Red
250	Green
251	Yellow
252	Blue
253	Magenta
254	Cyan
255	White

If you specify a number other than one of the preceding system color numbers, the text in the Office Assistant balloon is black.

This example creates a new command bar named "Custom" and adds to it a combo box that contains four list items. The example then uses the **Text** property to set Item 3 as the default list item.

```
Set myBar = CommandBars ______.Add(Name:="Custom", Position:=msoBarTop, ______
Temporary:=True)
With myBar
    .Controls.Add Type:=msoControlComboBox, ID:=1
    .Visible = True
End With
Set testComboBox = CommandBars("Custom").Controls(1)
With testComboBox
    .AddItem "Item 1", 1
    .AddItem "Item 1", 1
    .AddItem "Item 2", 2
    .AddItem "Item 3", 3
    .AddItem "Item 4", 4
    .Text = "Item 3"
End With
```

This example creates a new Office Assistant balloon with a heading, text, and three region choices. The example uses the **Text** property to provide balloon-related instructions to the user and a label for each text box.

```
With Assistant.NewBalloon
   .Heading = "Regional Sales Data"
   .Text = "Select a region"
   For i = 1 To 3
        .CheckBoxes(i).Text = "Region " & i
        Next
        .Show
End With
```

This example creates a new Office Assistant balloon that contains underlined heading text, red text, and blue text that is also underlined.

```
With Assistant.NewBalloon
   .Heading = "Underlined {ul 1}Heading{ul 0}"
   .Text = "Some {cf 249}Red{cf 0} text and some " & _
    "underlined {cf 252}{ul 1}Blue{ul 0}{cf 0} text."
```

.Show End With

This example creates a new Office Assistant balloon that contains a Windows metafile.

```
With Assistant.NewBalloon
    .Heading = "Underlined {ul 1}Heading{ul 0}"
    .Text ="{WMF ""C:\Favorites\MyPicture.WMF""}"
    .Show
End With
```

## **TextOrProperty Property**

Returns or sets the word or phrase to be searched for, in either the body of a file or the file's properties, during the file search. The word or phrase can include the \* (asterisk) or ? (question mark) wildcard character. Read/write **String**.

Use the question mark wildcard character to match any single character. For example, type **gr?y** to find all files that contain at least one instance of either "gray" or "grey."

Use the asterisk wildcard character to match any number of characters. For example, type **San\*** to return all files that contain at least one word that begins with "San."

This example searches the C:\My Documents folder and all of its subfolders and returns all files whose body text or file properties contain any words that begin with "San." The **TextOrProperty** property sets the word to be searched for and limits the search to either the body of the file or the file properties.

```
With Application.FileSearch
    .NewSearch
    .LookIn = "C:\My Documents"
    .SearchSubFolders = True
    .TextOrProperty = "San*"
    .FileType = msoFileTypeAllFiles
End With
```

# **TipOfDay Property**

Some of the content in this topic may not be applicable to some languages.

**True** if the Office Assistant displays a special tip each time the Office application is opened. Read/write **Boolean**.

The default value is **False**. **TipOfDay** property corresponds to the **Show the Tip of the Day at startup** option under **Show tips about** on the **Options** tab in the **Office Assistant** dialog box.

This example displays the Office Assistant in a specific location and it sets several options before making the Assistant visible.

```
With Assistant
    .On = True
    .Visible = True
    .Left = 400
    .MoveWhenInTheWay = True
    If Not TipOfDay Then TipOfDay = True
    .Animation = msoAnimationGreeting
End With
```



# **Title Property**

As it applies to the **FileDialog** object.

Sets or returns the title of a file dialog box displayed using the **FileDialog** object. Read/write **String**.

expression.Title

*expression* Required. An expression that returns one of the objects in the Applies To list.

As it applies to the **SharedWorkspaceTask** object.

Sets or returns the title of a **SharedWorkspaceTask** object. Read/write **String**.

expression.Title

*expression* Required. An expression that returns a **SharedWorkspaceTask** object.

The **Title** property is the single required property of a shared workspace task. Use the optional **Description** property to provide or return additional information about the task.

#### As it applies to the **FileDialog**object.

The following example displays a File Picker dialog box using the **FileDialog** object and displays each selected file in a message box.

```
Sub main()
    'Declare a variable as a FileDialog object.
    Dim fd As FileDialog
    'Create a FileDialog object as a File Picker dialog box.
    Set fd = Application.FileDialog(msoFileDialogFilePicker)
    'Declare a variable to contain the path
    'of each selected item. Even though the path is a String,
    'the variable must be a Variant because For Each...Next
    'routines only work with Variants and Objects.
    Dim vrtSelectedItem As Variant
    'Use a With...End With block to reference the FileDialog object.
   With fd
        'Change the title of the dialog
        .Title = "Archive"
        'Use the Show method to display the file picker dialog and r
        'If the user presses the action button...
        If .Show = -1 Then
            'Step through each string in the FileDialogSelectedItems
            For Each vrtSelectedItem In .SelectedItems
                'vrtSelectedItem is a String that contains the path
                'You can use any file I/O functions that you want to
                'This example simply displays the path in a message
                MsgBox "Selected item's path: " & vrtSelectedItem
            Next vrtSelectedItem
        'If the user presses Cancel...
        Else
        End If
    End With
    'Set the object variable to Nothing.
```

```
Set fd = Nothing
```

End Sub

As it applies to the **SharedWorkspaceTask** object.

The following example displays a list of the titles of all tasks in the current shared workspace.

```
Dim swsTask As Office.SharedWorkspaceTask
Dim strTasks As String
For Each swsTask In ActiveWorkbook.SharedWorkspace.Tasks
   strTasks = strTasks & swsTask.Title & vbCrLf
Next
MsgBox strTasks, vbInformation + vbOKOnly, _
   "Tasks in Shared Workspace"
Set swsTask = Nothing
```

# **TooltipText Property**

Returns or sets the text displayed in a command bar control's ScreenTip. Read/write **String**.

By default, the value of the <u>Caption</u> property is used as the ScreenTip.

This example adds a ScreenTip to the last control on the active menu bar.

```
Set myMenuBar = CommandBars.ActiveMenuBar
Set lastCtrl = myMenuBar _
    .Controls(myMenuBar.Controls.Count)
lastCtrl.BeginGroup = True
lastCtrl.TooltipText = "Click for help on UI feature"
```



# **Top Property**

Some of the content in this topic may not be applicable to some languages.

Top property as it applies to the **Assistant** and **CommandBar** objects.

Sets or returns the distance (in points) from the top of the Office Assistant, or from the top edge of the specified <u>command bar</u>, to the top edge of the screen. For docked command bars, this property returns or sets the distance from the command bar to the top of the docking area. Read/write **Long**.

#### expression.**Top**

*expression* Required. An expression that returns one of the above objects.

Top property as it applies to the **CommandBarButton**, **CommandBarComboBox**, **CommandBarControl**, and **CommandBarPopup** objects.

Returns the distance (in pixels) from the top edge of the specified <u>command bar</u> <u>control</u> to the top edge of the screen. Read-only **Long**.

#### expression.**Top**

*expression* Required. An expression that returns one of the above objects.

As it applies to the Assistant and CommandBar objects.

This example moves the Office Assistant to another coordinate and sets its **Top** property for subsequent appearances.

This example positions the upper-left corner of the floating command bar named Custom 140 pixels from the left edge of the screen and 100 pixels from the top of the screen.

```
Set myBar = CommandBars("Custom")
myBar.Position = msoBarFloating
With myBar
    .Left = 140
    .Top = 100
End With
```



# **Type Property**

Type property as it applies to the **CommandBar** object.

Returns the type of <u>command bar</u>. Read-only <u>MsoBarType</u>.

MsoBarType can be one of these MsoBarType constants. msoBarTypeMenuBar msoBarTypeNormal msoBarTypePopup

expression.**Type** 

*expression* Required. An expression that returns a **CommandBar** object.

Type property as it applies to the **CommandBarButton**, **CommandBarComboBox**, **CommandBarControl**, and **CommandBarPopup** <u>objects</u>.

Returns the type of command bar control. Read-only MsoControlType.

MsoControlType can be one of these MsoControlType constants.

msoControlActiveX msoControlAutoCompleteCombo msoControlButton msoControlButtonDropdown msoControlButtonPopup msoControlComboBox msoControlCustom msoControlDropdown msoControlEdit msoControlExpandingGrid msoControlGauge **msoControlGenericDropdown** msoControlGraphicCombo msoControlGraphicDropdown msoControlGraphicPopup **msoControlGrid** msoControlLabel **msoControlLabelEx** msoControlOCXDropdown **msoControlPane msoControlPopup msoControlSpinner** msoControlSplitButtonMRUPopup msoControlSplitButtonPopup msoControlSplitDropdown msoControlSplitExpandingGrid msoControlWorkPane

#### expression.Type

*expression* Required. An expression that returns one of the above objects.

Type property as it applies to the **SearchScope** object.

Returns a value that corresponds to the type of **SearchScope** object. The type indicates the area in which the **Execute** method of the **FileSearch** object will search for files. Read-only **MsoSearchIn**.

MsoSearchIn can be one of these MsoSearchIn constants.

```
msoSearchInCustom
msoSearchInMyComputer
msoSearchInMyNetworkPlaces
msoSearchInOutlook
```

```
expression.Type
```

*expression* Required. An expression that returns a **SearchScope** object.

Type property as it applies to the **DocumentProperty** object.

Returns or sets the document property type. Read-only for built-in document properties; read/write for custom document properties.

expression.**Type** 

*expression* Required. An expression that returns a **DocumentProperty** object.

The return value will be a **MsoDocProperties** constant.

MsoDocProperties can be one of these MsoDocProperties constants. msoPropertyTypeBoolean msoPropertyTypeDate msoPropertyTypeFloat msoPropertyTypeNumber msoPropertyTypeString

As it applies to the **CommandBar** object.

This example finds the first control on the command bar named Custom. Using the **Type** property, the example determines whether the control is a button. If the control is a button, the example copies the face of the **Copy** button (on the **Standard** toolbar) and then pastes it onto the control.

```
Set oldCtrl = CommandBars("Custom").Controls(1)
If oldCtrl.Type = msoControlButton Then
    Set newCtrl = CommandBars.FindControl(Type:= _
        MsoControlButton, ID:= _
        CommandBars("Standard").Controls("Copy").ID)
    NewCtrl.CopyFace
    OldCtrl.PasteFace
End If
```

#### As it applies to the **DocumentProperty** object.

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
```



# **URL Property**

As it applies to the **SharedWorkspace** object.

Returns the top-level <u>Uniform Resource Locator (URL</u>) of the shared workspace. Read-only **String**.

expression.**URL** 

*expression* Required. An expression that returns a **SharedWorkspace** object.

The **URL** property returns the address of the shared workspace in this format: http://server/sites/user/workspace/.

The **URL** property returns a URL-encoded string. For example, a space in the folder name is represented by %20. Use a simple function like the following example to replace this escaped character with a space.

As it applies to the **SharedWorkspaceFile** object.

Returns the full <u>Uniform Resource Locator (URL</u>) and file name of the shared workspace file. Read-only **String**.

expression.URL

*expression* Required. An expression that returns a **SharedWorkspaceFile** object.

The **URL** property returns the address of the shared workspace file in this format:

http://server/sites/user/workspace/Shared%Documents/MyWorkbook.xls.

The **URL** property returns a URL-encoded string. For example, a space in the folder name is represented by %20.

The **SharedWorkspaceFile** object does not have a **Name** or **FileName** property. The filename must be extracted from the **URL** property. Use simple functions as in the following example to extract the filename and to decode spaces in the URL.

