Microsoft Office Object Model

AnswerWizard
  AnswerWizardFiles

Assistant
  Balloon

BalloonCheckboxes
  BalloonCheckbox

BalloonLabels
  BalloonLabel

COMAddIns
  COMAddIn

CommandBarButton
  CommandBar
    CommandBarControls
      CommandBarControl

CommandBarComboBox
  CommandBar
    CommandBarControls
      CommandBarControl

CommandBarPopup
  CommandBar
    CommandBarControls
      CommandBarControl

CommandBars
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.

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

<table>
<thead>
<tr>
<th>New Property</th>
<th>Object(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AssignedTo</strong></td>
<td>SharedWorkspaceTask</td>
</tr>
<tr>
<td><strong>Comments</strong></td>
<td>DocumentLibraryVersion</td>
</tr>
<tr>
<td><strong>Connected</strong></td>
<td>SharedWorkspace</td>
</tr>
<tr>
<td></td>
<td>SharedWorkspaceTask</td>
</tr>
<tr>
<td><strong>CreatedBy</strong></td>
<td>SharedWorkspaceFile,</td>
</tr>
<tr>
<td></td>
<td>SharedWorkspaceFolder,</td>
</tr>
<tr>
<td></td>
<td>SharedWorkspaceLink,</td>
</tr>
<tr>
<td></td>
<td>SharedWorkspaceTask</td>
</tr>
<tr>
<td><strong>CreatedDate</strong></td>
<td>SharedWorkspaceFolder,</td>
</tr>
<tr>
<td></td>
<td>SharedWorkspaceLink,</td>
</tr>
<tr>
<td></td>
<td>SharedWorkspaceTask</td>
</tr>
<tr>
<td><strong>DocumentAuthor</strong></td>
<td>Permission</td>
</tr>
<tr>
<td><strong>DomainName</strong></td>
<td>SharedWorkspaceMember</td>
</tr>
<tr>
<td><strong>DueDate</strong></td>
<td>SharedWorkspaceTask</td>
</tr>
<tr>
<td><strong>Email</strong></td>
<td>SharedWorkspaceMember</td>
</tr>
<tr>
<td><strong>EnableTrustedBrowser</strong></td>
<td>Permission</td>
</tr>
<tr>
<td><strong>ErrorType</strong></td>
<td>Sync</td>
</tr>
<tr>
<td><strong>ExpirationDate</strong></td>
<td>UserPermission</td>
</tr>
<tr>
<td><strong>FolderName</strong></td>
<td>SharedWorkspaceFolder</td>
</tr>
<tr>
<td><strong>Folders</strong></td>
<td>SharedWorkspace</td>
</tr>
<tr>
<td><strong>IsVersioningEnabled</strong></td>
<td>DocumentLibraryVersions</td>
</tr>
<tr>
<td></td>
<td>SharedWorkspaceFiles,</td>
</tr>
<tr>
<td></td>
<td>SharedWorkspaceFolders,</td>
</tr>
<tr>
<td></td>
<td>SharedWorkspaceLinks,</td>
</tr>
<tr>
<td></td>
<td>SharedWorkspaceMembers</td>
</tr>
<tr>
<td></td>
<td>SharedWorkspaceTasks</td>
</tr>
</tbody>
</table>
**LastRefreshed**  SharedWorkspace

**LastSyncTime**  Sync

**Links**  SharedWorkspace

**Members**  SharedWorkspace

**Modified**  DocumentLibraryVersion

**ModifiedBy**  SharedWorkspaceLink, SharedWorkspaceFile, SharedWorkspaceFolder, SharedWorkspaceTask

**ModifiedDate**  SharedWorkspaceFile, SharedWorkspaceFolder, SharedWorkspaceLink, SharedWorkspaceTask

**Notes**  SharedWorkspaceLink

**Permission**  UserPermission

**PermissionFromPolicy**  Permission

**PolicyDescription**  Permission

**PolicyName**  Permission

**RequestPermissionURL**  Permission

**SolutionID**  SmartDocument

**SolutionURL**  SmartDocument

**SourceURL**  SharedWorkspace

**Status**  SharedWorkspaceTask, Sync

**StoreLicenses**  Permission

**Tasks**  SharedWorkspace

**UserId**  UserPermission

**WorkspaceLastChangedBy**  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).

<table>
<thead>
<tr>
<th>Object</th>
<th>New Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>DocumentLibraryVersion</td>
<td>Comments, Modified, ModifiedBy</td>
</tr>
<tr>
<td>DocumentLibraryVersions</td>
<td>IsVersioningEnabled</td>
</tr>
<tr>
<td>Permission</td>
<td>DocumentAuthor, EnableTrustedBrowser, PermissionFromPolicy, RequestPermissionURL, StoreLicenses, PolicyDescription, PolicyName</td>
</tr>
<tr>
<td>SharedWorkspace</td>
<td>Connected</td>
</tr>
<tr>
<td>SharedWorkspaceFile</td>
<td>CreatedBy, CreatedDate, ModifiedBy, ModifiedDate</td>
</tr>
<tr>
<td>SharedWorkspaceFiles</td>
<td>ItemCountExceeded</td>
</tr>
<tr>
<td>SharedWorkspaceFolder</td>
<td>CreatedBy, CreatedDate, FolderName, ModifiedBy, ModifiedDate</td>
</tr>
<tr>
<td>SharedWorkspace</td>
<td>Folders</td>
</tr>
<tr>
<td>SharedWorkspaceFolders</td>
<td>ItemCountExceeded</td>
</tr>
<tr>
<td>SharedWorkspace</td>
<td>LastRefreshed</td>
</tr>
<tr>
<td>SharedWorkspaceLink</td>
<td>Notes, CreatedBy, CreatedDate, ModifiedBy, ModifiedDate</td>
</tr>
<tr>
<td>SharedWorkspace</td>
<td>Links</td>
</tr>
<tr>
<td>SharedWorkspaceLinks</td>
<td>ItemCountExceeded</td>
</tr>
<tr>
<td>SharedWorkspaceMember</td>
<td>DomainName, Email</td>
</tr>
<tr>
<td>SharedWorkspace</td>
<td>Members</td>
</tr>
<tr>
<td>SharedWorkspaceMembers</td>
<td>ItemCountExceeded</td>
</tr>
<tr>
<td>SharedWorkspace</td>
<td>SourceURL</td>
</tr>
<tr>
<td>SharedWorkspaceTask</td>
<td>AssignedTo, CreatedBy, CreatedDate, DueDate, ModifiedBy, ModifiedDate</td>
</tr>
<tr>
<td>---------------------</td>
<td>------------------------------------------------------------------</td>
</tr>
<tr>
<td>SharedWorkspace</td>
<td>Tasks</td>
</tr>
<tr>
<td>SharedWorkspaceTasks</td>
<td>ItemCountExceeded</td>
</tr>
<tr>
<td>SharedWorkspaceTask</td>
<td>Status</td>
</tr>
<tr>
<td>SmartDocument</td>
<td>SolutionID, SolutionURL</td>
</tr>
<tr>
<td>Sync</td>
<td>ErrorType, LastSyncTime, Status, WorkspaceLastChangedBy</td>
</tr>
<tr>
<td>UserPermission</td>
<td>ExpirationDate, Permission, UserId</td>
</tr>
</tbody>
</table>
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).

<table>
<thead>
<tr>
<th>New Method</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>ApplyPolicy</td>
<td>Permission</td>
</tr>
<tr>
<td>CreateNew</td>
<td>SharedWorkspace</td>
</tr>
<tr>
<td>Disconnect</td>
<td>SharedWorkspace</td>
</tr>
<tr>
<td>GetUpdate</td>
<td>Sync</td>
</tr>
<tr>
<td>OpenVersion</td>
<td>Sync</td>
</tr>
<tr>
<td>PickSolution</td>
<td>SmartDocument</td>
</tr>
<tr>
<td>PutUpdate</td>
<td>Sync</td>
</tr>
<tr>
<td>Refresh</td>
<td>SharedWorkspace</td>
</tr>
<tr>
<td>RefreshPane</td>
<td>SmartDocument</td>
</tr>
<tr>
<td>RemoveAll</td>
<td>Permission</td>
</tr>
<tr>
<td>RemoveDocument</td>
<td>SharedWorkspace</td>
</tr>
<tr>
<td>ResolveConflict</td>
<td>Sync</td>
</tr>
<tr>
<td>Restore</td>
<td>DocumentLibraryVersion</td>
</tr>
<tr>
<td>Unsuspend</td>
<td>Sync</td>
</tr>
</tbody>
</table>
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).

<table>
<thead>
<tr>
<th>New Method</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>DocumentLibraryVersion</td>
<td>Restore</td>
</tr>
<tr>
<td>Permission</td>
<td>ApplyPolicy, RemoveAll</td>
</tr>
<tr>
<td>SharedWorkspace</td>
<td>CreateNew, Disconnect, Refresh, RemoveDocument</td>
</tr>
<tr>
<td>SmartDocument</td>
<td>PickSolution, RefreshPane</td>
</tr>
<tr>
<td>Sync</td>
<td>GetUpdate, OpenVersion, PutUpdate, ResolveConflict, Unsuspend</td>
</tr>
</tbody>
</table>
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).

Office 2003 macro and add-in security setting options

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

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 add-ins, 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.
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 ← BalloonLabel

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 assigned to its **Text** 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.

```vbnet
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

COMAddIns [COMAddIn]

A collection of COMAddIn objects that provide information about a COM add-in 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 Count property of the COMAddins collection returns the number of available COM add-ins, as in the following example.

MsgBox Application.COMAddIns.Count

Use the Update 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").Description
CommandBarControls Collection Object

Multiple objects

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

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

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

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

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

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

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

CommandBars is a collection of CommandBar objects that represent the command bars in the container application.
Using the CommandBars Collection

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

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

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

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

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

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

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

If two or more custom menus or submenus have the same name, `CommandBars(index)` returns the first one. To ensure that you return the correct menu or submenu, locate the `pop-up control` that displays that menu. Then apply the `CommandBar` property to the pop-up control to return the command bar that represents that menu.
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.

```vbnet
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 Information"
Set dlvVersion = Nothing
Set dlvVersions = Nothing
```
DocumentProperties Collection Object

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

FileDialogFileDialogFiltersFileDialogFilter

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.

```vba
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.

```vba
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
```

```vba
    '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 | 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.

```vbnet
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
```
End Sub
FileTypes Collection

FileSearch.FileTypes

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:

```vba
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.

```vba
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
```

```vba
    'Declare a variable to act as a generic counter
    Dim lngCount As Long

    'Use a With...End With block to reference the
    'FileSearch object
    With Application.FileSearch

        'Clear all the parameters of the previous searches.
        'This method doesn't clear the LookIn property or
        'the SearchFolders collection.
        .NewSearch

        'Setting the FileType property clears the
        'FileTypes collection and sets the first
        'item in the collection to the file type
        'defined by the FileType property.
        .FileType = msoFileTypeWebPages
```
'Add a second item to the FileTypes collection
.FileTypes.Add msoFileTypeExcelWorkbooks

'Display the number of FileTypes in the collection.
MsgBox "You are about to search for " & .FileTypes.Count & _
   
" file types."

'Set up the search to look in all subfolders on the C:\ drive
.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
    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** | **FoundFiles**

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.

```vba
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.

```vba
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

- HTMLProject
- HTMLProjectItems
- HTMLProjectItem

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

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  ODSOFilters

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
    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
        .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
End Sub
PropertyTests Collection Object

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.

```vba
```

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.

```vba
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.

```vba
With Application.FileSearch.PropertyTests(1)
  ' Display the search criteria for the first property test
End With
```
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
ScopeFolders Collection

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.

```vba
Dim sfs As ScopeFolders
Set sfs = Application.FileSearch.SearchScopes.Item(1).ScopeFolder.ScopeFolders
```

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

A collection of Script 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 Count property to determine the number of Script objects in the specified document. The following example displays the number of scripts in the active document.

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

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

```
ActiveDocument.Scripts("ScriptOne").Delete
```
SearchFolders Collection

SearchFolders Collection

FileSearch - SearchFolders
  - ScopeFolder
  - ScopeFolders

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:

```vba
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.

```vba
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
  Case Else
End Select
Next ss

'Test to see if any ScopeFolders collections were added to 'the SearchFolders collection.
If .SearchFolders.Count > 0 Then

'Set the LookIn property to the path of 'the first ScopeFolder object in the SearchFolders 'collection. This is here so that any previous 'setting of the LookIn property doesn't affect 'the search.
.LookIn = .SearchFolders.Item(1).Path

'Execute the search and test to see if any files 'were found.
If .Execute <> 0 Then

'Display the number of files found.
.MsgBox "Files found: " & .FoundFiles.Count

'Loop through the list of found files and 'display the path of each one in a message box.
For lngCount = 1 To .FoundFiles.Count
    If 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

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

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.