```
Private Function FilenameFromURL(FileURL As String) As String
   Dim intLastSeparator As Integer
   FileURL = URLDecode(FileURL)
   intLastSeparator = InStrRev(FileURL, "/")
   FilenameFromURL = Right(FileURL, Len(FileURL) - intLastSeparator
End Function
Private Function URLDecode(URLtoDecode As String) As String
   URLDecode = Replace(URLtoDecode, "%20", " ")
End Function
```

As it applies to the **SharedWorkspaceLink** object.

Returns or sets the <u>Uniform Resource Locator (URL</u>) of the link saved in the shared workspace. Read/write **String**.

expression.URL

*expression* Required. An expression that returns a **SharedWorkspaceLink** object.

Use the **URL** property of the **SharedWorkspaceLink** object to retrieve the web address saved in the shared workspace link. Use the optional **Description** and **Notes** properties to retrieve additional information about the link.

The **URL** property returns a URL-encoded string. For example, a space in the folder name is represented by %20. Use a simple function like the following example to replace this escaped character with a space.

```
Private Function URLDecode(URLtoDecode As String) As String
    URLDecode = Replace(URLtoDecode, "%20", " ")
End Function
```

As it applies to the **SharedWorkspace** object.

The following example displays the base URL of the shared workspace.

```
MsgBox "URL: " & ActiveWorkbook.SharedWorkspace.URL, _
vbInformation + vbOKOnly, "Shared Workspace URL"
```

As it applies to the **SharedWorkspaceFile** object.

The following example locates all Microsoft Excel workbooks in the **SharedWorkspaceFiles** collection by examining the **URL** property of each file for the "xls" file extension.

```
Dim strExcelFiles As String
Dim swsFile As Office.SharedWorkspaceFile
For Each swsFile In ActiveWorkbook.SharedWorkspace.Files
If Right(swsFile.URL, 3) = "xls" Then
strExcelFiles = strExcelFiles & swsFile.URL & vbCrLf
End If
Next
MsgBox "Excel Files: " & vbCrLf & strExcelFiles, _
vbInformation + vbOKOnly, "Excel Files in Shared Workspace"
```

The following example lists all files in the shared workspace, using supporting functions to extract their filenames from the URL and to convert URL-encoded spaces from %20 to a space character.

```
Dim swsFile As Office.SharedWorkspaceFile
Dim strFileList As String
For Each swsFile In ActiveWorkbook.SharedWorkspace.Files
strFileList = "Filename: " & FilenameFromURL(swsFile.URL) &
vbCrLf & " URL: " & URLDecode(swsFile.URL) & vbCrLf
Next
MsgBox strFileList, vbInformation + vbOKOnly, "Files in Shared W
Set swsFile = Nothing
Private Function FilenameFromURL(FileURL As String) As String
Dim intLastSeparator As Integer
FileURL = URLDecode(FileURL)
intLastSeparator = InStrRev(FileURL, "/")
FilenameFromURL = Right(FileURL, Len(FileURL) - intLastSeparator
```

End Function

```
Private Function URLDecode(URLtoDecode As String) As String
    URLDecode = Replace(URLtoDecode, "%20", " ")
End Function
```

As it applies to the **SharedWorkspaceLink** object.

The following example locates all links to the Microsoft Developer Network (MSDN) web site in the **SharedWorkspaceLinks** collection by examining the **URL** property of each link for the string "msdn".

```
Dim strMSDNLinks As String
Dim swsLink As Office.SharedWorkspaceLink
For Each swsLink In ActiveWorkbook.SharedWorkspace.Links
If InStr(swsLink.URL, "msdn", vbTextCompare) > 0 Then
strMSDNLinks = strMSDNLinks & swsLink.URL & vbCrLf
End If
Next
MsgBox "MSDN Links: " & vbCrLf & strMSDNLinks, _
vbInformation + vbOKOnly, "MSDN Links in Shared Workspace"
```

# **UserId Property**

Returns the email name of the user whose permissions on the active document are determined by the specified **UserPermission** object. Read-only **String**.

expression.UserId

*expression* Required. An expression that returns a **UserPermission** object.

The **UserPermission** object associates a set of permissions on the active document with a single user and an optional expiration date. The **UserID** property returns the name in email form of the user whose permissions are determined by the specified **UserPermission** object.

The following example lists the users who have permissions on the active document.

```
Dim irmPermission As Office.Permission
Dim irmUserPerm As Office.UserPermission
Dim strUsers As String
Set irmPermission = ActiveWorkbook.Permission
If irmPermission. Enabled Then
   For Each irmUserPerm In irmPermission
        strUsers = strUsers & irmUserPerm.UserId & vbCrLf
   Next
   MsgBox strUsers, _
        vbInformation + vbOKOnly, "IRM Information"
Else
   MsgBox "Permissions are not enabled for this document.", _
        vbInformation + vbOKOnly, "IRM Information"
End If
Set irmUserPerm = Nothing
Set irmPermission = Nothing
```



# **Value Property**

Value property as it applies to the **DocumentProperty** object.

Returns or sets the value of a document property. Read/write **Variant**.

expression.Value

*expression* Required. An expression that returns a **DocumentProperty** object.

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 as it applies to the **PropertyTest** object.

Returns the value of a property test for a file search. Read-only **Variant**.

expression.Value

*expression* Required. An expression that returns a **PropertyTest** object.

#### As it applies to the **DocumentProperty** object.

This example displays the name, type, and value of a document property. For the example to work, dp must be a valid **DocumentProperty** object.

```
Sub DisplayPropertyInfo(dp As DocumentProperty)
    MsgBox "value = " & dp.Value & Chr(13) & _
        "type = " & dp.Type & Chr(13) & _
        "name = " & dp.Name
End Sub
```

As it applies to the **PropertyTest** object.

This example displays the value of the search criteria (if it exists) in a message box. If the second value doesn't exist, the example displays another message.

```
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
```

# **Visible Property**

Some of the content in this topic may not be applicable to some languages.

**True** if the specified object is visible. Read/write **Boolean**.

#### expression.Visible

*expression* Required. An expression that returns one of the objects in the Applies To list.

The **Visible** property for newly created custom command bars is **False** by default.

The **Enabled** property for a command bar must be set to **True** before the visible property is set to **True**.

This example steps through the collection of command bars to find the Forms command bar. If the Forms command bar is found, the example makes it visible and protects its docking state.

```
foundFlag = False
For Each cmdbar In CommandBars
    If cmdbar.Name = "Forms" Then
        cmdbar.Protection = msoBarNoChangeDock
        cmdbar.Visible = True
        foundFlag = True
        End If
Next
If Not foundFlag Then
        MsgBox "'Forms'command bar is not in the collection."
End If
```

This example makes the Office Assistant visible and sets its animation.

```
With Application.Assistant
   .Visible = True
   .Sounds = True
   .Animation = msoAnimationBeginSpeaking
End With
```



## Width Property

Returns or sets the width (in pixels) of the specified <u>command bar</u> or <u>command</u> <u>bar control</u>. Read/write **Integer**.

#### expression.Width

*expression* Required. An expression that returns one of the objects in the Applies To list.

This example adds a custom control to the command bar named Custom. The example sets the height of the custom control to twice the height of the command bar and sets its width to 50 pixels. Notice how the command bar automatically resizes itself to accommodate the control.

```
Set myBar = CommandBars("Custom")
barHeight = myBar.Height
Set myControl = myBar.Controls _
    .Add(Type:=msoControlButton, _
    Id:= CommandBars("Standard").Controls("Save").Id, _
    Temporary:=True)
With myControl
    .Height = barHeight * 2
    .Width = 50
End With
myBar.Visible = True
```

## WorkspaceLastChangedBy Property

Displays the friendly name of the user who last saved changes to the server copy of a shared document. Read-only **String**.

#### expression.WorkspaceLastChangedBy

*expression* Required. An expression that returns a **Sync** object.

#### Remarks

If the active document is not configured for synchronization between the local copy and the server copy, the **WorkspaceLastChangedBy** property raises a run-time error.

The following example checks for a conflict between the local and the server copies of the shared document and reports the name of the user who last saved changes to the server copy.

```
Dim objSync As Office.Sync
Dim strStatus As String
Set objSync = ActiveDocument.Sync
If objSync.Status = msoSyncStatusConflict Then
    strStatus = "The server copy has been changed." & vbCrLf & _
        "Changes have been made by: " & _
        objSync.WorkspaceLastChangedBy
    MsgBox strStatus, vbInformation + vbOKOnly, "Server Copy Cha
End If
Set objSync = Nothing
```

## **Change Event**

Occurs when the end user changes the selection in a command bar combo box.

#### Private Sub CommandBarComboBox\_Change

(ByVal Ctrl As CommandBarComboBox)

#### Remarks

The **Change** event is recognized by the **CommandBarComboBox** object. To return the Change event for a particular **CommandBarComboBox** control, use the **WithEvents** keyword to declare a variable, and then set the variable to the **CommandBarComboBox** control. When the **Change** event is triggered, it executes the macro or code that you specified with the **OnAction** property of the control.

The following example creates a command bar with a **CommandBarComboBox** control containing four selections. The combo box responds to user interaction through the CommandBarComboBox\_Change event.

```
Private ctlComboBoxHandler As New ComboBoxHandler
Sub AddComboBox()
```

```
Set HostApp = Application
Dim newBar As Office.CommandBar
Set newBar = HostApp.CommandBars.Add(Name:="Test CommandBar", Te
Dim newCombo As Office.CommandBarComboBox
Set newCombo = newBar.Controls.Add(msoControlComboBox)
With newCombo
    .AddItem "First Class", 1
    .AddItem "Business Class", 2
    .AddItem "Coach Class", 3
    .AddItem "Standby", 4
    .DropDownLines = 5
    .DropDownWidth = 75
    .ListHeaderCount = 0
End With
ctlComboBoxHandler.SyncBox newCombo
newBar.Visible = True
```

```
End Sub
```

The preceding example relies on the following code, which is stored in a class module in the VBA project.

```
Private WithEvents ComboBoxEvent As Office.CommandBarComboBox
Public Sub SyncBox(box As Office.CommandBarComboBox)
    Set ComboBoxEvent = box
    If Not box Is Nothing Then
        MsgBox "Synced " & box.Caption & " ComboBox events."
    End If
End Sub
Private Sub Class_Terminate()
```

```
Set ComboBoxEvent = Nothing
End Sub
Private Sub ComboBoxEvent_Change(ByVal Ctrl As Office.CommandBarComb
    Dim stComboText As String
    stComboText = Ctrl.Text
        Select Case stComboText
        Case "First Class"
            FirstClass
        Case "Business Class"
            BusinessClass
        Case "Coach Class"
            CoachClass
        Case "Standby"
            Standby
    End Select
End Sub
Private Sub FirstClass()
    MsgBox "You selected First Class reservations"
End Sub
Private Sub BusinessClass()
    MsgBox "You selected Business Class reservations"
End Sub
Private Sub CoachClass()
    MsgBox "You selected Coach Class reservations"
End Sub
Private Sub Standby()
    MsgBox "You chose to fly standby"
End Sub
```

## **Click Event**

Occurs when the user clicks a **CommandBarButton** object.

#### Private Sub CommandBarButton\_Click

#### (ByVal Ctrl As CommandBarButton,

#### ByVal CancelDefault As Boolean)

The syntax for the Click event includes the two arguments described in the following table.

Argument	Description	
Ctrl	Required <b>CommandBarButton</b> . Denotes the	
Cui	<b>CommandBarButton</b> control that initiated the event.	
	Required <b>Boolean</b> . <b>False</b> if the default behavior associated with	
CancelDefault the CommandBarButton control occurs, unless it's canceled		
	by another process or add-in.	

#### Remarks

The Click event is recognized by the **CommandBarButton** object. To return the Click event for a particular **CommandBarButton** control, use the **WithEvents** keyword to declare a variable, and then set the variable to the control.

The following example creates a new command bar button on the File menu of the host application that enables the user to save a workbook as a commaseparated value file. (This example works in all applications, but the context of saving as CSV is applicable to Microsoft Excel.)

```
Private HostApp As Object
Sub createAndSynch()
    Dim iIndex As Integer
    Dim iCount As Integer
    Dim fBtnExists As Boolean
    Dim obCmdBtn As Object
    Dim btnSaveAsCSVHandler as new Class1
    On Error GoTo errHandler
    Set HostApp = Application
    Dim barHelp As Office.CommandBar
    Set barHelp = Application.CommandBars("File")
    fBtnExists = False
    iCount = barHelp.Controls.Count
    For iIndex = 1 To iCount
        If barHelp.Controls(iIndex).Caption = "Save As CSV (Comma De
    Next
    Dim btnSaveAsCSV As Office.CommandBarButton
    If fBtnExists Then
        Set btnSaveAsCSV = barHelp.Controls("Save As CSV (Comma Deli
    Else
        Set btnSaveAsCSV = barHelp.Controls.Add(msoControlButton)
        btnSaveAsCSV.Caption = "Save As CSV (Comma Delimited)"
    End If
    btnSaveAsCSV.Tag = "btn1"
    btnSaveAsCSVHandler.SyncButton btnSaveAsCSV
    Exit Sub
errHandler:
    ' Insert error handling code here
End Sub
```

## **EnvelopeHide Event**

Occurs when the user interface (UI) that corresponds to the **MsoEnvelope** object is hidden.

#### Private Sub object\_EnvelopHide()

*object* A variable which references an object of type **MsoEnvelope** declared with events in a class module.

The following example sets up event-handling routines for the **MsoEnvelope** object.

```
Public WithEvents env As MsoEnvelope
Private Sub Class_Initialize()
   Set env = Application.ActiveDocument.MailEnvelope
End Sub
Private Sub env_EnvelopeShow()
   MsgBox "The MsoEnvelope UI is showing."
End Sub
Private Sub env_EnvelopeHide()
   MsgBox "The MsoEnvelope UI is hidden."
End Sub
```

## **EnvelopeShow Event**

Occurs when the user interface (UI) that corresponds to the **MsoEnvelope** object is displayed.

#### Private Sub object\_EnvelopShow()

*object* A variable which references an object of type **MsoEnvelope** declared with events in a class module.

The following example sets up event-handling routines for the **MsoEnvelope** object.