```vba
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**

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.

```vba
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

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.

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

Collection

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.

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

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.

```vba
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**

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:

```vba
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.

```vba
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.Expires > Now Then

        Set sig = ActiveDocument.Signatures.Add

        sig.Issuer = strIssuer
        sig.Signer = strSigner
        sig.Expires = Now + _
```
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 → WebPageFont

A collection of WebPageFont 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 **character set**.

The following character sets are supported.

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

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

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

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.

```vba
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.

```vba
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.

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

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

With Assistant.NewBalloon
  .BalloonType = msoBalloonTypeBullets
  .Icon = msoIconTip
  .Button = 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.

```vba
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 assigned to its **Text** property.

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

```vba
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").Description

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

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

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

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

Multiple objects

\[\text{CommandBar} \rightarrow \text{CommandBarControls}\]

Represents a command bar in the container application. The CommandBar object is a member of the CommandBars collection.

Using the CommandBar Object

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

```vba
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."

```vba
Set newItem = CommandBars("Tools").Controls.Add(Type:=msoControlButton)
With newItem
    .BeginGroup = True
End With
```
If two or more custom menus or submenus have the same name, `CommandBars(index)` returns the first one. To ensure that you return the correct menu or submenu, locate the pop-up control that displays that menu. Then apply the `CommandBar` property to the pop-up control to return the command bar that represents that menu.

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

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

`CommandBarButton` in `CommandBar`

Represents a button control on a command bar.
Using the CommandBarButton Object

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

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

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

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

CommandBarComboBox

Represents a combo box control on a command bar.
Using the CommandBarComboBox Object

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

CommandBarControl

CommandBar

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

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

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

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

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

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

CommandBarPopup

Multiple objects

Represents a pop-up control on a command bar.
Using the CommandBarPopup Object

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

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

```vba
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

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.

```vba
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 Information"
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

FileDialog

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.

```vba
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 and return the user's action.
  If .Show = -1 Then

    'Step through each string in the FileDialogSelectedItems collection.
    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

  Else
    End If
  End With

' 'The user pressed Cancel.
Else
End If

'Set the object variable to Nothing.
Set fd = Nothing

End Sub
FileDialogFilter Object

FileDialogFilters.LFileDialogFilter

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.

```vba
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

FileSearch  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.

```vba
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.

```vba
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.

```vba
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 ⊑ HTMLProjectItem

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

Using the HTMLProjectItem Object

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

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

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

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

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

ActiveDocument.HTMLProject.HTMLProjectItems("ItemOne")._
   .Open (msoHTMLProjectOpenTextView)
ActiveDocument.HTMLProject.HTMLProjectItems("ItemOne")._
   .SaveCopyAs("C:\NewItem.txt")
Assuming that the text file C:\NewText.txt exists, the following example uses the **LoadFromFile** property to set the text of ItemOne to the text contained in the file. The following example uses the **Text** property to display the new text in a message box.

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

<table>
<thead>
<tr>
<th>Application</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>Microsoft Access</td>
<td><strong>NewFileTaskPane</strong></td>
</tr>
<tr>
<td>Microsoft Excel</td>
<td><strong>NewWorkbook</strong></td>
</tr>
<tr>
<td>Microsoft FrontPage</td>
<td><strong>NewPageOrWeb</strong></td>
</tr>
<tr>
<td>Microsoft PowerPoint</td>
<td><strong>NewPresentation</strong></td>
</tr>
<tr>
<td>Microsoft Word</td>
<td><strong>NewDocument</strong></td>
</tr>
</tbody>
</table>

**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.

```vba
Sub AddNewDocToTaskPane()
        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.

```vba
Sub RemoveDocFromTaskPane()
        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 **Columns**(index), where *index* is the data field name or index number, to return a single **ODSOCColumn** 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.

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: " & .Item(1).Name & vbCrLf & _
            "Value: " & .Item(1).Value
    End With
End Sub
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. 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 ODSOFilters object

Each filter is a line in a query string. 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".

```vba
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"
                End If
                If .Conjunction = "Or" Then .Conjunction = "And"
            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.

```vba
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
```
End With
End Sub
OfficeDataSourceObject Object

OfficeDataSourceObject

Represents the mail merge data source in a mail merge operation.
Using the **OfficeDataSourceObject** object

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

Sub SetDataSortOrder()
  Dim appOffice As OfficeDataSourceObject
  
  Set appOffice = Application.OfficeDataSourceObject
  appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName;UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"

  appOffice.SetSortOrder SortField1:="ZipCode", SortAscending1:=False, SortField2:="LastName", SortField3:="FirstName"
End Sub

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

Sub SetQueryCriterion()
  Dim appOffice As Office.OfficeDataSourceObject
  Dim intItem As Integer

  appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName;UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"

  With appOffice.Filters
    For intItem = 1 To .Count
      With .Item(intItem)
        If .Column = "Region" Then
          .Comparison = msoFilterComparisonNotEqual
          .CompareTo = "WA"
          If .Conjunction = "Or" Then .Conjunction = "And"
        End If
      End With
    Next intItem
  End With
End Sub
End With
End Sub
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.RequestPermissionURL & vbCrLf
   If irmPermission.PermissionFromPolicy Then
      strIRMInfo = strIRMInfo & " Permissions applied from policy:"
      " Policy name: " & irmPermission.PolicyName & vbCrLf & _
      " Policy description: " & irmPermission.PolicyDescription
   Else
      strIRMInfo = strIRMInfo & " Default permissions applied."
      " Default policy name: " &irmPermission.PolicyName
      " Default policy description: " & irmPermission.PolicyDescription
   End If
Else
   strIRMInfo = "Permissions are NOT restricted on this document"
End If
MsgBox strIRMInfo, vbInformation + vbOKOnly, "IRM Information"
Set irmPermission = Nothing
PropertyTest Object

FileSearch  PropertyTests
    PropertyTest

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.

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

ScopeFolder Object

Multiple objects \-m ScopeFolder
\-m ScopeFolders

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:

```vbnet
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:

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

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:

```vbnet
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 **SearchFolders** 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 **SearchFolders** 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.Path
                    Next sf
                Case Else
                    End Select
            Next ss
        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.

```vba
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).

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

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

```vba
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 **ScriptText** 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

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:

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

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

See the `ScopeFolder` 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.

```vba
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 scope."
                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
```
Next ss
End With
End Sub
SharedWorkspace Object

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 SharedWorkspaceFiles collection, accessed through the Files property of the SharedWorkspace object, to manage documents and files saved in a shared workspace.

Use the SharedWorkspaceFolders 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.

```vbnet
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
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 Sync 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.
The **SharedWorkspaceFile** object represents a file saved in a shared document workspace. Member of the [SharedWorkspaceFiles](#) 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)." & vbCrLf
For Each swsFile In ActiveWorkbook.SharedWorkspace.Files
    strFileInfo = strFileInfo & FilenameFromURL(swsFile.URL) & vbCrLf	& " - 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

The **SharedWorkspaceFolder** object represents a folder in a shared document workspace. Member of the **SharedWorkspaceFolders** 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.

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

The **SharedWorkspaceLink** object represents a URL link saved in a shared document workspace. Member of the **SharedWorkspaceLinks** 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.

```vbnet
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**

The **SharedWorkspaceMember** object represents a user who has rights in a shared document workspace. Member of the **SharedWorkspaceMembers** 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 Roles 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.

```vbnet
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
```
The **SharedWorkspaceTask** object represents a task in a shared document workspace. Member of the **SharedWorkspaceTasks** 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.

```vba
Dim swsTask As Office.SharedWorkspaceTask
Dim strTaskInfo As String
strTaskInfo = "The shared workspace contains " & _
ActiveWorkbook.SharedWorkspace.Tasks.Count & " Task(s)." & vbCrLf
For Each swsTask In ActiveWorkbook.SharedWorkspace.Tasks
    strTaskInfo = strTaskInfo & swsTask.Title & 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

**SignatureSet** | **Signature**

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.

```vba
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 to disk
    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 collection
    Else
        sig.Delete
        MsgBox "Not signed"
        AddSignature = False
    Error_Handler:
    End Function
```
End If

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

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

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 Status Property 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 re-open 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

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.

```vba
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

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 character set.

The following character sets are supported.

- msoCharacterSetArabic
- msoCharacterSetCyrillic
- msoCharacterSetEnglishWesternEuropeanOtherLatinScript
- msoCharacterSetGreek
- msoCharacterSetHebrew
- msoCharacterSetJapanese
- 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
```
Show All
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 [StartWizard](#) 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 [MsoWizardActType](#). 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.

```vbnet
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 `command bar` 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.

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoBarLeft, msoBarTop,</td>
<td>Indicates the left, top, right, and bottom coordinates of the new command bar</td>
</tr>
<tr>
<td>msoBarRight, msoBarBottom</td>
<td>Indicates that the new command bar won't be docked</td>
</tr>
<tr>
<td>msoBarFloating</td>
<td>Indicates that the new command bar will be</td>
</tr>
</tbody>
</table>
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.

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.

```plaintext
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 MsoFileType. 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
Add method as it applies to the PropertyTests object.

Adds a PropertyTest object to the PropertyTests collection.

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

equation.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 MsoCondition. The condition of the search criteria.

MsoCondition can be one of these MsoCondition constants.

msoConditionAnyNumberBetween
msoConditionAnytime
msoConditionAnytimeBetween
msoConditionAtLeast
msoConditionAtMost
msoConditionBeginsWith
msoConditionDoesNotEqual
msoConditionEndsWith
msoConditionEquals
msoConditionEqualsCompleted
msoConditionEqualsDeferred
msoConditionEqualsHigh
msoConditionEqualsInProgress
msoConditionEqualsLow
msoConditionEqualsNormal
msoConditionEqualsNotStarted
msoConditionEqualsWaitingForSomeoneElse
msoConditionFileTypeAllFiles
msoConditionFileTypeBinders
msoConditionFileTypeCalendarItem
msoConditionFileTypeContactItem
msoConditionFileTypeDatabases
msoConditionFileTypeDataConnectionFiles
msoConditionFileTypeDesignerFiles
msoConditionFileTypeDocumentImagingFiles
msoConditionFileTypeExcelWorkbooks
msoConditionFileTypeJournalItem
msoConditionFileTypeMailItem
msoConditionFileTypeNoteItem
msoConditionFileTypeOfficeFiles
msoConditionFileTypeOutlookItems
msoConditionFileTypePhotoDrawFiles
msoConditionFileTypePowerPointPresentations
msoConditionFileTypeProjectFiles
msoConditionFileTypePublisherFiles
msoConditionFileTypeTaskItem
msoConditionFileTypeTemplates
The value of the search criterion.

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

Specifies the way two search criteria are combined.

MsoConnector can be one of these MsoConnector constants.

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.

```plaintext
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 **MsoFilterComparison**. 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 **MsoFilterConjunction**. 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 **msoPermission**. 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 MsoScriptLocation. 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 MsoScriptLanguage. 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.

```vba
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.

```vba
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.

```vba
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.

**OverwriteIfExists**  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\(\text{Add(FolderName, ParentFolder)}\)

**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\(\text{Add(URL, Description, Notes)}\)

**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 **msoSharedWorkspaceTaskStatus**. The status of the new task. Default is **msoSharedWorkspaceTaskStatusNotStarted**.

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

**Priority**  Optional **msoSharedWorkspaceTaskPriority**. 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.

```vba
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.

```vba
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.

```vba
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.

```vba
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("MyNewFolder")
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
    "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.