```
Public WithEvents env As MsoEnvelope
Private Sub Class_Initialize()
   Set env = Application.ActiveDocument.MailEnvelope
End Sub
Private Sub env_EnvelopeShow()
   MsgBox "The MsoEnvelope UI is showing."
End Sub
Private Sub env_EnvelopeHide()
   MsgBox "The MsoEnvelope UI is hidden."
End Sub
```

### **OnUpdate Event**

Occurs when any change is made to a command bar.

Private Sub CommandBars\_OnUpdate()

#### Remarks

The OnUpdate event is recognized by the **CommandBar** object and all command bar controls. The event is triggered by any change to a command bar or command bar control or any change to the state of a command bar or command bar control. These changes can occur due to a text or cell selection, for example. Since a large number of OnUpdate events can occur during normal usage, developers should exercise caution when using this event. It is strongly recommended that this event be used primarily for checking that a custom command bar has been added or removed by a **COMAddIn**.



## **Microsoft Office Enumerated Constants**

This topic provides a list of all enumerated constants in the Microsoft Office object model.

<u>MailFormat</u>

## **Constant Value** mfHTML 2

mfPlainText 1 mfRTF 3

#### **MsoAlertButtonType**

Constant	Value
msoAlertButtonAbortRetryIgnore	2
msoAlertButtonOK	0
msoAlertButtonOKCancel	1
msoAlertButtonRetryCancel	5
msoAlertButtonYesAllNoCancel	6
msoAlertButtonYesNo	4
msoAlertButtonYesNoCancel	3

**MsoAlertCancelType** 

Constant	Value
msoAlertCancelDefault	:-1
msoAlertCancelFifth	4
msoAlertCancelFirst	0
	_

msoAlertCancelFourth 3

msoAlertCancelSecond 1

#### msoAlertCancelThird 2

#### <u>MsoAlertDefaultType</u>

ConstantValuemsoAlertDefaultFifth4msoAlertDefaultFirst0msoAlertDefaultFourth3msoAlertDefaultSecond1msoAlertDefaultThird2

#### **MsoAlertIconType**

# Constant Value msoAlertIconCritical 1 msoAlertIconInfo 4 msoAlertIconNoIcon 0 msoAlertIconQuery 2 msoAlertIconWarning 3

#### **MsoAlignCmd**

Constant	Value
msoAlignBottoms	5
msoAlignCenters	1
msoAlignLefts	0
msoAlignMiddles	4
msoAlignRights	2
msoAlignTops	3

#### **MsoAnimationType**

Constant	Value
msoAnimationAppear	32

msoAnimationBeginSpeaking	4
msoAnimationCharacterSuccessMajor	6
msoAnimationCheckingSomething	103
msoAnimationDisappear	31
msoAnimationEmptyTrash	116
msoAnimationGestureDown	113
msoAnimationGestureLeft	114
msoAnimationGestureRight	19
msoAnimationGestureUp	115
msoAnimationGetArtsy	100
msoAnimationGetAttentionMajor	11
msoAnimationGetAttentionMinor	12
msoAnimationGetTechy	101
msoAnimationGetWizardy	102
msoAnimationGoodbye	3
msoAnimationGreeting	2
msoAnimationIdle	1
msoAnimationListensToComputer	26
msoAnimationLookDown	104
msoAnimationLookDownLeft	105
msoAnimationLookDownRight	106
msoAnimationLookLeft	107
msoAnimationLookRight	108
msoAnimationLookUp	109
msoAnimationLookUpLeft	110
msoAnimationLookUpRight	111
msoAnimationPrinting	18
msoAnimationRestPose	5
msoAnimationSaving	112
msoAnimationSearching	13
msoAnimationSendingMail	25
msoAnimationThinking	24
msoAnimationWorkingAtSomething	23

#### msoAnimationWritingNotingSomething 22

#### **MsoAppLanguageID**

Constant	Value
msoLanguageIDExeMode	4
msoLanguageIDHelp	3
msoLanguageIDInstall	1
msoLanguageIDUI	2
msoLanguageIDUIPrevious	5

#### **MsoArrowheadLength**

Constant	Value
msoArrowheadLengthMedium	2
msoArrowheadLengthMixed	-2
msoArrowheadLong	3
msoArrowheadShort	1

#### **MsoArrowheadStyle**

#### Constant Value

- msoArrowheadDiamond 5 msoArrowheadNone 1
- msoArrowheadOpen 3 6
- msoArrowheadOval
- msoArrowheadStealth 4
- msoArrowheadStyleMixed -2
- msoArrowheadTriangle 2

#### <u>MsoArrowheadWidth</u>

Constant	Value
msoArrowheadNarrow	1

msoArrowheadWide 3 msoArrowheadWidthMedium 2 msoArrowheadWidthMixed -2

#### **MsoAutomationSecurity**

Constant	Value
msoAutomationSecurityByUI	2
msoAutomationSecurityForceD is able	3
msoAutomationSecurityLow	1

#### **MsoAutoShapeType**

Constant	Value
msoShape16pointStar	94
msoShape24pointStar	95
msoShape32pointStar	96
msoShape4pointStar	91
msoShape5pointStar	92
msoShape8pointStar	93
msoShapeActionButtonBackorPrevious	129
msoShapeActionButtonBeginning	131
msoShapeActionButtonCustom	125
msoShapeActionButtonDocument	134
msoShapeActionButtonEnd	132
msoShapeActionButtonForwardorNext	130
msoShapeActionButtonHelp	127
msoShapeActionButtonHome	126
msoShapeActionButtonInformation	128
msoShapeActionButtonMovie	136
msoShapeActionButtonReturn	133
msoShapeActionButtonSound	135
msoShapeArc	25
msoShapeBalloon	137

msoShapeBentArrow	41
msoShapeBentUpArrow	44
msoShapeBevel	15
msoShapeBlockArc	20
msoShapeCan	13
msoShapeChevron	52
msoShapeCircularArrow	60
msoShapeCloudCallout	108
msoShapeCross	11
msoShapeCube	14
msoShapeCurvedDownArrow	48
msoShapeCurvedDownRibbon	100
msoShapeCurvedLeftArrow	46
msoShapeCurvedRightArrow	45
msoShapeCurvedUpArrow	47
msoShapeCurvedUpRibbon	99
msoShapeDiamond	4
msoShapeDonut	18
msoShapeDoubleBrace	27
msoShapeDoubleBracket	26
msoShapeDoubleWave	104
msoShapeDownArrow	36
msoShapeDownArrowCallout	56
msoShapeDownRibbon	98
msoShapeExplosion1	89
msoShapeExplosion2	90
msoShapeFlowchartAlternateProcess	62
msoShapeFlowchartCard	75
msoShapeFlowchartCollate	79
msoShapeFlowchartConnector	73
msoShapeFlowchartData	64
msoShapeFlowchartDecision	63
msoShapeFlowchartDelay	84

msoShapeFlowchartDirectAccessStorage	87
msoShapeFlowchartDisplay	88
msoShapeFlowchartDocument	67
msoShapeFlowchartExtract	81
msoShapeFlowchartInternalStorage	66
msoShapeFlowchartMagneticDisk	86
msoShapeFlowchartManualInput	71
msoShapeFlowchartManualOperation	72
msoShapeFlowchartMerge	82
msoShapeFlowchartMultidocument	68
msoShapeFlowchartOffpageConnector	74
msoShapeFlowchartOr	78
msoShapeFlowchartPredefinedProcess	65
msoShapeFlowchartPreparation	70
msoShapeFlowchartProcess	61
msoShapeFlowchartPunchedTape	76
msoShapeFlowchartSequentialAccessStorage	85
msoShapeFlowchartSort	80
msoShapeFlowchartStoredData	83
msoShapeFlowchartSummingJunction	77
msoShapeFlowchartTerminator	69
msoShapeFoldedCorner	16
msoShapeHeart	21
msoShapeHexagon	10
msoShapeHorizontalScroll	102
msoShapeIsoscelesTriangle	7
msoShapeLeftArrow	34
msoShapeLeftArrowCallout	54
msoShapeLeftBrace	31
msoShapeLeftBracket	29
msoShapeLeftRightArrow	37
msoShapeLeftRightArrowCallout	57
msoShapeLeftRightUpArrow	40

msoShapeLeftUpArrow	43
msoShapeLightningBolt	22
msoShapeLineCallout1	109
msoShapeLineCallout1AccentBar	113
msoShapeLineCallout 1 Border and Accent Bar	121
msoShapeLineCallout1NoBorder	117
msoShapeLineCallout2	110
msoShapeLineCallout2AccentBar	114
msoShapeLineCallout2BorderandAccentBar	122
msoShapeLineCallout2NoBorder	118
msoShapeLineCallout3	111
msoShapeLineCallout3AccentBar	115
msoShapeLineCallout 3 Border and Accent Bar	123
msoShapeLineCallout3NoBorder	119
msoShapeLineCallout4	112
msoShapeLineCallout4AccentBar	116
msoShapeLineCallout 4 Border and Accent Bar	124
msoShapeLineCallout4NoBorder	120
msoShapeMixed	-2
msoShapeMoon	24
msoShapeNoSymbol	19
msoShapeNotchedRightArrow	50
msoShapeNotPrimitive	138
msoShapeOctagon	6
msoShapeOval	9
msoShapeOvalCallout	107
msoShapeParallelogram	2
msoShapePentagon	51
msoShapePlaque	28
msoShapeQuadArrow	39
msoShapeQuadArrowCallout	59
msoShapeRectangle	1
msoShapeRectangularCallout	105

msoShapeRegularPentagon	12
msoShapeRightArrow	33
msoShapeRightArrowCallout	53
msoShapeRightBrace	32
msoShapeRightBracket	30
msoShapeRightTriangle	8
msoShapeRoundedRectangle	5
msoShapeRoundedRectangularCallout	106
msoShapeSmileyFace	17
msoShapeStripedRightArrow	49
msoShapeSun	23
msoShapeTrapezoid	3
msoShapeUpArrow	35
msoShapeUpArrowCallout	55
msoShapeUpDownArrow	38
msoShapeUpDownArrowCallout	58
msoShapeUpRibbon	97
msoShapeUTurnArrow	42
msoShapeVerticalScroll	101
msoShapeWave	103

#### **MsoBalloonButtonType**

Constant	Value
msoBalloonButtonAbort	-8
msoBalloonButtonBack	-5
msoBalloonButtonCancel	-2
msoBalloonButtonClose	-12
msoBalloonButtonIgnore	-9
msoBalloonButtonNext	-6
msoBalloonButtonNo	-4
msoBalloonButtonNull	0
msoBalloonButtonOK	-1

- msoBalloonButtonOptions -14
- msoBalloonButtonRetry -7
- msoBalloonButtonSearch -10
- msoBalloonButtonSnooze -11
- msoBalloonButtonTips -13
- msoBalloonButtonYes -3
- msoBalloonButtonYesToAll -15

#### **MsoBalloonErrorType**

Constant	Value
msoBalloonErrorBadCharacter	8
msoBalloonErrorBadPictureRef	4
msoBalloonErrorBadReference	5
msoBalloonErrorButtonlessModal	6
msoBalloonErrorButtonModeless	7
msoBalloon Error Char Not Topmost For Modal	10
msoBalloonErrorCOMFailure	9
msoBalloonErrorNone	0
msoBalloonErrorOther	1
msoBalloonErrorOutOfMemory	3
msoBalloonErrorTooBig	2
msoBalloonErrorTooManyControls	11

#### **MsoBalloonType**

ConstantValuemsoBalloonTypeBullets1msoBalloonTypeButtons0msoBalloonTypeNumbers2

**MsoBarPosition** 

Constant Value

msoBarBottom	3
msoBarFloating	4
msoBarLeft	0
msoBarMenuBar	6
msoBarPopup	5
msoBarRight	2
msoBarTop	1

#### **MsoBarProtection**

Constant	Value
msoBarNoChangeDock	16
msoBarNoChangeVisible	8
msoBarNoCustomize	1
msoBarNoHorizontalDock	64
msoBarNoMove	4
msoBarNoProtection	0
msoBarNoResize	2
msoBarNoVerticalDock	32

#### **MsoBarRow**

ConstantValuemsoBarRowFirst0msoBarRowLast-1

#### **MsoBarType**

 Constant
 Value

 msoBarTypeMenuBar
 1

 msoBarTypeNormal
 0

 msoBarTypePopup
 2

<u>MsoBlackWhiteMode</u>

Constant	Value
msoBlackWhiteAutomatic	1
msoBlackWhiteBlack	8
msoBlackWhiteBlackTextAndLine	6
msoBlackWhiteDontShow	10
msoBlackWhiteGrayOutline	5
msoBlackWhiteGrayScale	2
msoBlackWhiteHighContrast	7
msoBlackWhiteInverseGrayScale	4
msoBlackWhiteLightGrayScale	3
msoBlackWhiteMixed	-2
msoBlackWhiteWhite	9

#### **MsoButtonSetType**

Constant	Value
msoButtonSetAbortRetryIgnore	10
msoButtonSetBackClose	6
msoButtonSetBackNextClose	8
msoButtonSetBackNextSnooze	12
msoButtonSetCancel	2
msoButtonSetNextClose	7
msoButtonSetNone	0
msoButtonSetOK	1
msoButtonSetOkCancel	3
msoButtonSetRetryCancel	9
msoButtonSetSearchClose	11
msoButtonSetTipsOptionsClose	13
msoButtonSetYesAllNoCancel	14
msoButtonSetYesNo	4
msoButtonSetYesNoCancel	5

**MsoButtonState** 

## ConstantValuemsoButtonDown-1msoButtonMixed2msoButtonUp0

#### **MsoButtonStyle**

Constant	Value
msoButtonAutomatic	0
msoButtonCaption	2
msoButtonIcon	1
msoButtonIconAndCaption	3
msoButtonIconAndCaptionBelow	11
msoButtonIconAndWrapCaption	7
msoButtonIconAndWrapCaptionBelow	15
msoButtonWrapCaption	14

#### **MsoCalloutAngleType**

Constant	Value
msoCalloutAngle30	2
msoCalloutAngle45	3
msoCalloutAngle60	4
msoCalloutAngle90	5
msoCalloutAngleAutomatic	1
msoCalloutAngleMixed	-2

#### **MsoCalloutDropType**

Constant	Value
msoCalloutDropBottom	4
	2

- msoCalloutDropCenter 3
- msoCalloutDropCustom 1
- msoCalloutDropMixed -2

#### msoCalloutDropTop 2

#### **MsoCalloutType**

 Constant
 Value

 msoCalloutFour
 4

 msoCalloutMixed
 -2

 msoCalloutOne
 1

 msoCalloutThree
 3

 msoCalloutTwo
 2

#### <u>MsoCharacterSet</u>

Constant	Value
msoCharacterSetArabic	1
msoCharacterSetCyrillic	2
mso Character SetEnglish Western European Other Latin Script the set of the	: 3
msoCharacterSetGreek	4
msoCharacterSetHebrew	5
msoCharacterSetJapanese	6
msoCharacterSetKorean	7
msoCharacterSetMultilingualUnicode	8
msoCharacterSetSimplifiedChinese	9
msoCharacterSetThai	10
msoCharacterSetTraditionalChinese	11
msoCharacterSetVietnamese	12

#### **MsoColorType**

Constant	Value
msoColorTypeCMS	4
msoColorTypeCMYK	3
msoColorTypeInk	5
msoColorTypeMixed	-2

msoColorTypeRGB 1 msoColorTypeScheme 2

#### **MsoComboStyle**

ConstantValuemsoComboLabel1msoComboNormal0

#### MsoCommandBarButtonHyperlinkType

Constant	Value
mso Command Bar Button Hyperlink Insert Picture	2
msoCommandBarButtonHyperlinkNone	0
msoCommandBarButtonHyperlinkOpen	1

#### **MsoCondition**

Constant	Value
msoConditionAnyNumberBetween	34
msoConditionAnytime	25
msoConditionAnytimeBetween	26
msoConditionAtLeast	36
msoConditionAtMost	35
msoConditionBeginsWith	11
msoConditionDoesNotEqual	33
msoConditionEndsWith	12
msoConditionEquals	32
msoConditionEqualsCompleted	66
msoConditionEqualsDeferred	68
msoConditionEqualsHigh	60
msoConditionEqualsInProgress	65
msoConditionEqualsLow	58
msoConditionEqualsNormal	59

msoConditionEqualsNotStarted	64
msoConditionEqualsWaitingForSomeoneElse	67
msoConditionFileTypeAllFiles	1
msoConditionFileTypeBinders	6
msoConditionFileTypeCalendarItem	45
msoConditionFileTypeContactItem	46
msoConditionFileTypeDatabases	7
msoConditionFileTypeDataConnectionFiles	51
msoConditionFileTypeDesignerFiles	56
msoConditionFileTypeDocumentImagingFiles	54
msoConditionFileTypeExcelWorkbooks	4
msoConditionFileTypeJournalItem	48
msoConditionFileTypeMailItem	44
msoConditionFileTypeNoteItem	47
msoConditionFileTypeOfficeFiles	2
msoConditionFileTypeOutlookItems	43
msoConditionFileTypePhotoDrawFiles	50
msoConditionFileTypePowerPointPresentations	5
msoConditionFileTypeProjectFiles	53
msoConditionFileTypePublisherFiles	52
msoConditionFileTypeTaskItem	49
msoConditionFileTypeTemplates	8
msoConditionFileTypeVisioFiles	55
msoConditionFileTypeWebPages	57
msoConditionFileTypeWordDocuments	3
msoConditionFreeText	42
msoConditionIncludes	9
msoConditionIncludesFormsOf	41
msoConditionIncludesNearEachOther	13
msoConditionIncludesPhrase	10
msoConditionInTheLast	31
msoConditionInTheNext	30
msoConditionIsExactly	14

msoConditionIsNo	40
msoConditionIsNot	15
msoConditionIsYes	39
msoConditionLastMonth	22
msoConditionLastWeek	19
msoConditionLessThan	38
msoConditionMoreThan	37
msoConditionNextMonth	24
msoConditionNextWeek	21
msoConditionNotEqualToCompleted	71
msoConditionNotEqualToDeferred	73
msoConditionNotEqualToHigh	63
msoConditionNotEqualToInProgress	70
msoConditionNotEqualToLow	61
msoConditionNotEqualToNormal	62
msoConditionNotEqualToNotStarted	69
mso Condition Not Equal To Waiting For Some one Else the second straight	72
msoConditionOn	27
msoConditionOnOrAfter	28
msoConditionOnOrBefore	29
msoConditionThisMonth	23
msoConditionThisWeek	20
msoConditionToday	17
msoConditionTomorrow	18
msoConditionYesterday	16

## **MsoConnector**

ConstantValuemsoConnectorAnd1msoConnectorOr2

**MsoConnectorType** 

 Constant
 Value

 msoConnectorCurve
 3

 msoConnectorElbow
 2

 msoConnectorStraight
 1

 msoConnectorTypeMixed
 2

## **MsoControlOLEUsage**

Constant	Value
msoControlOLEUsageBoth	3
msoControlOLEUsageClient	2
msoControlOLEUsageNeither	0
msoControlOLEUsageServer	1

## **MsoControlType**

Constant	Value
msoControlActiveX	22
msoControlAutoCompleteCombo	26
msoControlButton	1
msoControlButtonDropdown	5
msoControlButtonPopup	12
msoControlComboBox	4
msoControlCustom	0
msoControlDropdown	3
msoControlEdit	2
msoControlExpandingGrid	16
msoControlGauge	19
msoControlGenericDropdown	8
msoControlGraphicCombo	20
msoControlGraphicDropdown	9
msoControlGraphicPopup	11
msoControlGrid	18
msoControlLabel	15

msoControlLabelEx	24
msoControlOCXDropdown	7
msoControlPane	21
msoControlPopup	10
msoControlSpinner	23
mso Control Split Button MRUP op up	14
msoControlSplitButtonPopup	13
msoControlSplitDropdown	6
msoControlSplitExpandingGrid	17
msoControlWorkPane	25

## <u>MsoDiagramNodeType</u>

Constant	Value
msoDiagramAssistant	2
msoDiagramNode	1

## **MsoDiagramType**

Constant	Value
msoDiagramCycle	2
msoDiagramMixed	-2
msoDiagramOrgChart	1
msoDiagramPyramid	4
msoDiagramRadial	3
msoDiagramTarget	6
msoDiagramVenn	5

## **MsoDistributeCmd**

Constant	Value
msoDistributeHorizontally	0
msoDistributeVertically	1

## **MsoDocProperties**

Constant	Value
msoPropertyTypeBoolean	2
msoPropertyTypeDate	3
msoPropertyTypeFloat	5
msoPropertyTypeNumber	1
msoPropertyTypeString	4

## <u>MsoEditingType</u>

Constant	Value
msoEditingAuto	0
msoEditingCorner	1
msoEditingSmooth	2
msoEditingSymmetric	3

## **MsoEncoding**

Constant	Value
msoEncodingArabic	1256
msoEncodingArabicASMO	708
msoEncodingArabicAutoDetect	51256
msoEncodingArabicTransparentASMO	720
msoEncodingAutoDetect	50001
msoEncodingBaltic	1257
msoEncodingCentralEuropean	1250
msoEncodingCyrillic	1251
msoEncodingCyrillicAutoDetect	51251
msoEncodingEBCDICArabic	20420
msoEncodingEBCDICDenmarkNorway	20277
msoEncodingEBCDICFinlandSweden	20278
msoEncodingEBCDICFrance	20297
msoEncodingEBCDICGermany	20273

msoEncodingEBCDICGreek	20423
msoEncodingEBCDICGreekModern	875
msoEncodingEBCDICHebrew	20424
msoEncodingEBCDICIcelandic	20871
msoEncodingEBCDICInternational	500
msoEncodingEBCDICItaly	20280
msoEncodingEBCDICJapaneseKatakanaExtended	20290
msoEncodingEBCDICJapaneseKatakanaExtendedAndJapanese	50930
msoEncodingEBCDICJapaneseLatinExtendedAndJapanese	50939
msoEncodingEBCDICKoreanExtended	20833
msoEncodingEBCDICKoreanExtendedAndKorean	50933
msoEncodingEBCDICLatinAmericaSpain	20284
msoEncodingEBCDICMultilingualROECELatin2	870
msoEncodingEBCDICRussian	20880
msoEncodingEBCDICSerbianBulgarian	21025
mso Encoding EBCDICS implified Chinese Extended And Simplified Chinese Extended And Simplifi	50935
msoEncodingEBCDICThai	20838
msoEncodingEBCDICTurkish	20905
msoEncodingEBCDICTurkishLatin5	1026
msoEncodingEBCDICUnitedKingdom	20285
msoEncodingEBCDICUSCanada	37
msoEncodingEBCDICUSCanadaAndJapanese	50931
msoEncodingEBCDICUSCanadaAndTraditionalChinese	50937
msoEncodingEUCChineseSimplifiedChinese	51936
msoEncodingEUCJapanese	51932
msoEncodingEUCKorean	51949
msoEncodingEUCTaiwaneseTraditionalChinese	51950
msoEncodingEuropa3	29001
msoEncodingExtAlphaLowercase	21027
msoEncodingGreek	1253
msoEncodingGreekAutoDetect	51253
msoEncodingHebrew	1255
msoEncodingHZGBSimplifiedChinese	52936

msoEncodingIA5German	20106
msoEncodingIA5IRV	20105
msoEncodingIA5Norwegian	20108
msoEncodingIA5Swedish	20107
msoEncodingISCIIAssamese	57006
msoEncodingISCIIBengali	57003
msoEncodingISCIIDevanagari	57002
msoEncodingISCIIGujarati	57010
msoEncodingISCIIKannada	57008
msoEncodingISCIIMalayalam	57009
msoEncodingISCIIOriya	57007
msoEncodingISCIIPunjabi	57011
msoEncodingISCIITamil	57004
msoEncodingISCIITelugu	57005
msoEncodingISO2022CNSimplifiedChinese	50229