```vba
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 combo box control. 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 ScopeFolder object to the SearchFolders collection.

expression.AddToSearchFolders

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

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

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

\texttt{Application.FileSearch.SearchScopes(1).ScopeFolder.AddToSearchFolder}
ApplyFilter Method

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

`expression.ApplyFilter`

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

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

Sub OfficeFilters()
    Dim appOffice As OfficeDataSourceObject
    Dim appFilters As ODSOFilters

    Set appOffice = Application.OfficeDataSourceObject
    appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName;UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"

    Set appFilters = appOffice.Filters

    MsgBox appOffice.RowCount

    appFilters.Add Column:="Region", Comparison:=msoFilterComparisonEqual, Conjunction:=msoFilterConjunctionAnd, bstrCompareTo:="WA"
    appOffice.ApplyFilter

    MsgBox appOffice.RowCount

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.

```vba
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 document.
    Case False
        With irmPermission
            .Enabled = True
            .ApplyPolicy ("\\server\share\permissionpolicy.xml")
        End With
        strIRMInfo = "Permissions are now restricted on this document and the permission policy has been applied."
End Select
MsgBox strIRMInfo, vbInformation + vbOKOnly, "IRM Information"
Set irmPermission = Nothing
```
Show All
Clear Method

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

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

`expression.Clear`

`expression`    Required. An expression that returns a `CommandBarComboBox` object.
Example

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.

```vba
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
```
ClearFileList Method

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

expression.ClearFileList

expression An expression that returns an AnswerWizard object.
Remarks

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

To restore the default AnswerWizard file set for the host application, use the ResetFileList method.
Example

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.
Example

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 lbtn
        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.
Example

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

        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 command bar control to an existing command bar.

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.
Example

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 button control to the Clipboard.

expression.CopyFace

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

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

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.

```vba
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".
Remarks

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.
Example

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.

```vba
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
```
Show All
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.
Remarks

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 objects.**

Deletes the specified object from its collection.

```plaintext
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.

```plaintext
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.

```plaintext
expression.Delete
```
**expression**  Required. An expression that returns one of the above objects.
**Remarks**

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.
Example

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

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

```vba
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".

```vba
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.
Remarks

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.
Example

The following example determines whether the active document is connected to a shared workspace, then offers the user the option of disconnecting it.

```vbnet
Dim r As Long
If ActiveWorkbook.SharedWorkspace.Connected Then
    r = MsgBox("Are you sure you want to disconnect this document?", 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 workspace.", 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 MsoAlertButtonType. Determines which buttons are displayed on the alert.

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

alc Required MsoAlertIconType. Determines the icon that is displayed on the alert.

MsoAlertIconType can be one of these MsoAlertIconType constants.
msoAlertIconCritical
msoAlertIconInfo
**msoAlertIconNoIcon**
**msoAlertIconQuery**
**msoAlertIconWarning**

*alq*  Required **MsoAlertDefaultType**. 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 **MsoAlertCancelType**. Always set this to **msoAlertCancelDefault**. Any other setting may return an error.

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

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

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

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

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.", _
                msoAlertButtonYesAllINoCancel, _
                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 the return value for YesToAll
            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 `StartWizard` 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`.  ```
Example

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 **MsoSortBy**. 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 **MsoSortOrder**. 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 command bar control. For custom controls, use the OnAction 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.
Example

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.

```vba
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.

```vba
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 MsoControlType. 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**.
Example

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 MsoControlType. 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`. 
Example

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.

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

Compared 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.
**Example**

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.

```vba
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 Information"
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.
Example

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.

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.

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.
Example

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 **command bar control** to an existing **command bar**.

```vba
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.

```vba
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.
Example

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.

```vba
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.

```vba
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.
Example

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

```vba
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 **MsoHTMLProjectOpen**. The view in which the specified project or project item is opened.

MsoHTMLProjectOpen can be one of these MsoHTMLProjectOpen constants.

- msoHTMLProjectOpenSourceView
- msoHTMLProjectOpenTextView
Remarks

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.
Example

As it applies to the DocumentLibraryVersion object

The following example opens the previous saved version of the active document in read-only mode.

```vba
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.

```vba
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.

```vba
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 MsoSyncVersionType.

MsoSyncVersionType can be one of the following msoSyncVersionType constants.

msoSyncVersionLastViewed (0)
msoSyncVersionServer (1)
Remarks

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.
Example

The following example prompts the user to open the server copy of the shared document alongside the currently open local version.

```vba
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
```
Show All
**PasteFace Method**

Pastes the contents of the Clipboard onto a command bar **button control**.

`expression.PasteFace`  

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

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:=2)`
Set `myControl.CopyFace`
Set `myControl = CommandBars.FindControl(Type:=msoControlButton, Id:=23)`
Set `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 SmartDocument 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.
Remarks

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.
Example

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.
Remarks

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**.
Example

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.

```vba
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 Information"
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.
Remarks

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.
**Example**

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.

```vba
On Error GoTo err_NeverRefreshed
If DateDiff("s", ActiveWorkbook.SharedWorkspace.LastRefreshed, Now) > 180 Then
  ActiveWorkbook.SharedWorkspace.Refresh
End If
Exit Sub

err_NeverRefreshed:
  ActiveWorkbook.SharedWorkspace.Refresh
```
RefreshDocument Method

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

\textit{expression}.\texttt{RefreshDocument(Refresh)}

\textit{expression} An expression that returns an \texttt{HTMLProject} object.

\textbf{Refresh} Required \texttt{Boolean}. \texttt{True} if all changes are to be saved; \texttt{False} if all changes are to be ignored.
Remarks

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.
Example

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 [SmartDocument](#) object.
Remarks

The **RefreshPane** method raises an error if the active document does not have an XML expansion pack attached.
Example

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.

```vba
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.
Remarks

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.
Example

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.
Example

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

Next i
MsgBox "Folder Names on C:..." & vbCrLf & strResults

End With

End Sub
Show All
ReleaseFocus Method

Releases the user interface focus from all command bars.

expression.ReleaseFocus

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

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.

```vba
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.

\[ \text{expression}.\text{Remove}(\text{Index}) \]

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.

\[ \text{expression}.\text{Remove}(\text{FileName, Section, DisplayName, Action}) \]

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.

\[ \text{expression}.\text{Remove}() \]
expression  Required. An expression that returns a UserPermission object.
Remarks

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.
Example

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.

```vba
Sub RemoveDocFromTaskPane()
    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.

```vba
Dim irmPermission As Office.Permission
Dim irmUserPerm As Office.UserPermission
Set irmPermission = ActiveWorkbook.Permission
Set irmUserPerm = irmPermission.Item(2)
irmUserPerm.Remove
MsgBox "Permission removed.", _
    vbInformation + vbOKOnly, "IRM Information"
Set irmUserPerm = Nothing
Set irmPermission = Nothing
```
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.
Remarks

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).
Example

The following example uses the **RemoveAll** method to remove all user permissions and to disable restrictions on the active document.

```vba
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.
Remarks

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.
Example

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.

```vbnet
Dim r As Long
If ActiveWorkbook.SharedWorkspace.Connected Then
    r = MsgBox("Are you sure you want to remove this document?", vbQuestion + vbOKCancel, "Are you sure?")
    If r = vbOK Then
        ActiveWorkbook.SharedWorkspace.RemoveDocument
        MsgBox "The document has been removed.", _
            vbInformation + vbOKOnly, "Removed"
    Else
        MsgBox "Removal cancelled.", _
            vbInformation + vbOKOnly, "Still In Workspace"
    End If
Else
    MsgBox "The active document is not connected to a shared workspace.", _
        vbInformation + vbOKOnly, "Not Connected"
End If
```
RemoveItem Method

Removes an item from a command bar combo box control.

**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.
Example

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.

End With
    Set myControl = myBar.Controls(1)
With myControl
    If .ListCount > 3 Then
        .RemoveItem 2
        .Style = msoComboBoxNormal
        .Text = "New Default"
        Set ctrl = .Parent
    End If
End With
Reset Method

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

expression.Reset

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

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

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

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

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

`expression.ResetFileList`

`expression` An expression that returns an AnswerWizard object.
Remarks

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

**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.
Example

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 available through this AnswerWizard: " & _
    Chr(13) & strFileList
**ResetTips Method**

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

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

`expression.ResetTips`

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

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

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

With Application.Assistant
    .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 `MsoSyncConflictResolutionType`.

MsoSyncConflictResolutionType can be one of these soSyncConflictResolutionType constants.

- `msoSyncConflictClientWins` (0)
- `msoSyncConflictServerWins` (1)
- `msoSyncConflictMerge` (2)
Remarks

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.
Example

The following example attempts to resolve a conflict by merging changes between the local and the server copies of the active document.

```vba
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 Information"
End If
Set objSync = Nothing
```
### Restore Method

Restores a previous saved version of a shared document from the `DocumentLibraryVersions` collection.

```plaintext
expression.Restore()
```

`expression`  Required. An expression that returns a `DocumentLibraryVersion` object.
Remarks

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.
Example

The following example restores the previous version of the active document.

```vba
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.
Remarks

Use the **Save** method to upload changes to the server after changing the properties of a shared workspace link or task.
Example

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.

```vba
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.

\textit{expression}.\textit{SaveCopyAs(Filename)}

\textit{expression}   An \texttt{HTMLProjectItem} object.

\textit{Filename}   Required \texttt{String}. The fully qualified path of the file to which you want to save the HTML project item.
Example

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.

\[
expression.SetAvoidRectangle(\text{Left, Top, Right, Bottom})
\]

\textit{expression} Required. An expression that returns an \texttt{Assistant} object.

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

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

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

```vbnet
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 command bar control. 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.
Example

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.

```vba
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.
Example

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.Show`

`expression` Required. An expression that returns a `FileDialog` object.
Example

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.

```vba
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.

```vba
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
```
Case msoBalloonButtonNo  ' User selected No button.
    returnValue = MsgBox("Printer orientation not " & _
"reset.", vbOKOnly, "Printer Message")
End Select

As it applies to the FileDialog object.

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

Sub Main()

    'Declare a variable as a FileDialog object.
    Dim fd As FileDialog

    'Create a FileDialog object as a File Picker dialog box.
    Set fd = Application.FileDialog(msoFileDialogFilePicker)

    'Declare a variable to contain the path
    'of each selected item. Even though the path is a String,
    'the variable must be a Variant because For Each...Next
    'routines only work with Variants and Objects.
    Dim vrtSelectedItem As Variant

    'Use a With...End With block to reference the FileDialog object.
    With fd

        'Use the Show method to display the File Picker dialog box a
        'The user pressed the action button.
        If .Show = -1 Then

            'Step through each string in the FileDialogSelectedItems
            For Each vrtSelectedItem In .SelectedItems

                'vrtSelectedItem is a string that contains the path
                'You can use any file I/O functions that you want to
                'This example simply displays the path in a message
                MsgBox "The path is: " & vrtSelectedItem

            Next vrtSelectedItem

            'The user pressed Cancel.
        Else
            End If
    End With

    'Set the object variable to nothing.
    Set fd = Nothing
End Sub
Show All
ShowPopup Method

Displays a command bar as a shortcut menu at the specified coordinates or at the current pointer coordinates.

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

`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.
Example

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

```vba
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 **Visible** 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.
**Example**

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`.

\[ lHelpForWiz = \text{Assistant.StartWizard(On:=True, _} \\
\text{Callback:="myCallback", PrivateX:=23)} \]
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.
Example

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.
Example

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.
Example

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.

```vba
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.

```vba
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 CommandBar object that represents the active menu bar in the container application. Read-only.
Example

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, Temporary:=True)
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**.
Example

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.
Example

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.
Example

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 return the user's action.
        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 MsoAnimationType.

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
msoAnimationWorkingAtSomething
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`
- `msoAnimationGetTechy`
- `msoAnimationListensToComputer`
- `msoAnimationSearching`
- `msoAnimationThinking`
- `msoAnimationWorkingAtSomething`
- `msoAnimationWritingNotingSomething`
Example

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.
Example

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.
Example

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.

```vba
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.
Example

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

With Assistant
  .On = True
  .Visible = True
  .AssistWithHelp = True
  .AssistWithAlerts = 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.
Example

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.
Example

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.
Example

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

    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.

**msoBalloonErrorButtonClickless** 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).
Example

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.

```vba
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
Example

This example creates an instruction balloon that explains how to select a printer. The balloon is modeless, so the user can follow the instructions in the balloon and keep the balloon visible as he or she works.

```vba
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 = msoButtonTypeOK
    .Show
End With
```
BeginGroup Property

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

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 command bar or command bar control is a built-in command bar or control of the container application. **False** if it's a custom command bar or control, or if it's a built-in control whose OnAction property has been set. Read-only **Boolean**.
Example
This example deletes all custom command bars that aren't visible.
foundFlag = False
deletedBars = 0
For Each bar In CommandBars
If (bar.BuiltIn = False) And (bar.Visible = False) Then
bar.Delete
foundFlag = True
deletedBars = deletedBars + 1
End If
Next
If Not foundFlag Then
MsgBox "No command bars have been deleted."
Else
MsgBox deletedBars & " custom command bar(s) deleted."
End If


BuiltInFace Property

True if the face of a command bar button control 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.
Example

This example determines whether the face of the first control on the command bar named "Custom" is its built-in button face. If it is, the example copies the button face to the Clipboard.

```
Set myControl = CommandBars("My Custom Bar").Controls(1)
With myControl
    If .BuiltInFace = True Then .CopyFace
End With
```
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
- msoButtonSetNextClose
- msoButtonSetNone
- msoButtonSetOK
- msoButtonSetOkCancel
- msoButtonSetRetryCancel
- msoButtonSetSearchClose
- msoButtonSetTipsOptionsClose
- msoButtonSetYesAllNoCancel
- msoButtonSetYesNo
- msoButtonSetYesNoCancel
Example

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.
Example

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

```vba
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").
Example

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 lbtn
    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
Show All
Caption Property

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

Note A control's caption is also displayed as its default ScreenTip.
Example

This example adds a command bar control with a spelling checker button face to a custom command bar, and then it sets the caption to "Spelling checker."

```
Set myBar = CommandBars.Add(Name:="Custom", 
Position:=msoBarTop, Temporary:=True)
myBar.Visible = True
Set myControl = myBar.Controls .Add(Type:=msoControlButton, Id:=2)
With myControl
    .DescriptionText = "Starts the spelling checker"
    .Caption = "Spelling checker"
End With
```
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 [Returning an Object from a Collection](#).
Example

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.

```vba
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.**
Example

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.
Example

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.
Example

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
Show All
CommandBar Property

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

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

```vba
Set fourthLevel = CommandBars("Drawing") _
    .Controls(1).CommandBar.Controls(4)
```
CommandBars Property

Returns a CommandBars collection.

expression.CommandBars

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

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.
**Example**

The following example lists comments and other properties for each version of a shared document.

```vba
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
```
**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.
**Example**

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".

```vba
Sub SetQueryCriterion()
    Dim appOffice As Office.OfficeDataSourceObject
    Dim intItem As Integer

    Set appOffice = Application.Office.DataSourceObject
    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.
- **msoFilterComparisonContains**
- **msoFilterComparisonEqual**
- **msoFilterComparisonGreaterThan**
- **msoFilterComparisonGreaterThanOrEqual**
- **msoFilterComparisonIsBlank**
- **msoFilterComparisonIsNotBlank**
- **msoFilterComparisonLessThan**
- **msoFilterComparisonLessThanOrEqual**
- **msoFilterComparisonNotContains**
- **msoFilterComparisonNotEqual**

**expression**.**Comparison**

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

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
Example

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.
Example

The following example changes an existing filter to remove from the mail merge all records that do not have a Region field equal to "WA".

Sub SetQueryCriterion()
    Dim appOffice As Office.OfficeDataSourceObject
    Dim intItem As Integer

    Set appOffice = Application.OfficeDataSourceObject
    appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName;UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"

    With appOffice.Filters
        For intItem = 1 To .Count
            With .Item(intItem)
                If .Column = "Region" Then
                    .Comparison = msoFilterComparisonNotEqual
                    .CompareTo = "WA"
                    If .Conjunction = "Or" Then .Conjunction = "And"
                End If
            End With
        Next intItem
    End With
End Sub
Connect Property

Returns or sets the state of the connection for the specified COMAddIn object. Read/write Boolean.
Remarks

The Connect property returns True if the add-in is active; it returns False if the add-in is inactive. An active add-in is registered and connected; an inactive add-in is registered but not currently connected.
Example

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.
Remarks

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.
Example

The following example checks the **Connected** property to determine whether the active document is already saved in a shared workspace.

```vba
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`
Remarks

A connector specifies whether two similar search criteria will be combined to form one property test (as with msoConnectorAnd) or treated independently (as with msoConnectorOr).
Example

This example displays a message that describes how the search criteria will be evaluated in a file search.

```vba
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.
Example

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 source."
    End If
End Sub
Context Property

Returns or sets a string that determines where a command bar will be saved. The string is defined and interpreted by the application. Read/write String.
Remarks

You can set the **Context** property only for custom command bars. This property will fail if the application doesn't recognize the context string, or if the application doesn't support changing context strings programmatically.
Example

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 CommandBarControls object that represents all the controls on a command bar or pop-up control. Read-only.

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

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; read-only Long for all other objects in the Applies To list.

expression.Count

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

For the **CommandBars** collection, the count includes only menu bars, toolbars, and shortcut menus.

For the **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.
Example

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

```vba
MsgBox "There are " & CommandBars.Count & _
    " bars in the CommandBars collection."
```

This example uses the **Count** property to display the number of check boxes in the Office Assistant balloon.

```vba
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.

```vba
MsgBox ("There are " & _
    " 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.
Remarks

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.
Example

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 sswsOwner As Office.SharedWorkspaceMember
Dim strMemberFiles As String
Set sswsOwner = ActiveWorkbook.SharedWorkspace.Members(1)
For Each sswsFile In ActiveWorkbook.SharedWorkspace.Files
    If sswsFile.CreatedBy <> sswsOwner.Name Then
        strMemberFiles = strMemberFiles & sswsFile.URL & vbCrLf
    End If
Next
MsgBox "These files were created by other users:" & vbCrLf & strMemberFiles, vbInformation + vbOKOnly, "Files Created by Other Users"
Set sswsOwner = Nothing
Set sswsFile = Nothing
CreatedDate Property

Returns the date and time when the shared workspace object was created. Read-only Variant.

expression.CreatedDate

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

The `SharedWorkspaceFolder` and `SharedWorkspaceMember` objects do not have a `CreatedDate` property.
Example

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) & " 12:00:00 am")

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 \texttt{wdCreatorCode} in Word, or \texttt{xlCreatorCode} in Excel. Read-only \texttt{Long}.

\textit{expression}.\texttt{Creator}

\textit{expression} Required. An expression that returns one of the above objects.
Remarks

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 application 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.
Example

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.
Example

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.
**Example**

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

The following example displays the description text of the Microsoft Accessibility COM add-in for drawing.

```vba
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.

```vba
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 command bar control. 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.
Example

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.
Example

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: .Execute
                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.
Example

The following example toggles the `DisableAskAQuestionDropdown` property.

```vba
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.
Example

The following example toggles the \texttt{DisableCustomize} property.

\begin{verbatim}
Sub ToggleCustomize()
    With Application.CommandBars
        If \texttt{.DisableCustomize} = True Then
            \texttt{.DisableCustomize} = False
        Else
            \texttt{.DisableCustomize} = True
        End If
    End With
End Sub
\end{verbatim}
DisplayFonts Property

True if the font names in the Font box are displayed in their actual fonts. Read/write Boolean.
**Example**

This example sets three options for all command bars in Microsoft Office, including custom command bars and the controls on them.

```vba
With CommandBars
    .LargeButtons = True
    .DisplayFonts = True
    .AdaptiveMenus = True
End With
```
DisplayKeysInTooltips Property

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

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

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 command bar controls. 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.
Example

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.
Example

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." & vbCrLf
    strIRMInfo = strIRMInfo & " View in trusted browser: " & _
    irmPermission.EnableTrustedBrowser & vbCrLf & _
    " Document author: " & irmPermission.DocumentAuthor & vbCrLf & _
    " Users with rights: " & irmPermission.Count & vbCrLf & _
    " Cache licenses locally: " & irmPermission.StoreLicense & vbCrLf & _
    " Request permission URL: " & irmPermission.RequestPermissionURL & vbCrLf
    If irmPermission.PermissionFromPolicy Then
        strIRMInfo = strIRMInfo & " Permissions applied from policy:"
        strIRMInfo = strIRMInfo & vbCrLf & _
        " Policy name: " & irmPermission.PolicyName & vbCrLf & _
        " Policy description: " & irmPermission.PolicyDescription
    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.
Example

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.

```vba
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 combo box control. The combo box control must be a custom control and it must be a drop-down list box or a combo box. Read/write Long.

**Note** An error occurs if you attempt to set this property for a combo box control that's an edit box or a built-in combo box control.
Remarks

If this property is set to 0 (zero), the number of lines in the control will be based on the number of items in the list.
Example

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.

```vba
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 combo box control. Read/write Long.

Note  An error occurs if you attempt to set this property for a built-in control.
Remarks

If this property is set to -1, the width of the list is based on the length of the longest item in the combo box list. If this property is set to 0, the width of the list is based on the width of the control.
**Example**

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.
Example

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.

```vbnet
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.
Example

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.

```vba
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 command bar or command bar control is enabled. Read/write Boolean.
Remarks

For command bars, setting this property to **True** causes the name of the command bar to appear in the list of available command bars.

For built-in controls, if you set the **Enabled** property to **True**, the application determines its state, but setting it to **False** will force it to be disabled.

[Enabled property as it applies to the Permission object](https://example.com/)

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.
Remarks

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.
Example

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.

```vba
Set myBar = CommandBars(" VB Custom Bar")
    myBar.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.

```vba
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.
Remarks

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.
Example

The following example reports whether the active document has restricted permissions and, if so, whether the EnableTrustedBrowser property is also enabled.

```vba
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 document." & vbCrLf & "Viewing in trusted browser is also enabled.", vbInformation + vbOKOnly, "IRM Information"
            Case False
                MsgBox "Permissions are restricted on this document. However, viewing in trusted browser is disabled.", vbInformation + vbOKOnly, "IRM Information"
        End Select
    Case False
        MsgBox "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.
Remarks

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.
Example

The following example checks the **Status** property, then uses the **ErrorType** property to display the type of sync error that has occurred.

```vba
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.ExpirationDate

expression Required. An expression that returns a UserPermission object.
Remarks

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.
Example

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 & vbCrLf "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.
Example

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

        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 disk
    ActiveDocument.SignaturesCommit

    Exit Function

Error_Handler:
    AddSignature = False
    MsgBox "Action cancelled."
End Function
Extended Property

Sets or returns attributes added to the `<SCRIPT>` tag, with the exception of the LANGUAGE and ID attributes. Read/write `String`. 
Remarks

Attributes are separated by spaces, the same as in HTML. You cannot pass the LANGUAGE attribute or the ID attribute by using the Extended property.

The Microsoft Office host application doesn’t provide any means of checking the syntax of passed attributes.

If you pass the LANGUAGE attribute in the Extended property, the <SCRIPT> tag receives two language settings, which causes a conflict.

If you pass an ID attribute in the Extended property and no ID has been set through either the ID parameter of the Add method or the Id property of the Script object, the ID is exported correctly.
**Example**

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.

```vbnet
If ActiveWorkbook.Worksheets(1).Scripts(1).Extended _
    <> "" Then
    MsgBox "This script contains extended attributes."
End If
```
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.
Example

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.

```vbnet
Sub Main()
    'Declare a variable as a FileDialogFilters collection.
    Dim fdfs As FileDialogFilters

    'Declare a variable as a FileDialogFilter object.
    Dimfdf As FileDialogFilter

    'Set the FileDialogFilters collection variable to
    'the FileDialogFilters collection of the SaveAs dialog box.
    Setfdfs = Application.FileDialog(msoFileDialogSaveAs).Filters

    'Iterate through the description and extensions of each
    'default filter in the SaveAs dialog box.
    For Eachfdf 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 button control. Read/write Long.
Remarks

The **FaceId** property dictates the look, not the function, of a command bar button. The **Id** property of the **CommandBarControl** object determines the function of the button.

The value of the **FaceId** property for a command bar button with a custom face is 0 (zero).
Example

This example adds a command bar button to a custom command bar. Clicking this button is equivalent to clicking the **Open** command on the **File** menu because the ID number is 23, yet the button has the same button face as the built-in **Charting** button.