msoEncodingISO2022CNTraditionalChinese	50227
msoEncodingISO2022JPJISX02011989	50222
msoEncodingISO2022JPJISX02021984	50221
msoEncodingISO2022JPNoHalfwidthKatakana	50220
msoEncodingISO2022KR	50225
msoEncodingISO6937NonSpacingAccent	20269
msoEncodingISO885915Latin9	28605
msoEncodingISO88591Latin1	28591
msoEncodingISO88592CentralEurope	28592
msoEncodingISO88593Latin3	28593
msoEncodingISO88594Baltic	28594
msoEncodingISO88595Cyrillic	28595
msoEncodingISO88596Arabic	28596
msoEncodingISO88597Greek	28597
msoEncodingISO88598Hebrew	28598
msoEncodingISO88599Turkish	28599
msoEncodingJapaneseAutoDetect	50932
msoEncodingJapaneseShiftJIS	932

mas En as din «VOIOD	20000
msoEncodingKOI8R	20866
msoEncodingKOI8U	21866
msoEncodingKorean	949
msoEncodingKoreanAutoDetect	50949
msoEncodingKoreanJohab	1361
msoEncodingMacArabic	10004
msoEncodingMacCroatia	10082
msoEncodingMacCyrillic	10007
msoEncodingMacGreek1	10006
msoEncodingMacHebrew	10005
msoEncodingMacIcelandic	10079
msoEncodingMacJapanese	10001
msoEncodingMacKorean	10003
msoEncodingMacLatin2	10029
msoEncodingMacRoman	10000
msoEncodingMacRomania	10010
msoEncodingMacSimplifiedChineseGB2312	10008
msoEncodingMacTraditionalChineseBig5	10002
msoEncodingMacTurkish	10081
msoEncodingMacUkraine	10017
msoEncodingOEMArabic	864
msoEncodingOEMBaltic	775
msoEncodingOEMCanadianFrench	863
msoEncodingOEMCyrillic	855
msoEncodingOEMCyrillicII	866
msoEncodingOEMGreek437G	737
msoEncodingOEMHebrew	862
msoEncodingOEMIcelandic	861
msoEncodingOEMModernGreek	869
msoEncodingOEMMultilingualLatinI	850
msoEncodingOEMMultilingualLatinII	852
msoEncodingOEMNordic	865
msoEncodingOEMPortuguese	860

msoEncodingOEMTurkish	857
msoEncodingOEMUnitedStates	437
msoEncodingSimplifiedChineseAutoDetect	50936
msoEncodingSimplifiedChineseGBK	936
msoEncodingT61	20261
msoEncodingTaiwanCNS	20000
msoEncodingTaiwanEten	20002
msoEncodingTaiwanIBM5550	20003
msoEncodingTaiwanTCA	20001
msoEncodingTaiwanTeleText	20004
msoEncodingTaiwanWang	20005
msoEncodingThai	874
msoEncodingTraditionalChineseAutoDetect	50950
msoEncodingTraditionalChineseBig5	950
msoEncodingTurkish	1254
msoEncodingUnicodeBigEndian	1201
msoEncodingUnicodeLittleEndian	1200
msoEncodingUSASCII	20127
msoEncodingUTF7	65000
msoEncodingUTF8	65001
msoEncodingVietnamese	1258
msoEncodingWestern	1252

**MsoExtraInfoMethod** 

ConstantValuemsoMethodGet0msoMethodPost1

## **MsoExtrusionColorType**

Constant	Value
msoExtrusionColorAutomatic	1
msoExtrusionColorCustom	2

#### msoExtrusionColorTypeMixed -2

#### MsoFarEastLineBreakLanguageID

Constant	Value
MsoFarEastLineBreakLanguageJapanese	1041
MsoFarEastLineBreakLanguageKorean	1042
MsoFarEastLineBreakLanguageSimplifiedChinese	2052
MsoFarEastLineBreakLanguageTraditionalChinese	1028

### **MsoFeatureInstall**

Constant	Value
msoFeatureInstallNone	0
msoFeatureInstallOnDemand	1
msoFeatureInstallOnDemandWithUI	2

### **MsoFileDialogType**

Constant	Value
msoFileDialogFilePicker	3
msoFileDialogFolderPicker	4
msoFileDialogOpen	1
msoFileDialogSaveAs	2

### **MsoFileDialogView**

#### Constant Value

msoFileDialogViewDetails2msoFileDialogViewLargeIcons6msoFileDialogViewList1msoFileDialogViewPreview4msoFileDialogViewProperties3msoFileDialogViewSmallIcons7

msoFileDialogViewThumbnail 5 msoFileDialogViewTiles 9 msoFileDialogViewWebView 8

## **MsoFileFindListBy**

ConstantValuemsoListbyName1msoListbyTitle2

### **MsoFileFindOptions**

ConstantValuemsoOptionsAdd2msoOptionsNew1msoOptionsWithin3

#### **MsoFileFindSortBy**

Constant	Value
msoFileFindSortbyAuthor	1
msoFileFindSortbyDateCreated	2
msoFileFindSortbyDateSaved	4
msoFileFindSortbyFileName	5
msoFileFindSortbyLastSavedBy	3
msoFileFindSortbySize	6
msoFileFindSortbyTitle	7

## **MsoFileFindView**

Constant	Value
msoViewFileInfo	1
msoViewPreview	2
msoViewSummaryInfo	3

## **MsoFileNewAction**

## Constant Value

msoCreateNewFile	e 1
msoEditFile	0
msoOpenFile	2

## **MsoFileNewSection**

Constant	Value
msoBottomSection	4
msoNew	1
msoNewfromExistingFile	2
msoNewfromTemplate	3
msoOpenDocument	0

## **MsoFileType**

Constant	Value
msoFileTypeAllFiles	1
msoFileTypeBinders	6
msoFileTypeCalendarItem	11
msoFileTypeContactItem	12
msoFileTypeDatabases	7
msoFileTypeDataConnectionFiles	17
msoFileTypeDesignerFiles	22
msoFileTypeDocumentImagingFiles	20
msoFileTypeExcelWorkbooks	4
msoFileTypeJournalItem	14
msoFileTypeMailItem	10
msoFileTypeNoteItem	13
msoFileTypeOfficeFiles	2
msoFileTypeOutlookItems	9
msoFileTypePhotoDrawFiles	16

msoFileTypePowerPointPresentations 5	
msoFileTypeProjectFiles	19
msoFileTypePublisherFiles	18
msoFileTypeTaskItem	15
msoFileTypeTemplates	8
msoFileTypeVisioFiles	21
msoFileTypeWebPages	23
msoFileTypeWordDocuments	3

## <u>MsoFillType</u>

Constant	Value
msoFillBackground	5
msoFillGradient	3
msoFillMixed	-2
msoFillPatterned	2
msoFillPicture	6
msoFillSolid	1
msoFillTextured	4

## **MsoFilterComparison**

Constant	Value
msoFilterComparisonContains	8
msoFilterComparisonEqual	0
msoFilterComparisonGreaterThan	3
msoFilter Comparison Greater Than Equal	5
msoFilterComparisonIsBlank	6
msoFilterComparisonIsNotBlank	7
msoFilterComparisonLessThan	2
msoFilterComparisonLessThanEqual	4
msoFilterComparisonNotContains	9
msoFilterComparisonNotEqual	1

### **MsoFilterConjunction**

#### Constant Value

msoFilterConjunctionAnd 0 msoFilterConjunctionOr 1

## **MsoFlipCmd**

ConstantValuemsoFlipHorizontal0msoFlipVertical1

### **MsoGradientColorType**

- Constant Value
- msoGradientColorMixed -2
- msoGradientOneColor 1
- msoGradientPresetColors 3
- msoGradientTwoColors 2

## **MsoGradientStyle**

Constant	Value
msoGradientDiagonalDown	4

- msoGradientDiagonalUp 3
- msoGradientFromCenter 7
- msoGradientFromCorner 5
- msoGradientFromTitle 6
- msoGradientHorizontal 1
- msoGradientMixed -2
- msoGradientVertical 2

<u>MsoHorizontalAnchor</u>

ConstantValuemsoAnchorCenter2msoAnchorNone1msoHorizontalAnchorMixet-2

## **MsoHTMLProjectOpen**

ConstantValuemsoHTMLProjectOpenSourceView1msoHTMLProjectOpenTextView2

### **MsoHTMLProjectState**

Constant	Value
msoHTMLProjectStateDocumentLocked	1
msoHTMLProjectStateDocumentProjectUnlocked	3
msoHTMLProjectStateProjectLocked	2

**MsoHyperlinkType** 

Constant	Value
msoHyperlinkInlineShape	2
msoHyperlinkRange	0
msoHyperlinkShape	1

### **MsoIconType**

Constant	Value
msoIconAlert	2
msoIconAlertCritical	7
msoIconAlertInfo	4
msoIconAlertQuery	6
msoIconAlertWarning	5
msoIconNone	0

## msoIconTip

### 3

# **MsoLanguageID**

Constant	Value
msoLanguageIDAfrikaans	1078
msoLanguageIDAlbanian	1052
msoLanguageIDAmharic	1118
msoLanguageIDArabic	1025
msoLanguageIDArabicAlgeria	5121
msoLanguageIDArabicBahrain	15361
msoLanguageIDArabicEgypt	3073
msoLanguageIDArabicIraq	2049
msoLanguageIDArabicJordan	11265
msoLanguageIDArabicKuwait	13313
msoLanguageIDArabicLebanon	12289
msoLanguageIDArabicLibya	4097
msoLanguageIDArabicMorocco	6145
msoLanguageIDArabicOman	8193
msoLanguageIDArabicQatar	16385
msoLanguageIDArabicSyria	10241
msoLanguageIDArabicTunisia	7169
msoLanguageIDArabicUAE	14337
msoLanguageIDArabicYemen	9217
msoLanguageIDArmenian	1067
msoLanguageIDAssamese	1101
msoLanguageIDAzeriCyrillic	2092
msoLanguageIDAzeriLatin	1068
msoLanguageIDBasque	1069
msoLanguageIDBelgianDutch	2067
msoLanguageIDBelgianFrench	2060
msoLanguageIDBengali	1093
msoLanguageIDBosnian	4122

msoLanguageIDBrazilianPortuguese	1046
msoLanguageIDBulgarian	1026
msoLanguageIDBurmese	1109
msoLanguageIDByelorussian	1059
msoLanguageIDCatalan	1027
msoLanguageIDCherokee	1116
msoLanguageIDChineseHongKong	3076
msoLanguageIDChineseHongKongSAR	3076
msoLanguageIDChineseMacao	5124
msoLanguageIDChineseMacaoSAR	5124
msoLanguageIDChineseSingapore	4100
msoLanguageIDCroatian	1050
msoLanguageIDCzech	1029
msoLanguageIDDanish	1030
msoLanguageIDDivehi	1125
msoLanguageIDDutch	1043
msoLanguageIDDzongkhaBhutan	2129
msoLanguageIDEdo	1126
msoLanguageIDEnglishAUS	3081
msoLanguageIDEnglishBelize	10249
msoLanguageIDEnglishCanadian	4105
msoLanguageIDEnglishCaribbean	9225
msoLanguageIDEnglishIndonesia	14345
msoLanguageIDEnglishIreland	6153
msoLanguageIDEnglishJamaica	8201
msoLanguageIDEnglishNewZealand	5129
msoLanguageIDEnglishPhilippines	13321
msoLanguageIDEnglishSouthAfrica	7177
msoLanguageIDEnglishTrinidad	11273
msoLanguageIDEnglishTrinidadTobago	11273
msoLanguageIDEnglishUK	2057
msoLanguageIDEnglishUS	1033
msoLanguageIDEnglishZimbabwe	12297

msoLanguageIDEstonian	1061
msoLanguageIDFaeroese	1080
msoLanguageIDFarsi	1065
msoLanguageIDFilipino	1124
msoLanguageIDFinnish	1035
msoLanguageIDFrench	1036
msoLanguageIDFrenchCameroon	11276
msoLanguageIDFrenchCanadian	3084
msoLanguageIDFrenchCotedIvoire	12300
msoLanguageIDFrenchHaiti	15372
msoLanguageIDFrenchLuxembourg	5132
msoLanguageIDFrenchMali	13324
msoLanguageIDFrenchMonaco	6156
msoLanguageIDFrenchMorocco	14348
msoLanguageIDFrenchReunion	8204
msoLanguageIDFrenchSenegal	10252
msoLanguageIDFrenchWestIndies	7180
msoLanguageIDFrenchZaire	9228
msoLanguageIDFrisianNetherlands	1122