```vba
Set newBar = CommandBars.Add(Name:="Custom2", _
    Position:=msoBarTop, Temporary:=True)
newBar.Visible = True
Set con = newBar.Controls.Add(Type:=msoControlButton, Id:=23)
con.FaceId = 17
```
FeatureTips Property

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

True if the Office Assistant provides information about using application features more effectively. Read/write Boolean.
Remarks

The FeatureTips property corresponds to the Using features more effectively check box on the Options tab in the Assistant dialog box.
Example

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

Assistant. FeatureTips = True
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.
Example

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 & ": " & .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 AnswerWizardFiles collection that represents the list of files available to the current AnswerWizard. Read-only.

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

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.
Example

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 available 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.
Remarks

Example

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 FileTypes collection.

expression.FileTypes

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

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
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
FilterIndex Property

Returns or sets a **Long** indicating the default file filter of a file dialog box. The default filter determines which types of files are displayed when the file dialog box is first opened. Read/write.

`expression.FilterIndex`

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

If you try to set this property to a number greater than the number of filters, the last available filter will be selected.
Example

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 .SelectedItems
            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 If
    End With
'Set the object variable to Nothing.
Set fd = Nothing

End Sub
Filters Property

Returns a **FileDialogFilters** collection.

*expression.Filters*

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

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."

```vba
Sub Main()

    'Declare a variable as a FileDialog object.
    Dim fd As FileDialog

    'Create a FileDialog object as a File Picker dialog.
    Set fd = Application.FileDialog(msoFileDialogFilePicker)

    'Declare a variable to contain the path
    'of each selected item. Even though the path is a String,
    'the variable must be a Variant because For Each...Next
    'routines only work with Variants and Objects.
    Dim vrtSelectedItem As Variant

    'Use a With...End With block to reference the FileDialog object.
    With fd

        'Add a filter that includes GIF and JPEG images and make it
        Filters.Add "Images", "*.gif; *.jpg; *.jpeg", 1

        'Use the Show method to display the File Picker dialog box a
        'If the user presses the action button...
        If .Show = -1 Then

            'Step through each string in the FileDialogSelectedItems
            For Each vrtSelectedItem In .SelectedItems

                'vrtSelectedItem is a String that contains the path
                'You can use any file I/O functions that you want to
                'This example simply displays the path in a message
                MsgBox "Selected item's path: " & vrtSelectedItem

            Next vrtSelectedItem

        'If the user presses Cancel...
        Else
            End If
    End With

    'Set the object variable to Nothing.
    Set fd = Nothing
```

End Sub
FixedWidthFont Property

Sets or returns the fixed-width font setting in the host application. Read/write String.
Remarks

When you set the `FixedWidthFont` property, the host application does not check the value for validity.
**Example**

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.

```plaintext
Application.DefaultWebOptions._
Fonts(msoCharacterSetEnglishWesternEuropeanOtherLatinScript)._.
 FixedWidthFont = "System"
Application.DefaultWebOptions._
Fonts(msoCharacterSetEnglishWesternEuropeanOtherLatinScript)._.
 FixedWidthFontSize = 12
```
FixedWidthFontSize Property

Sets or returns the fixed-width font size setting in the host application, in points. Read/write Single.
Remarks

When you set the **FixedWidthFontSize** property, the host application does not check the value for validity. If you enter an invalid value, such as a nonnumber, the host application sets the size to 0 points. You can enter half-point sizes; if you enter other fractional point sizes, they are rounded up or down to the nearest half-point.
Example

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)_ .FixedSizeFont = "System"
Application.DefaultWebOptions._ Fonts(msoCharacterSetEnglishWesternEuropeanOtherLatinScript)_ .FixedSizeFontSize = 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.
Remarks

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`. 
Example

The following example displays the number of subfolders in the shared workspace and their names.

```vba
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 & vbCrLf
    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.
Remarks

The **SharedWorkspaceFolders** collection does not include the root document library folder itself, which by default is named "Shared Documents".
Example

The following example lists the subfolders in the current shared workspace.

```vba
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.
Example

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

```vbnet
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.
Remarks

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

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.
Example

The following example displays the ProgID and GUID for COM add-in one in a message box.

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

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

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

You can specify a graphic to be displayed in the balloon heading by using the following syntax: `{type location sizing_factor}`, where type is bmp (bitmap) or wmf (Windows metafile), location is the resource ID or the path and file name, and sizing_factor specifies the width of the wmf (omitted for bmp).

The balloon heading also supports underlined text and text that has one of the 16 system palette colors applied to it. To display underlined text in a heading, use the syntax `{ul}` or `{ul 1}; use `{ul 0}` to turn underlining off. To change the color of heading text, precede the text string with the character sequence `{cf number}`, where number is one of the system color numbers listed in the following table.

<table>
<thead>
<tr>
<th>System color number</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Black</td>
</tr>
<tr>
<td>1</td>
<td>Dark red</td>
</tr>
<tr>
<td>2</td>
<td>Dark green</td>
</tr>
<tr>
<td>3</td>
<td>Dark yellow</td>
</tr>
<tr>
<td>4</td>
<td>Dark blue</td>
</tr>
<tr>
<td>5</td>
<td>Dark magenta</td>
</tr>
<tr>
<td>6</td>
<td>Dark cyan</td>
</tr>
<tr>
<td>7</td>
<td>Light gray</td>
</tr>
<tr>
<td>248</td>
<td>Medium gray</td>
</tr>
<tr>
<td>249</td>
<td>Red</td>
</tr>
<tr>
<td>250</td>
<td>Green</td>
</tr>
<tr>
<td>251</td>
<td>Yellow</td>
</tr>
<tr>
<td>252</td>
<td>Blue</td>
</tr>
<tr>
<td>253</td>
<td>Magenta</td>
</tr>
<tr>
<td>254</td>
<td>Cyan</td>
</tr>
<tr>
<td>255</td>
<td>White</td>
</tr>
</tbody>
</table>

If you specify a number other than one of the preceding system color numbers, the heading text is black.
Example

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
Show All
Height Property

Returns or sets the height of a command bar control or command bar. Read/write Long.

expression.**Height**

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

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 command bar control. Read/write Long.
Remarks

To use this property, you must also set the HelpFile property. Help topics respond to Shift+F1.
Example

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.

```vba
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 command bar control. Read/write String.
Remarks

To use this property, you must also set the HelpContextID property. Help topics respond to the user pressing SHIFT+F1.
Example

This example adds a custom command bar with a combo box that tracks stock data. The example also specifies the Help topic to be displayed for the combo box when the user presses SHIFT+F1.

Set myBar = CommandBars._
  .Add(Name:="Custom", Position:=msoBarTop, _
      Temporary:=True)
With myBar
  .Controls.Add Type:=msoControlComboBox, ID:=1
     .Visible = True
End With
With CommandBars("Custom").Controls(1)
  .AddItem "Get Stock Quote", 1
  .AddItem "View Chart", 2
  .AddItem "View Fundamentals", 3
  .AddItem "View News", 4
  .Caption = "Stock Data"
  .DescriptionText = "View Data For Stock"
  .HelpFile = "C:\corphelp\custom.hlp"
  .HelpContextID = 47
End With
HighPriorityTips Property

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

**True** if the Office Assistant displays high-priority tips. Read/write **Boolean**.
Remarks

The **HighPriorityTips** property corresponds to the **Only show high priority tips** option under **Show tips about** on the **Options** tab in the **Office Assistant** dialog box.
Example

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**.
Example

This example returns the number of items in the **HTMLProjectItems** collection in the HTML project in the active workbook.

```vbnet
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
Example

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.

```vba
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


**Example**

This example creates a balloon with an “Alert” icon that instructs the user to select a printer.

```vba
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 command bar control. 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.
Example

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

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

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

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

This example searches the command bar named "Custom2" for a control with an Id value of 23. If such a control is found and the index number of the 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

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

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

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

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

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

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 document
With .Item

'All of the mail item settings are saved with the document
'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

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

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 signers 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 to
    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
**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

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

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 signers 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 to disk
    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 collection
    Else
       sig.Delete
       MsgBox "Not signed"
       AddSignature = False
    End If

    'Commit all signatures in the SignatureSet collection to the disk
    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.
Example

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`.

<table>
<thead>
<tr>
<th>Number of uses of the command bar control</th>
<th>Number of sessions of the application</th>
</tr>
</thead>
<tbody>
<tr>
<td>0, 1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
</tr>
<tr>
<td>4, 5</td>
<td>12</td>
</tr>
<tr>
<td>6–8</td>
<td>17</td>
</tr>
<tr>
<td>9–13</td>
<td>23</td>
</tr>
<tr>
<td>14–24</td>
<td>29</td>
</tr>
<tr>
<td>25 or more</td>
<td>31</td>
</tr>
</tbody>
</table>
Example

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.

```vbnet
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`

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

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.

```vba
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 to disk
    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 collection
    Else
       sig.Delete
       MsgBox "Not signed"
       AddSignature = False
    End If

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

Exit Function
```

Error_Handler:
```vba
Resume Next
```
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`

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

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 signers 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 committing the Signature object to disk.
    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 collection.
    Else
        sig.Delete
        MsgBox "Not signed"
        AddSignature = False
    End If

    'Commit all signatures in the SignatureSet collection to the disk.
    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.
Example

The following example displays the number of saved versions of the active document, if versioning is enabled.

```vba
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.


expression.Item(Index)

expression Required. An expression that returns a DocumentLibraryVersions object.


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 MsoCharacterSet. The specified character set.

MsoCharacterSet can be one of these MsoCharacterSet constants.
msoCharacterSetArabic
msoCharacterSetCyrillic
msoCharacterSetEnglishWesternEuropeanOtherLatinScript
msoCharacterSetGreek
msoCharacterSetHebrew
msoCharacterSetJapanese
msoCharacterSetKorean
msoCharacterSetMultilingualUnicode
msoCharacterSetSimplifiedChinese
msoCharacterSetThai
msoCharacterSetTraditionalChinese
msoCharacterSetVietnamese
Example

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 available 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.

```vba
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.

```vba
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.
Example

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.

```vba
ActiveWorkbook.SharedWorkspace.Refresh
If ActiveWorkbook.SharedWorkspace.Links.Count = 0 Then
    If ActiveWorkbook.SharedWorkspace.Links.ItemCountExceeded Then
        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
Else
    MsgBox "More than 99 links in shared workspace.", vbInformation + vbOKOnly, "Item Count Exceeded"
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.
Example

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 Returning an Object from a Collection.
Example

This example creates a balloon containing three choices. The variable x is set to the return value of the Show method, which will be 1, 2 or 3, 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
\texttt{MsoScriptLanguage}.

\texttt{MsoScriptLanguage} can be one of these \texttt{MsoScriptLanguage} constants.
\texttt{msoScriptLanguageASP}
\texttt{msoScriptLanguageJava}
\texttt{msoScriptLanguageOther}
\texttt{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.
Example

This example checks the **Language** property to ensure that the first script in worksheet one in the active workbook is written in VBScript.

```vba
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(\textit{Id})

\textit{expression} Required. An expression that returns a LanguageSettings object.

\textit{Id} Required \texttt{MsoAppLanguageID}.

<table>
<thead>
<tr>
<th>\texttt{MsoAppLanguageID} constant</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

\texttt{msoLanguageIDExeMode} 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

\texttt{msoLanguageIDHelp} The language used for online Help.
**msoLanguageIDInstall**

The language settings used by Microsoft Office to set up defaults. For example, Microsoft Word 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.

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.

**msoLanguageIDUIPrevious**
Example

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."
Show All
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 **MsoLanguageID**.

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`
Remarks

You must test all valid msoLanguageID values to enumerate the set of preferred languages.
Example

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**.
Example

This example toggles the display size of toolbar buttons on all command bars.

Set allBars = CommandBars
If allBars.LargeButtons Then
    allBars.LargeButtons = False
Else
    allBars.LargeButtons = True
End If
LastModified Property

Returns or sets a constant that represents the amount of time since the specified file was last modified and saved. The default value is `msoLastModifiedAnyTime`. Read/write `MsoLastModified`.

MsoLastModified can be one of these MsoLastModified constants.

- `msoLastModifiedAnyTime`
- `msoLastModifiedLastMonth`
- `msoLastModifiedLastWeek`
- `msoLastModifiedThisMonth`
- `msoLastModifiedThisWeek`
- `msoLastModifiedToday`
- `msoLastModifiedYesterday`
Example

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.
Remarks

The **LastRefreshed** property raises an error if the **Refresh** method has never been called.
Example

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.

```vba
On Error GoTo err_NeverRefreshed
If DateDiff("s", ActiveWorkbook.SharedWorkspace.LastRefreshed, Now) > 180 Then
    ActiveWorkbook.SharedWorkspace.Refresh
End If
Exit Sub

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.
Remarks

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.
Example

The following example alerts the user and displays the sync status if more than 24 hours have elapsed since the LastSyncTime.

```vbnet
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 Information"
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 command bar, 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 command bar control (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.
Example

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.

```vba
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.

```vba
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.
Example

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.
Remarks

This property applies only to custom document properties; you cannot use it with built-in document properties.

The source of the specified link is defined by the container application.

Setting the LinkSource property sets the LinkToContent property to True.
Example

This example displays the linked status of a custom document property. For the example to work, dp must be a valid DocumentProperty object.

```vba
Sub DisplayLinkStatus(dp As DocumentProperty)
    Dim stat As String, tf As String
    If dp.LinkToContent Then
        tf = ""
    Else
        tf = "not 
    End If
    stat = "This property is " & tf & "linked"
    If dp.LinkToContent Then
        stat = stat + Chr(13) & "The link source is " & dp.LinkSource
    End If
    MsgBox stat
End Sub
```
LinkToContent Property

**True** if the value of the custom document property is linked to the content of the container document. **False** if the value is static. Read/write **Boolean**.
Remarks

This property applies only to custom document properties. For built-in document properties, the value of this property is **False**.

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

This example displays the linked status of the custom document property. For the example to work, dp must be a valid DocumentProperty object.

Sub DisplayLinkStatus(dp As DocumentProperty)
    Dim stat As String, tf As String
    If dp.LinkToContent Then
        tf = ""
    Else
        tf = "not "
    End If
    stat = "This property is " & tf & "linked"
    If dp.LinkToContent Then
        stat = stat + Chr(13) & "The link source is " & dp.LinkSource
    End If
    MsgBox stat
End Sub
List Property

Returns or sets an item in the command bar combo box control. 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.
Example

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.

```vba
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 combo box control. Read-only Long.
Example

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 combo box control that appears above the separator line. Read/write Long.

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

A `ListHeaderCount` property value of – 1 indicates that there's no separator line in the combo box control.
Example

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.

```vba
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 [combo box control](#). If nothing is selected in the list, this property returns zero. Read/write **Long**.

*Note*  This property fails when applied to controls other than list controls.
Remarks

Setting the **ListIndex** property causes the specified control to select the given item and execute the appropriate action in the application.
Example

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.

```vba
Sub processSelection()
    Dim userChoice As Long
    userChoice = CommandBars("My Custom Bar").Controls(1).ListIndex
    Select Case userChoice
        Case 1
            chartcourse
        Case 2
            displaygraph
        Case Else
            MsgBox ("Invalid choice. Please choose again.")
    End Select
End Sub
```
Location Property

Returns the location of the script anchor in the specified HTML document. Read-only MsoScriptLocation.

MsoScriptLocation can be one of these MsoScriptLocation constants.

msoScriptLocationInBody
msoScriptLocationInHead
Remarks

Script tags in an HTML document can appear anywhere between the <HTML> tags in the document. In Microsoft Word, Excel, or PowerPoint, only the script anchors located between the <BODY> tags are visible. Additional HTML script that appears before or after the <BODY> tags is stored in but isn’t visible to the user.

The Scripts collection contains all of the script anchors that appear in the document, whether inside or outside of the main body of the document. Using the Location argument of the Add method, you can insert script anchors within the <HEAD> and <BODY> tags in the HTML document. You can also use the Location property to determine where a particular script anchor is stored within the document.
Example

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.

```vba
If ActiveWorkbook.Worksheets(1).Scripts(1).Location = 1 Then
    MsgBox "The script is located in the header."
End If
If ActiveWorkbook.Worksheets(1).Scripts(1).Location = 2 Then
    MsgBox "The script is 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.
Example

This example searches the My Documents folders for all files that begin with "Cmd" and displays the name and location of each file that's found.

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

Returns an **IPictureDisp** object representing the mask image of a **CommandBarButton** object. The mask image determines what parts of the button image are transparent.

*expression*.Mask

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

When you create an image that you plan on using as a mask image, all of the areas that you want to be transparent should be white, and all of the areas that you want to show should be black.

Always set the mask after you have set the picture for a CommandBarButton object.
Example

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")
    Set picMask = stdole.StdFunctions.LoadPicture("c:\images\mask.bmp")
    With Application.CommandBars.FindControl(msoControlButton)
        'Reference the first button on the first command bar
        'using a With...End With block.
        '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 object
        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**.
Remarks

This property is available only if the file Mswds_en.lex has been installed and registered. Note that this file isn't installed as part of a Typical setup.
Example

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**.
Example

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.
Example

The following example lists the members in the current shared workspace.

```vba
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
```
Show All
MenuAnimationStyle Property

Returns or sets the way a command bar is animated. Read/write MsoMenuAnimation.

MsoMenuAnimation can be one of these MsoMenuAnimation constants.
- msoMenuAnimationNone
- msoMenuAnimationRandom
- msoMenuAnimationSlide
- msoMenuAnimationUnfold
Example

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
Remarks

If the Mode property for a balloon is set to msoModeModeless, the user can work in the application while the balloon is visible. If the property is set to msoModeModal, the user must dismiss the balloon before 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.
Example

This example creates a balloon with an alert icon that instructs the user to select a printer. Because the balloon is modeless, the user has access to printer commands while the balloon is visible. 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.
Remarks

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.
Example

The following example displays the **Modified** date and time along with other properties of each version of a shared document.

```vba
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
```
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.
Remarks

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.
**Example**

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.

```vba
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 Information"
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.

```vba
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.ModifiedBy <> swsOwner.Name Then
        strMemberFiles = strMemberFiles & swsFile.URL & vbCrLf
    End If
Next
MsgBox "These files were last modified by other users:" & _
    vbCrLf & strMemberFiles, _
```
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.
Remarks

The SharedWorkspaceFolder and SharedWorkspaceMember objects do not have a ModifiedDate property.
Example

The following example returns a list of shared workspace files whose date and time last modified is earlier than today.

```vba
Dim swsFile As Office.SharedWorkspaceFile
Dim dtmMidnight As Date
Dim dtmFileDate As Date
Dim strOlderFiles As String

dtmMidnight = CDate(FormatDateTime(Now, vbShortDate) & " 12:00:0"
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**.
Remarks

The **MouseTips** property corresponds to the **Using the mouse more effectively** option under **Show tips about** on the **Options** tab in the **Office Assistant** dialog box.
Example

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**.
 Remarks

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.
Example

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.
Remarks

**Name property as it applies to the CommandBar object.**

The local name of a built-in command bar is displayed in the title bar (when the command bar isn't docked) and in the list of available command bars— wherever that list is displayed in the container application.

For a built-in command bar, the **Name** property returns the command bar’s U.S. English name. Use the **NameLocal** property to return the localized name.

If you change the value of the **LocalName** property for a custom command bar, the value of **Name** changes as well, and vice versa.

**Name property 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**.
Example

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.

```vba
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.

```vba
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 command bar as it's displayed in the language version of the container application, or returns or sets the name of a custom command bar. Read/write String.

**Note** If you attempt to set this property for a built-in command bar, an error occurs.
Remarks

The local name of a built-in command bar is displayed in the title bar (when the command bar isn't docked) and in the list of available command bars, wherever that list is displayed in the container application.

If you change the value of the **LocalName** property for a custom command bar, the value of **Name** changes as well, and vice versa.
Example

This example displays the name and localized name of the first command bar in the container application.

With CommandBars(1)
    MsgBox "The name of the command bar is " & .Name
    MsgBox "The localized name of the command bar is " & .NameLocal
End With
NewBalloon Property

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

Creates an Office Assistant balloon. Returns a Balloon object. Read-only.
Example

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

With Assistant.NewBalloon
    .Heading = "Regional Sales Data"
    .Text = "Select 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.

```vba
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.Description & vbCrLf
    End If
Next
MsgBox "Building Links: " & vbCrLf & strBuildingLinks, _
    vbInformation + vbOKOnly, "Building Links in Shared Workspace"
```
Object Property

Sets or returns the object that is the basis for the specified COMAddIn object. Read/write Object.
Remarks

This property is primarily used for enabling one COMAddIn to communicate with another COMAddIn.
Example

The following example returns the object associated with the COM add-in `msodraa9.ShapeSelect`.

```vba
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 pop-up control belongs to when the menu groups of the OLE server are merged with the menu groups of an OLE client (that is, when an object of the container application type is embedded in another application). Read/write MsoOLEMenuGroup.

MsoOLEMenuGroup can be one of these MsoOLEMenuGroup constants.

msoOLEMenuGroupContainer
msoOLEMenuGroupEdit
msoOLEMenuGroupFile
msoOLEMenuGroupHelp
msoOLEMenuGroupNone
msoOLEMenuGroupObject
msoOLEMenuGroupWindow

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

This property is intended to allow add-in applications to specify how their command bar controls will be represented in the Office application. If either the container or the server does not implement command bars, normal OLE menu merging will occur: the menu bar will be merged, as well as all the toolbars from the server, and none of the toolbars from the container. This property is relevant only for pop-up controls on the menu bar because menus are merged on the basis of their menu group category.

If both of the merging applications implement command bars, command bar controls are merged according to the OLEUsage property.
**Example**

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**.

```vba
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 command bar control will be used when two Microsoft Office applications are merged. Read/write MsoControlOLEUsage.

MsoControlOLEUsage can be one of these MsoControlOLEUsage constants.

msoControlOLEUsageBoth
msoControlOLEUsageClient
msoControlOLEUsageNeither
msoControlOLEUsageServer
Remarks

This property is intended to allow you to specify how individual add-in applications’ command bar controls will be represented in one Office application when it is merged with another Office application. If both the client and server implement command bars, the command bar controls are embedded in the client control by control. Custom controls marked as client-only (or neither client nor server) are dropped from the server, and controls marked as server-only (or neither server nor client) are dropped from the client. The remaining controls are merged.

If one of the merging applications isn't an Office application, normal OLE menu merging is used, which is controlled by the OLEMenuGroup property.
Example

This example adds a new button to the command bar named Tools, and sets its OLEUsage property.

Set myControl = CommandBars("Tools").Controls.Add(Type:=msoControlButton, Temporary:=True)
myControl.OLEUsage = msoControlOLEUsageNeither
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.
Example

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 user enables the Assistant, the Assistant appears and performs the animation **msoAnimationGreeting**.

```vba
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 command bar control. Read/write String.

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

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.
Remarks

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).
Example

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.
Example

This example displays the name of the parent object for a document property. You must pass a valid DocumentProperty object to the procedure.

Sub DisplayParent(dp as DocumentProperty)
    MsgBox dp.Parent.Name
End Sub
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.
**Example**

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.

```vba
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.
Remarks

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.
Example

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.

```vbnet
Dim irmPermission As Office.Permission
Dim irmUserPerm As Office.UserPermission
Set irmPermission = ActiveWorkbook.Permission
Set irmUserPerm = irmPermission.Item(2)
If irmUserPerm.Permission And Office.msoPermissionSave Then
    MsgBox "User " & irmUserPerm.UserId & " has permission to save this document.", vbInformation + vbOKOnly, "IRM Information"
Else
    MsgBox "User " & irmUserPerm.UserId & " does NOT have permission to save this document.", vbInformation + vbOKOnly, "IRM Information"
End If
Set irmUserPerm = Nothing
Set irmPermission = Nothing
```
PermissionFromPolicy Property

Returns a **Boolean** value that indicates whether a permission policy has been applied to the active document. Read-only **Boolean**.

`expression.PermissionFromPolicy`

`expression` | Required. An expression that returns a **Permission** object.
Remarks

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.
**Example**

The following example displays permission policy information about the active document.

```vba
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.PolicyDescription
    Else
        strIRMInfo = strIRMInfo & " Default permissions applied."
            " Default policy name: " & irmPermission.PolicyName
            " Default policy description: " & irmPermission.PolicyDescription
    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.
Remarks

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.
Example

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")
    Set picMask = stdole.StdFunctions.LoadPicture("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.
Remarks

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.
Example

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.PolicyDescription
  Else
    strIRMInfo = strIRMInfo & " Default permissions applied.
      " Default policy name: " & irmPermission.PolicyName
      " Default policy description: " & irmPermission.PolicyDescription
  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.
Remarks

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.
**Example**

The following example displays permission policy information about the active document.

```vba
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.PolicyDescription
    Else
        strIRMInfo = strIRMInfo & " Default permissions applied."
        " Default policy name: " & irmPermission.PolicyName
        " Default policy description: " & irmPermission.PolicyDescription
    End If
Else
    strIRMInfo = "Permission are NOT restricted on this document"
End If
MsgBox strIRMInfo, vbInformation + vbOKOnly, "IRM Information"
Set irmPermission = Nothing
```
Show All
Position Property

Returns or sets the position of a command bar. Read/write MsoBarPosition.