msoLanguageIDFulfulde	1127
msoLanguageIDGaelicIreland	2108
msoLanguageIDGaelicScotland	1084
msoLanguageIDGalician	1110
msoLanguageIDGeorgian	1079
msoLanguageIDGerman	1031
msoLanguageIDGermanAustria	3079
msoLanguageIDGermanLiechtenstein	5127
msoLanguageIDGermanLuxembourg	4103
msoLanguageIDGreek	1032
msoLanguageIDGuarani	1140
msoLanguageIDGujarati	1095
msoLanguageIDHausa	1128
msoLanguageIDHawaiian	1141

msoLanguageIDHebrew	1037
msoLanguageIDHindi	1081
msoLanguageIDHungarian	1038
msoLanguageIDIbibio	1129
msoLanguageIDIcelandic	1039
msoLanguageIDIgbo	1136
msoLanguageIDIndonesian	1057
msoLanguageIDInuktitut	1117
msoLanguageIDItalian	1040
msoLanguageIDJapanese	1041
msoLanguageIDKannada	1099
msoLanguageIDKanuri	1137
msoLanguageIDKashmiri	1120
msoLanguageIDKashmiriDevanagari	2144
msoLanguageIDKashmiriIndia	2144
msoLanguageIDKazakh	1087
msoLanguageIDKhmer	1107
msoLanguageIDKirghiz	1088
msoLanguageIDKonkani	1111
msoLanguageIDKorean	1042
msoLanguageIDKyrgyz	1088
msoLanguageIDLao	1108
msoLanguageIDLatin	1142
msoLanguageIDLatvian	1062
msoLanguageIDLithuanian	1063
msoLanguageIDMacedonian	1071
msoLanguageIDMalayalam	1100
msoLanguageIDMalayBruneiDarussalam	2110
msoLanguageIDMalaysian	1086
msoLanguageIDMaltese	1082
msoLanguageIDManipuri	1112
msoLanguageIDMaori	1153
msoLanguageIDMarathi	1102

msoLanguageIDMexicanSpanish	2058
msoLanguageIDMixed	-2
msoLanguageIDMongolian	1104
msoLanguageIDNepali	1121
msoLanguageIDNone	0
msoLanguageIDNoProofing	1024
msoLanguageIDNorwegianBokmol	1044
msoLanguageIDNorwegianNynorsk	2068
msoLanguageIDOriya	1096
msoLanguageIDOromo	1138
msoLanguageIDPashto	1123
msoLanguageIDPolish	1045
msoLanguageIDPortuguese	2070
msoLanguageIDPunjabi	1094
msoLanguageIDRhaetoRomanic	1047
msoLanguageIDRomanian	1048
msoLanguageIDRomanianMoldova	2072
msoLanguageIDRussian	1049
msoLanguageIDRussianMoldova	2073
msoLanguageIDSamiLappish	1083
msoLanguageIDSanskrit	1103
msoLanguageIDSerbianCyrillic	3098
msoLanguageIDSerbianLatin	2074
msoLanguageIDSesotho	1072
msoLanguageIDSimplifiedChinese	2052
msoLanguageIDSindhi	1113
msoLanguageIDSindhiPakistan	2137
msoLanguageIDSinhalese	1115
msoLanguageIDSlovak	1051
msoLanguageIDSlovenian	1060
msoLanguageIDSomali	1143
msoLanguageIDSorbian	1070
msoLanguageIDSpanish	1034

msoLanguageIDSpanishArgentina	11274
msoLanguageIDSpanishBolivia	16394
msoLanguageIDSpanishChile	13322
msoLanguageIDSpanishColombia	9226
msoLanguageIDSpanishCostaRica	5130
msoLanguageIDSpanishDominicanRepublic	27178
msoLanguageIDSpanishEcuador	12298
msoLanguageIDSpanishElSalvador	17418
msoLanguageIDSpanishGuatemala	4106
msoLanguageIDSpanishHonduras	18442
msoLanguageIDSpanishModernSort	3082
msoLanguageIDSpanishNicaragua	19466
msoLanguageIDSpanishPanama	6154
msoLanguageIDSpanishParaguay	15370
msoLanguageIDSpanishPeru	10250
msoLanguageIDSpanishPuertoRico	20490
msoLanguageIDSpanishUruguay	14346
msoLanguageIDSpanishVenezuela	8202
msoLanguageIDSutu	1072
msoLanguageIDSwahili	1089
msoLanguageIDSwedish	1053
msoLanguageIDSwedishFinland	2077
msoLanguageIDSwissFrench	4108
msoLanguageIDSwissGerman	2055
msoLanguageIDSwissItalian	2064
msoLanguageIDSyriac	1114
msoLanguageIDTajik	1064
msoLanguageIDTamazight	1119
msoLanguageIDTamazightLatin	2143
msoLanguageIDTamil	1097
msoLanguageIDTatar	1092
msoLanguageIDTelugu	1098
msoLanguageIDThai	1054

msoLanguageIDTibetan	1105
msoLanguageIDTigrignaEritrea	2163
msoLanguageIDTigrignaEthiopic	1139
msoLanguageIDTraditionalChinese	1028
msoLanguageIDTsonga	1073
msoLanguageIDTswana	1074
msoLanguageIDTurkish	1055
msoLanguageIDTurkmen	1090
msoLanguageIDUkrainian	1058
msoLanguageIDUrdu	1056
msoLanguageIDUzbekCyrillic	2115
msoLanguageIDUzbekLatin	1091
msoLanguageIDVenda	1075
msoLanguageIDVietnamese	1066
msoLanguageIDWelsh	1106
msoLanguageIDXhosa	1076
msoLanguageIDYi	1144
msoLanguageIDYiddish	1085
msoLanguageIDYoruba	1130
msoLanguageIDZulu	1077

## **MsoLastModified**

#### Constant

Value

msoLastModifiedAnyTime 7 msoLastModifiedLastMonth 3 msoLastModifiedThisMonth 4 msoLastModifiedThisWeek 4 msoLastModifiedToday 2 msoLastModifiedYesterday 1

<u>MsoLineDashStyle</u>

Value Constant msoLineDash 4 msoLineDashDot 5 msoLineDashDotDot 6 msoLineDashStyleMixed -2 msoLineLongDash 7 msoLineLongDashDot 8 3 msoLineRoundDot msoLineSolid 1 2 msoLineSquareDot

## **MsoLineStyle**

Constant	Value
msoLineSingle	1
msoLineStyleMixed	-2
msoLineThickBetweenThin	5
msoLineThickThin	4
msoLineThinThick	3
msoLineThinThin	2

### **MsoMenuAnimation**

#### Constant

Value

- msoMenuAnimationNone 0
- msoMenuAnimationRandom 1
- msoMenuAnimationSlide 3
- msoMenuAnimationUnfold 2

### **MsoMixedType**

ConstantValuemsoIntegerMixed32768msoSingleMixed-2147483648

## **MsoModeType**

#### Constant Value

msoModeAutoDown 1 msoModeModal 0

msoModeModeless 2

#### <u>MsoMoveRow</u>

ConstantValuemsoMoveRowFirst-4msoMoveRowNbr-1msoMoveRowNext-2msoMoveRowPrev-3

### **MsoOLEMenuGroup**

Constant	Value
msoOLEMenuGroupContainer	2
msoOLEMenuGroupEdit	1
msoOLEMenuGroupFile	0
msoOLEMenuGroupHelp	5
msoOLEMenuGroupNone	-1
msoOLEMenuGroupObject	3
msoOLEMenuGroupWindow	4

### **MsoOrgChartLayoutType**

#### Constant

#### Value

- msoOrgChartLayoutBothHanging 2
- msoOrgChartLayoutLeftHanging 3
- msoOrgChartLayoutMixed -2
- $msoOrgChartLayoutRightHanging\ 4$
- msoOrgChartLayoutStandard 1

## **MsoOrgChartOrientation**

Constant

Value

msoOrgChartOrientationMixed -2 msoOrgChartOrientationVertical 1

## **MsoOrientation**

Constant	Value
msoOrientationHorizontal	1
msoOrientationMixed	-2
msoOrientationVertical	2

## **MsoPatternType**

Constant	Value
msoPattern10Percent	2
msoPattern20Percent	3
msoPattern25Percent	4
msoPattern30Percent	5
msoPattern40Percent	6
msoPattern50Percent	7
msoPattern5Percent	1
msoPattern60Percent	8
msoPattern70Percent	9
msoPattern75Percent	10
msoPattern80Percent	11
msoPattern90Percent	12
msoPatternDarkDownwardDiagonal	15
msoPatternDarkHorizontal	13
msoPatternDarkUpwardDiagonal	16
msoPatternDarkVertical	14
msoPattern Dashed Downward Diagonal	28
msoPatternDashedHorizontal	32

msoPatternDashedUpwardDiagonal	27
msoPatternDashedVertical	31
msoPatternDiagonalBrick	40
msoPatternDivot	46
msoPatternDottedDiamond	24
msoPatternDottedGrid	45
msoPatternHorizontalBrick	35
msoPatternLargeCheckerBoard	36
msoPatternLargeConfetti	33
msoPatternLargeGrid	34
msoPatternLightDownwardDiagonal	21
msoPatternLightHorizontal	19
msoPatternLightUpwardDiagonal	22
msoPatternLightVertical	20
msoPatternMixed	-2
msoPatternNarrowHorizontal	30
msoPatternNarrowVertical	29
msoPatternOutlinedDiamond	41
msoPatternPlaid	42
msoPatternShingle	47
msoPatternSmallCheckerBoard	17
msoPatternSmallConfetti	37
msoPatternSmallGrid	23
msoPatternSolidDiamond	39
msoPatternSphere	43
msoPatternTrellis	18
msoPatternWave	48
msoPatternWeave	44
msoPatternWideDownwardDiagonal	25
msoPatternWideUpwardDiagonal	26
msoPatternZigZag	38

**MsoPermission** 

Constant	Value
msoPermissionChange	15
msoPermissionEdit	2
msoPermissionExtract	8
msoPermissionFullControl	64
msoPermissionObjModel	32
msoPermissionPrint	16
msoPermissionRead	1
msoPermissionSave	4
msoPermissionView	1

## <u>MsoPictureColorType</u>

Constant	Value
msoPictureAutomatic	1
msoPictureBlackAndWhite	3
msoPictureGrayscale	2
msoPictureMixed	-2
msoPictureWatermark	4

## **MsoPresetExtrusionDirection**

Constant	Value
msoExtrusionBottom	2
msoExtrusionBottomLeft	3
msoExtrusionBottomRight	1
msoExtrusionLeft	6
msoExtrusionNone	5
msoExtrusionRight	4
msoExtrusionTop	8
msoExtrusionTopLeft	9
msoExtrusionTopRight	7
msoPresetExtrusionDirectionMixed	-2

## <u>MsoPresetGradientType</u>

Constant	Value
msoGradientBrass	20
msoGradientCalmWater	8
msoGradientChrome	21
msoGradientChromeII	22
msoGradientDaybreak	4
msoGradientDesert	6
msoGradient Early Sunset	1
msoGradientFire	9
msoGradientFog	10
msoGradientGold	18
msoGradientGoldII	19
msoGradientHorizon	5
msoGradientLateSunset	2
msoGradientMahogany	15
msoGradientMoss	11
msoGradientNightfall	3
msoGradientOcean	7
msoGradientParchment	14
msoGradientPeacock	12
msoGradientRainbow	16
msoGradientRainbowII	17
msoGradientSapphire	24
msoGradientSilver	23
msoGradientWheat	13
msoPresetGradientMixed	-2

## **MsoPresetLightingDirection**

Constant	Value
msoLightingBottom	8

msoLightingBottomLeft	7
msoLightingBottomRight	9
msoLightingLeft	4
msoLightingNone	5
msoLightingRight	6
msoLightingTop	2
msoLightingTopLeft	1
msoLightingTopRight	3
msoPresetLightingDirectionMixed	-2

## **MsoPresetLightingSoftness**

Constant	Value
msoLightingBright	3
msoLightingDim	1
msoLightingNormal	2
msoPresetLightingSoftnessMixed	-2

## **MsoPresetMaterial**

Constant	Value
msoMaterialMatte	1
msoMaterialMetal	3
msoMaterialPlastic	2
msoMaterialWireFrame	4
msoPresetMaterialMixed	-2

## MsoPresetTextEffect

Constant	Value
msoTextEffect1	0
msoTextEffect10	9
msoTextEffect11	10
msoTextEffect12	11

- msoTextEffect13 12
- msoTextEffect14 13
- msoTextEffect15 14
- msoTextEffect16 15
- msoTextEffect17 16