MsoBarPosition can be one of these MsoBarPosition constants.

msoBarBottom
msoBarFloating
msoBarLeft
msoBarMenuBar
msoBarPopup
msoBarRight
msoBarTop
Example

This example steps through the collection of command bars, docking the custom command bars at the bottom of the application window and docking the built-in command bars at the top of the window.

For Each bar In CommandBars
    If bar.Visible = True Then
        If bar.BuiltIn Then
            bar.Position = msoBarTop
        Else
            bar.Position = msoBarBottom
        End If
    End If
Next
Priority Property

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

Returns or sets the priority of a command bar control. 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.
Remarks

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.
**Remarks**

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.
Example

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.

```vba
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
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.)

```vba
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**.
Remarks

This property is helpful if you run the same callback procedure from more than one balloon.
Example

This example identifies the Office Assistant balloon by setting the **Private** property to 129.

```vba
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**.
Example

The following example displays the ProgID and GUID for COM add-in one in a message box.

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

Returns the PropertyTests collection that represents all the search criteria for a file search. Read-only.

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

This example displays all the search criteria for the first property test in the collection.

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

Sets or returns the proportional font setting in the host application. Read/write String.
Remarks

When you set the `ProportionalFont` property, the host application does not check the value for validity.
Example

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.
Remarks

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.
Example

This example sets the proportional font and proportional font size for the English/Western European/Other Latin Script character set in the active application.

```vba
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 command bar is protected from user customization. Read/write MsoBarProtection.

MsoBarProtection can be one of these MsoBarProtection constants.

- msoBarNoChangeDock
- msoBarNoChangeVisible
- msoBarNoCustomize
- msoBarNoHorizontalDock
- msoBarNoMove
- msoBarNoProtection
- msoBarNoResize
- msoBarNoVerticalDock
Remarks

Using the constant `msoBarNoCustomize` prevents users from accessing the Add or Remove Buttons menu (this menu enables users to customize a toolbar).
Example

This example steps through the collection of command bars to find the command bar named "Forms." If this command bar is found, it's docking state is protected and it's made visible.

foundFlag = False
For i = 1 To CommandBars.Count
    If CommandBars(i).Name = "Forms" Then
        CommandBars(i).Protection = msoBarNoChangeDock
        CommandBars(i).Visible = True
        foundFlag = True
    End If
Next
If Not foundFlag Then
    MsgBox "'Forms' command bar is not in the collection."
End If
Reduced Property

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

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

This property is not used in Microsoft Office.
Example

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

With Assistant
    .Reduced = True
    .Left = 400
    .MoveWhenInTheWay = True
    .TipOfDay = True
    .Visible = True
    .Animation = msoAnimationGreeting
End With
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.
Remarks

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`
Example

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 & vbCrLf & _
        " Users with permissions: " & irmPermission.Count & vbCrLf & _
        " Cache licenses locally: " & irmPermission.StoreLicense & vbCrLf & _
        " Request permission URL: " & irmPermission.RequestPermissionURL & vbCrLf
    If irmPermission.PermissionFromPolicy Then
        strIRMInfo = strIRMInfo & " Permissions applied from policy:
            Policy name: " & irmPermission.PolicyName & vbCrLf & _
            Policy description: " & irmPermission.PolicyDescription
    Else
        strIRMInfo = strIRMInfo & " Default permissions applied."
    End If
Else
    strIRMInfo = "Permissions are NOT restricted on this document"
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.
Example

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

Sub OfficeFilters()
    Dim appOffice As OfficeDataSourceObject
    Dim appFilters As ODSOFilters

    Set appOffice = Application.OfficeDataSourceObject
    appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName;UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"

    Set appFilters = appOffice.Filters

    MsgBox appOffice.RowCount

    appFilters.Add Column:="Region", Comparison:=msoFilterComparisonEqual, Conjunction:=msoFilterConjunctionAnd, bstrCompareTo:="WA"
    appOffice.ApplyFilter

    MsgBox appOffice.RowCount

End Sub
RowIndex Property

Returns or sets the docking order of a command bar in relation to other command bars in the same docking area. Can be an integer greater than zero, or either of the following MsoBarRow constants: msoBarRowFirst or msoBarRowLast. Read/write Long.
Remarks

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.
Example

This example adjusts the position of the command bar named "Custom" by moving it to the left 110 pixels more than the default, and it makes this command bar the first to be docked by changing its row index to `msoBarRowFirst`.

```vba
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 `ScopeFolder` object.

`expression.ScopeFolder`

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

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.

```vba
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 `ScopeFolders` 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.
Example

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
                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.
Example

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 SearchFolders collection.

expression.SearchFolders

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

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 collection: " & Application.FileSearch.SearchFolders.Count
SearchScopes Property

Returns a `SearchScopes` collection.

`expression.SearchScopes`

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

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 scope."
                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
SearchSubFolders Property

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

This example searches the My Documents folder and all of its subfolders for all files whose names begin with "Cmd." The example also displays the name and location of each file that's found.

```vba
Set fs = Application.FileSearch
With fs
    .LookIn = "C:\My Documents"
    .SearchSubFolders = True
    .FileName = "cmd*"
    If .Execute() > 0 Then
        MsgBox "There were " & .FoundFiles.Count & " file(s) found."
        For i = 1 To .FoundFiles.Count
            MsgBox .FoundFiles(i)
        Next i
    Else
        MsgBox "There were no files found."
    End If
End With
```
SearchWhenProgramming Property

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

**True** if the Office Assistant displays application and programming Help while the user is working in Visual Basic. Read/write **Boolean**.
Remarks

The default value is **False**. The `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.
Example

This example allows the user to search both application and programming help while working in Visual Basic.

Assistant.SearchWhenProgramming = True
SecondValue Property

Returns an optional second value property test (as in a range) for the file search. Read-only Variant.
Remarks

This property is intended to be used to specify a range, and it can only be used with the `MsoCondition` constant `msoConditionAnyTimeBetween` or `msoConditionAnyNumberBetween`.
Example

This example displays the second value of the search criteria (if it exists) in a dialog box. If the second value doesn't exist, the example displays another message.

```vbnet
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 `FileDialog.SelectedItems` 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.
Example

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 If
    End With

    'Set the object variable to Nothing.
    Set fd = Nothing

End Sub
Shape Property

Returns a **Shape** object or **InlineShape** object, depending on the Microsoft Office host application. Read-only **Object**.
Remarks

The **Shape** property returns a **Shape** object in Microsoft Excel and PowerPoint. In Word, the **Shape** property returns a **Shape** object if the script anchor is floating; if it’s an inline anchor, however, this property returns an **InlineShape** object.
Example

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 button control when the button appears on a menu, submenu, or shortcut menu. Read/write String.
Remarks

You can set this property only for command bar buttons that contain an OnAction macro.
Example

This example displays the shortcut text for the **Open** command (**File** menu) on the Microsoft Excel Worksheet menu bar in a message box.

```vba
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.
Example

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

        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 disk
    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.
Example

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.

```vba
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 to disk.
    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 collection.
    Else
        sig.Delete
        MsgBox "Not signed"
        AddSignature = False
    End If

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

Exit Function
```

Error_Handler:
    MsgBox "Error: " & Err.Description
    On Error GoTo 0
```
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 SmartDocument object.
Remarks

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.
Example

The following example determines whether an XML expansion pack is attached to the active Excel workbook by checking the **SolutionID** property.

```vba
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 absolute URL 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 SmartDocument object.
Remarks

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.
Example

The following example determines whether an XML expansion pack is attached to the active Word document, then displays the smart document's Solution URL.

```vba
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.
Remarks

The default value is **True**. The **Sounds** property corresponds to the **Make sounds** option under **Use the Office Assistant** on the **Options** tab in the **Office Assistant** dialog box. If a sound card is not installed, this property has no effect.
Example

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.
Remarks

The Microsoft Windows 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.
Show All
State Property

State property as it applies to the **CommandBarButton** object.

Returns or sets the appearance of a command bar *button control*. Read/write **MsoButtonState**. 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.
Example

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**.

```vba
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)
- `msoSharedWorkspaceTaskStatusDeferred` (2)
- `msoSharedWorkspaceTaskStatusInProgress` (1)
- `msoSharedWorkspaceTaskStatusNotStarted` (4)
- `msoSharedWorkspaceTaskStatusWaiting` (5)

`expression.Status`

`expression` Required. An expression that returns a `SharedWorkspaceTask` object.
Remarks

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 MsoSyncStatusType.

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.
Remarks

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
Example

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.

```vba
Dim swsTask As Office.SharedWorkspaceTask
Dim strTaskStatus As String
For Each swsTask In ActiveWorkbook.SharedWorkspace.Tasks
    If swsTask.Status <> msoSharedWorkspaceTaskStatusCompleted Then
        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.

```vba
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
    End Select
```

```vba
MsgBox strStatus, vbInformation + vbOKOnly, "Sync Status"
```
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.
Remarks

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**.
Example

The following example displays information about the permissions settings of the active document, including the StoreLicenses setting.

```vba
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.StoreLicenses & _
        " Request permission URL: " & irmPermission.RequestPermissionURL & vbCrLf
    If irmPermission.PermissionFromPolicy Then
        strIRMInfo = strIRMInfo & " Permissions applied from policy:"
        " Policy name: " & irmPermission.PolicyName & vbCrLf & _
        " Policy description: " & irmPermission.PolicyDescription & ""
    Else
        strIRMInfo = strIRMInfo & " Custom permissions applied."
    End If
Else
    strIRMInfo = "Permissions are NOT restricted on this document."
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 button control is displayed. Read/write MsoButtonStyle.

MsoButtonStyle can be one of these MsoButtonStyle constants:
  msoButtonAutomatic
  msoButtonCaption
  msoButtonIcon
  msoButtonIconAndCaption
  msoButtonIconAndCaptionBelow
  msoButtonIconAndWrapCaption
  msoButtonIconAndWrapCaptionBelow
  msoButtonWrapCaption

Style property as it applies to the CommandBarComboBox object.

Returns or sets the way a command bar combo box control is displayed. Can be either of the following MsoComboStyle constants: msoComboBoxLabel or msoComboBoxNormal. Read/write MsoComboStyle.

MsoComboBoxStyle can be one of these MsoComboBoxStyle constants:
  msoComboBoxLabel
  msoComboBoxNormal
Example

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 = msoComboBoxLabel
    .AddItem "vanilla"
    .AddItem "chocolate"
    .AddItem "cookie dough"
  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.
Example

This example sets the name of the table if the table name is currently blank.

Sub OfficeTest()
    Dim appOffice As OfficeDataSourceObject
    Set appOffice = Application.OfficeDataSourceObject
    appOffice.Open bstrConnect:="DRIVER=SQL Server;SERVER=ServerName
    "UID=user;PWD=;DATABASE=Northwind", bstrTable:="Employees"
    If appOffice.Table = "" Then
        appOffice.Table = "Employees"
    End If
End Sub
Tag Property

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

expression.Tag

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

To avoid duplicate calls of the same class when triggered 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, CancelDefault As Boolean)
  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").Controls.Add(msoControlButton)
    With oEvt.oBtn
      Caption = "Btn" & i
      Style = msoButtonCaption
      .Tag = "Hello" & i
    End With
    oBtns.Add oEvt
  Next

End Sub
Example

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.
Example

The following example lists the tasks in the current shared workspace.

```vba
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 [combo box control]. Read/write **String**.

**HTMLProjectItem** object: Returns or sets the HTML text in the HTML editor. Read/write **String**.

`expression.Text`

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

For the **Balloon**, **BalloonLabel**, and **BalloonCheckbox** objects, you can specify that a particular graphic be displayed by using the following syntax: `{type location sizing_factor}`, where `type` is bmp (bitmap) or wmf (Windows metafile); `location` is the resource ID or the path and file name; and `sizing_factor` denotes the width of the .wmf file (`sizing_factor` is omitted for .bmp files).

The **Balloon** object also supports underlined text and text that has one of the 16 system palette colors applied to it. To display underlined text, use the syntax `{ul}` or `{ul 1}`; use `{ul 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.

<table>
<thead>
<tr>
<th>System color number</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Black</td>
</tr>
<tr>
<td>1</td>
<td>Dark red</td>
</tr>
<tr>
<td>2</td>
<td>Dark green</td>
</tr>
<tr>
<td>3</td>
<td>Dark yellow</td>
</tr>
<tr>
<td>4</td>
<td>Dark blue</td>
</tr>
<tr>
<td>5</td>
<td>Dark magenta</td>
</tr>
<tr>
<td>6</td>
<td>Dark cyan</td>
</tr>
<tr>
<td>7</td>
<td>Light gray</td>
</tr>
<tr>
<td>248</td>
<td>Medium gray</td>
</tr>
<tr>
<td>249</td>
<td>Red</td>
</tr>
<tr>
<td>250</td>
<td>Green</td>
</tr>
<tr>
<td>251</td>
<td>Yellow</td>
</tr>
<tr>
<td>252</td>
<td>Blue</td>
</tr>
<tr>
<td>253</td>
<td>Magenta</td>
</tr>
<tr>
<td>254</td>
<td>Cyan</td>
</tr>
<tr>
<td>255</td>
<td>White</td>
</tr>
</tbody>
</table>

If you specify a number other than one of the preceding system color numbers, the text in the Office Assistant balloon is black.
**Example**

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.