- msoTextEffect18 17
- msoTextEffect19 18
- msoTextEffect2 1
- msoTextEffect20 19
- msoTextEffect21 20
- msoTextEffect22 21
- msoTextEffect23 22
- msoTextEffect24 23
- msoTextEffect25 24
- msoTextEffect26 25
- msoTextEffect27 26
- msoTextEffect28 27
- msoTextEffect29 28
- msoTextEffect3 2
- msoTextEffect30 29
- msoTextEffect4 3
- msoTextEffect5 4
- msoTextEffect6 5
- msoTextEffect7 6
- msoTextEffect8 7
- msoTextEffect9 8
- msoTextEffectMixed -2

## **MsoPresetTextEffectShape**

Constant	Value
msoTextEffectShapeArchDownCurve	10
msoTextEffectShapeArchDownPour	14

msoTextEffectShapeArchUpCurve 9 msoTextEffectShapeArchUpPour 13 msoTextEffectShapeButtonCurve 12 msoTextEffectShapeButtonPour 16 msoTextEffectShapeCanDown 20 msoTextEffectShapeCanUp 19 msoTextEffectShapeCascadeDown 40 msoTextEffectShapeCascadeUp 39 msoTextEffectShapeChevronDown 6 msoTextEffectShapeChevronUp 5 msoTextEffectShapeCircleCurve 11 msoTextEffectShapeCirclePour 15 msoTextEffectShapeCurveDown 18 msoTextEffectShapeCurveUp 17 msoTextEffectShapeDeflate 26 msoTextEffectShapeDeflateBottom 28 msoTextEffectShapeDeflateInflate 31 msoTextEffectShapeDeflateInflateDeflate 32 msoTextEffectShapeDeflateTop 30 msoTextEffectShapeDoubleWave1 23 msoTextEffectShapeDoubleWave2 24 36 msoTextEffectShapeFadeDown msoTextEffectShapeFadeLeft 34 msoTextEffectShapeFadeRight 33 msoTextEffectShapeFadeUp 35 msoTextEffectShapeInflate 25 msoTextEffectShapeInflateBottom 27 msoTextEffectShapeInflateTop 29 msoTextEffectShapeMixed -2 msoTextEffectShapePlainText 1 7 msoTextEffectShapeRingInside msoTextEffectShapeRingOutside 8 msoTextEffectShapeSlantDown 38

msoTextEffectShapeSlantUp	37
msoTextEffectShapeStop	2
msoTextEffectShapeTriangleDown	4
msoTextEffectShapeTriangleUp	3
msoTextEffectShapeWave1	21
msoTextEffectShapeWave2	22

## **MsoPresetTexture**

Constant	Value
msoPresetTextureMixed	-2
msoTextureBlueTissuePaper	17
msoTextureBouquet	20
msoTextureBrownMarble	11
msoTextureCanvas	2
msoTextureCork	21
msoTextureDenim	3
msoTextureFishFossil	7
msoTextureGranite	12
msoTextureGreenMarble	9
msoTextureMediumWood	24
msoTextureNewsprint	13
msoTextureOak	23
msoTexturePaperBag	6
msoTexturePapyrus	1
msoTextureParchment	15
msoTexturePinkTissuePaper	18
msoTexturePurpleMesh	19
msoTextureRecycledPaper	14
msoTextureSand	8
msoTextureStationery	16
msoTextureWalnut	22
msoTextureWaterDroplets	5

msoTextureWhiteMarble	10
msoTextureWovenMat	4

## **MsoPresetThreeDFormat**

<b>Constant</b> V	alue
msoPresetThreeDFormatMixed -2	2
msoThreeD1 1	
msoThreeD10 10	)
msoThreeD11 11	L
msoThreeD12 12	2
msoThreeD13 13	3
msoThreeD14 14	4
msoThreeD15 15	5
msoThreeD16 16	5
msoThreeD17 17	7
msoThreeD18 18	3
msoThreeD19 19	Э
msoThreeD2 2	
msoThreeD20 20	)
msoThreeD3 3	
msoThreeD4 4	
msoThreeD5 5	
msoThreeD6 6	
msoThreeD7 7	
msoThreeD8 8	
msoThreeD9 9	

## **MsoRelativeNodePosition**

Constant	Value
msoAfterLastSibling	4
msoAfterNode	2
msoBeforeFirstSibling	3

#### msoBeforeNode 1

#### **MsoScaleFrom**

Constant	Value
msoScaleFromBottomRight	2
msoScaleFromMiddle	1
msoScaleFromTopLeft	0

#### <u>MsoScreenSize</u>

### Constant Value

- msoScreenSize1024x768 4
- msoScreenSize1152x882 5
- msoScreenSize1152x900 6
- msoScreenSize1280x1024 7
- msoScreenSize1600x1200 8
- msoScreenSize1800x1440 9
- msoScreenSize1920x1200 10
- msoScreenSize544x376 0
- msoScreenSize640x480 1
- msoScreenSize720x512 2
- msoScreenSize800x600 3

### **MsoScriptLanguage**

Constant	Value
msoScriptLanguageASP	3
msoScriptLanguageJava	1
msoScriptLanguageOther	4
msoScriptLanguageVisualBasic	2

**MsoScriptLocation** 

ConstantValuemsoScriptLocationInBody2msoScriptLocationInHead1

## **MsoSearchIn**

Constant	Value
msoSearchInCustom	3
msoSearchInMyComputer	0
msoSearchInMyNetworkPlaces	2
msoSearchInOutlook	1

## **MsoSegmentType**

Constant	Value
msoSegmentCurve	1
msoSegmentLine	0

## <u>MsoShadowType</u>

Constant	Value
msoShadow1	1
msoShadow10	10
msoShadow11	11
msoShadow12	12
msoShadow13	13
msoShadow14	14
msoShadow15	15
msoShadow16	16
msoShadow17	17
msoShadow18	18
msoShadow19	19
msoShadow2	2
msoShadow20	20

msoShadow3	3
msoShadow4	4
msoShadow5	5
msoShadow6	6
msoShadow7	7
msoShadow8	8
msoShadow9	9
msoShadowMixee	d -2

## **MsoShapeType**

Constant	Value
msoAutoShape	1
msoCallout	2
msoCanvas	20
msoChart	3
msoComment	4
msoDiagram	21
msoEmbeddedOLEObject	7
msoFormControl	8
msoFreeform	5
msoGroup	6
msoInk	22
msoInkComment	23
msoLine	9
msoLinkedOLEObject	10
msoLinkedPicture	11
msoMedia	16
msoOLEControlObject	12
msoPicture	13
msoPlaceholder	14
msoScriptAnchor	18
msoShapeTypeMixed	-2

msoTable	19
msoTextBox	17
msoTextEffect	15

### **MsoSharedWorkspaceTaskPriority**

### Constant Value

 $mso Shared Work space Task Priority High \qquad 1$ 

msoSharedWorkspaceTaskPriorityLow 3

msoSharedWorkspaceTaskPriorityNormal 2

### MsoSharedWorkspaceTaskStatus

#### Constant

Value

msoSharedWorkspaceTaskStatusCompleted3msoSharedWorkspaceTaskStatusDeferred4msoSharedWorkspaceTaskStatusInProgress2msoSharedWorkspaceTaskStatusNotStarted1msoSharedWorkspaceTaskStatusWaiting5

## **MsoSortBy**

Constant	Value
msoSortByFileName	1
msoSortByFileType	3
msoSortByLastModified	4
msoSortByNone	5
msoSortBySize	2

## **MsoSortOrder**

ConstantValuemsoSortOrderAscending1msoSortOrderDescending2

## <u>MsoSyncAvailableType</u>

#### Constant Value

msoSyncAvailableAnywhere 2

msoSyncAvailableNone 0

msoSyncAvailableOffline 1

### **MsoSyncCompareType**

Constant Value

msoSyncCompareAndMerge 0 msoSyncCompareSideBySide 1

### MsoSyncConflictResolutionType

ConstantValuemsoSyncConflictClientWins0msoSyncConflictMerge2msoSyncConflictServerWins1

## **MsoSyncErrorType**

Constant	Value
msoSyncErrorCouldNotCompare	13
msoSyncErrorCouldNotConnect	2
msoSyncErrorCouldNotOpen	11
msoSyncErrorCouldNotResolve	14
msoSyncErrorCouldNotUpdate	12
msoSyncErrorFileInUse	6
msoSyncErrorFileNotFound	4
msoSyncErrorFileTooLarge	5
msoSyncErrorNone	0
msoSyncErrorNoNetwork	15
msoSyncErrorOutOfSpace	3

msoSyncErrorUnauthorizedUser 1 msoSyncErrorUnknown 16

 $msoSyncErrorUnknownDownload\ 10$ 

- msoSyncErrorUnknownUpload 9
- msoSyncErrorVirusDownload 8
- msoSyncErrorVirusUpload 7

## **MsoSyncEventType**

# Constant Value

msoSyncEventDownloadFailed	2
msoSyncEventDownloadInitiated	0
msoSyncEventDownloadNoChange 6	
msoSyncEventDownloadSucceeded 1	
msoSyncEventOffline	7
msoSyncEventUploadFailed	5
msoSyncEventUploadInitiated	3
msoSyncEventUploadSucceeded	4

## **MsoSyncStatusType**

Constant	Value
msoSyncStatusConflict	4
msoSyncStatusError	6
msoSyncStatusLatest	1
msoSyncStatusLocalChanges	3
msoSyncStatusNewerAvailable	2
msoSyncStatusNoSharedWorkspace	0
msoSyncStatusSuspended	5

## **MsoSyncVersionType**

ConstantValuemsoSyncVersionLastViewed 0

### msoSyncVersionServer 1

## **MsoTargetBrowser**

#### Constant Value

msoTargetBrowserIE4 2 msoTargetBrowserIE5 3 msoTargetBrowserIE6 4 msoTargetBrowserV3 0 msoTargetBrowserV4 1

### **MsoTextEffectAlignment**

Constant	Value
msoTextEffectAlignmentCentered	2
msoTextEffectAlignmentLeft	1
msoTextEffectAlignmentLetterJustify	4
msoTextEffectAlignmentMixed	-2
msoTextEffectAlignmentRight	3
msoTextEffectAlignmentStretchJustify	6
msoTextEffectAlignmentWordJustify	5

### **MsoTextOrientation**

Constant	Value
msoTextOrientationDownward	3
msoTextOrientationHorizontal	1
msoTextOrientationHorizontalRotatedFarEast	6
msoTextOrientationMixed	-2
msoTextOrientationUpward	2
msoTextOrientationVertical	5
msoTextOrientationVerticalFarEast	4

**MsoTextureType** 

 Constant
 Value

 msoTexturePreset
 1

 msoTextureTypeMixed
 -2

 msoTextureUserDefined
 2

### **MsoTriState**

Constant	Value
msoCTrue	1
msoFalse	0
msoTriStateMixed	-2
msoTriStateToggle	-3
msoTrue	-1

#### **MsoVerticalAnchor**

Constant	Value
msoAnchorBottom	4
msoAnchorBottomBaseLine	5
msoAnchorMiddle	3
msoAnchorTop	1
msoAnchorTopBaseline	2
msoVerticalAnchorMixed	-2

## **MsoWizardActType**

Constant	Value
msoWizardActActive	1
msoWizardActInactive	0
msoWizardActResume	3
msoWizardActSuspend	2

MsoWizardMsgType

### Constant Value

- msoWizardMsgLocalStateOff 2
- msoWizardMsgLocalStateOn 1
- msoWizardMsgResuming 5
- msoWizardMsgShowHelp 3
- msoWizardMsgSuspending 4

## **MsoZOrderCmd**

Constant	Value
msoBringForward	2
msoBringInFrontOfText	4
msoBringToFront	0
msoSendBackward	3
msoSendBehindText	5
msoSendToBack	1

# **Returning an Object from a Collection**

The **Item** property returns a single object from a collection. The following example sets the cmdbar variable to a **CommandBar** object that represents the first command bar in the **CommandBars** collection.

Set cmdbar = CommandBars.Item(1)

The **Item** property is the default property for most collections, so you can write the same statement more concisely by omitting the **Item** keyword.

```
Set cmdbar = CommandBars(1)
```

For more information about a specific collection, see the Help topic for the collection or the **Item** property for the collection.