```vba
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 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.

```vba
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.

```vba
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."
```
This example creates a new Office Assistant balloon that contains a Windows metafile.

```
With Assistant.NewBalloon
    .Heading = "Underlined {ul 1}Heading{ul 0}"
    .Text = "{WMF ""C:\Favorites\MyPicture.WMF""}"
    .Show
End With
```
TextOrProperty Property

Returns or sets the word or phrase to be searched for, in either the body of a file or the file's properties, during the file search. The word or phrase can include the * (asterisk) or ? (question mark) wildcard character. Read/write String.
Remarks

Use the question mark wildcard character to match any single character. For example, type gr??y to find all files that contain at least one instance of either "gray" or "grey."

Use the asterisk wildcard character to match any number of characters. For example, type San* to return all files that contain at least one word that begins with "San."
Example

This example searches the C:\My Documents folder and all of its subfolders and returns all files whose body text or file properties contain any words that begin with "San." The TextOrProperty property sets the word to be searched for and limits the search to either the body of the file or the file properties.

```
With Application.FileSearch
    .NewSearch
    .LookIn = "C:\My Documents"
    .SearchSubFolders = True
    .TextOrProperty = "San*"
    .FileType = msoFileTypeAllFiles
End With
```
TipOfDay Property

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

**True** if the Office Assistant displays a special tip each time the Office application is opened. Read/write **Boolean**.
Remarks

The default value is False. TipOfDay property corresponds to the **Show the Tip of the Day at startup** option under **Show tips about** on the **Options** tab in the **Office Assistant** dialog box.
Example

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.
Remarks

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.
Example

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 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.

```vbnet
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*. 
Remarks

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

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 command bar, 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 command bar control to the top edge of the screen. Read-only Long.

expression.Top

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

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.

With Assistant
  .On = True
  .Visible = True
  .Sounds = True
  .Animation = msoAnimationBeginSpeaking
End With
Assistant.Top = 100
MsgBox "Click OK to move the Assistant to a " & _
  "new location."
Assistant.Top = 500

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 command bar. Read-only MsoBarType.

MsoBarType can be one of these MsoBarType constants.
- msoBarTypeMenuBar
- msoBarTypeNormal
- msoBarTypePopup

expression.Type

expression Required. An expression that returns a CommandBar object.

Type property as it applies to the CommandBarButton, CommandBarComboBox, CommandBarControl, and CommandBarPopup objects.

Returns the type of command bar control. Read-only MsoControlType.

MsoControlType can be one of these MsoControlType constants.
- msoControlActiveX
- msoControlAutoCompleteCombo
- msoControlButton
- msoControlButtonDropdown
- msoControlButtonPopup
- msoControlComboBox
- msoControlCustom
- msoControlDropdown
- msoControlEdit
- msoControlExpandingGrid
- msoControlGauge
expression.Type

description

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

description

expression Required. An expression that returns a SearchScope object.
Type property as it applies to the DocumentProperty object.

Returns or sets the document property type. Read-only for built-in document properties; read/write for custom document properties.

expression.Type

expression  Required. An expression that returns a DocumentProperty object.
Remarks

The return value will be a MsoDocProperties constant.

MsoDocProperties can be one of these MsoDocProperties constants.

- msoPropertyTypeBoolean
- msoPropertyTypeDate
- msoPropertyTypeFloat
- msoPropertyTypeNumber
- msoPropertyTypeString
Example

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.

```vba
Set oldCtrl = CommandBars("Custom").Controls(1)
If oldCtrl.Type = msoControlBarButton Then
    Set newCtrl = CommandBars.FindControl(Type:= _
        MsoControlBarButton, 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.

```vba
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 Uniform Resource Locator (URL) of the shared workspace. Read-only `String`.

`expression.URL`

`expression`  Required. An expression that returns a `SharedWorkspace` object.
Remarks

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.

```vba
Private Function URLDecode(URLtoDecode As String) As String
    URLDecode = Replace(URLtoDecode, "%20", " ")
End Function
```

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

Returns the full **Uniform Resource Locator (URL)** and file name of the shared workspace file. Read-only **String**.

**expression**.**URL**

**expression** Required. An expression that returns a **SharedWorkspaceFile** object.
**Remarks**

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.

```vba
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 **Uniform Resource Locator (URL)** of the link saved in the shared workspace. Read/write **String**.

*expression*. **URL**

*expression* Required. An expression that returns a **SharedWorkspaceLink** object.
Remarks

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.

```vba
Private Function URLDecode(URLtoDecode As String) As String
    URLDecode = Replace(URLtoDecode, "%20", " ")
End Function
```
Example

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.
Remarks

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.
**Example**

The following example lists the users who have permissions on the active document.

```vba
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.
Remarks

If the container application doesn't define a value for one of the built-in document properties, reading the **Value** property for that document property causes an error.

Value property 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.
Example

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.

```vba
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.

```vba
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.
Remarks

The **Visible** property for newly created custom command bars is **False** by default.

The **Enabled** property for a command bar must be set to **True** before the visible property is set to **True**.
Example

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

foundFlag = False
For Each cmdBar In CommandBars
    If cmdBar.Name = "Forms" Then
        cmdBar.Protection = msoBarNoChangeDock
        cmdBar. Visible = True
        foundFlag = True
    End If
Next
If Not foundFlag Then
    MsgBox "'Forms' command bar is not in the collection."
End If

This example makes the Office Assistant visible and sets its animation.

With Application.Assistant
    .Visible = True
    .Sounds = True
    .Animation = msoAnimationBeginSpeaking
End With
**Width Property**

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

`expression.Width`

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

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.

```vba
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 runtime error.
Example

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.
Example

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()
Private Sub ComboBoxEvent_Change(ByVal Ctrl As Office.CommandBarComboBox)
    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.

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ctrl</td>
<td>Required <code>CommandBarButton</code>. Denotes the <code>CommandBarButton</code> control that initiated the event. Required <code>Boolean</code>. <code>False</code> if the default behavior associated with the <code>CommandBarButton</code> control occurs, unless it’s canceled by another process or add-in.</td>
</tr>
<tr>
<td>CancelDefault</td>
<td></td>
</tr>
</tbody>
</table>
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.
Example

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 comma-separated 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 Delimited)" Then fBtnExists = True
    Next

    Dim btnSaveAsCSV As Office.CommandBarButton
    If fBtnExists Then
        Set btnSaveAsCSV = barHelp.Controls("Save As CSV (Comma Delimited)"
    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.
**Example**

The following example sets up event-handling routines for the `MsoEnvelope` object.

```vba
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_EnvelopeShow()**

*object*  A variable which references an object of type `MsoEnvelope` declared with events in a class module.
**Example**

The following example sets up event-handling routines for the `MsoEnvelope` object.

```vba
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.

### MailFormat

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>mfHTML</td>
<td>2</td>
</tr>
<tr>
<td>mfPlainText</td>
<td>1</td>
</tr>
<tr>
<td>mfRTF</td>
<td>3</td>
</tr>
</tbody>
</table>

### MsoAlertButtonType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoAlertButtonAbortRetryIgnore</td>
<td>2</td>
</tr>
<tr>
<td>msoAlertButtonOK</td>
<td>0</td>
</tr>
<tr>
<td>msoAlertButtonOKCancel</td>
<td>1</td>
</tr>
<tr>
<td>msoAlertButtonRetryCancel</td>
<td>5</td>
</tr>
<tr>
<td>msoAlertButtonYesAllNoCancel</td>
<td>6</td>
</tr>
<tr>
<td>msoAlertButtonYesNo</td>
<td>4</td>
</tr>
<tr>
<td>msoAlertButtonYesNoCancel</td>
<td>3</td>
</tr>
</tbody>
</table>

### MsoAlertCancelType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoAlertCancelDefault</td>
<td>-1</td>
</tr>
<tr>
<td>msoAlertCancelFifth</td>
<td>4</td>
</tr>
<tr>
<td>msoAlertCancelFirst</td>
<td>0</td>
</tr>
<tr>
<td>msoAlertCancelFourth</td>
<td>3</td>
</tr>
<tr>
<td>msoAlertCancelSecond</td>
<td>1</td>
</tr>
<tr>
<td>Constant</td>
<td>Value</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>msoAlertDefaultFifth</td>
<td>4</td>
</tr>
<tr>
<td>msoAlertDefaultFirst</td>
<td>0</td>
</tr>
<tr>
<td>msoAlertDefaultFourth</td>
<td>3</td>
</tr>
<tr>
<td>msoAlertDefaultSecond</td>
<td>1</td>
</tr>
<tr>
<td>msoAlertDefaultThird</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoAlertIconCritical</td>
<td>1</td>
</tr>
<tr>
<td>msoAlertIconInfo</td>
<td>4</td>
</tr>
<tr>
<td>msoAlertIconNoIcon</td>
<td>0</td>
</tr>
<tr>
<td>msoAlertIconQuery</td>
<td>2</td>
</tr>
<tr>
<td>msoAlertIconWarning</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoAlignBottoms</td>
<td>5</td>
</tr>
<tr>
<td>msoAlignCenters</td>
<td>1</td>
</tr>
<tr>
<td>msoAlignLefts</td>
<td>0</td>
</tr>
<tr>
<td>msoAlignMiddles</td>
<td>4</td>
</tr>
<tr>
<td>msoAlignRights</td>
<td>2</td>
</tr>
<tr>
<td>msoAlignTops</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoAnimationAppear</td>
<td>32</td>
</tr>
</tbody>
</table>
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
**MsoAppLanguageID**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoLanguageIDExeMode</td>
<td>4</td>
</tr>
<tr>
<td>msoLanguageIDHelp</td>
<td>3</td>
</tr>
<tr>
<td>msoLanguageIDInstall</td>
<td>1</td>
</tr>
<tr>
<td>msoLanguageIDUI</td>
<td>2</td>
</tr>
<tr>
<td>msoLanguageIDUIPrevious</td>
<td>5</td>
</tr>
</tbody>
</table>

**MsoArrowheadLength**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoArrowheadLengthMedium</td>
<td>2</td>
</tr>
<tr>
<td>msoArrowheadLengthMixed</td>
<td>-2</td>
</tr>
<tr>
<td>msoArrowheadLong</td>
<td>3</td>
</tr>
<tr>
<td>msoArrowheadShort</td>
<td>1</td>
</tr>
</tbody>
</table>

**MsoArrowheadStyle**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoArrowheadDiamond</td>
<td>5</td>
</tr>
<tr>
<td>msoArrowheadNone</td>
<td>1</td>
</tr>
<tr>
<td>msoArrowheadOpen</td>
<td>3</td>
</tr>
<tr>
<td>msoArrowheadOval</td>
<td>6</td>
</tr>
<tr>
<td>msoArrowheadStealth</td>
<td>4</td>
</tr>
<tr>
<td>msoArrowheadStyleMixed</td>
<td>-2</td>
</tr>
<tr>
<td>msoArrowheadTriangle</td>
<td>2</td>
</tr>
</tbody>
</table>

**MsoArrowheadWidth**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoArrowheadNarrow</td>
<td>1</td>
</tr>
</tbody>
</table>
msoArrowheadWide 3
msoArrowheadWidthMedium 2
msoArrowheadWidthMixed -2

MsoAutomationSecurity

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoAutomationSecurityByUI</td>
<td>2</td>
</tr>
<tr>
<td>msoAutomationSecurityForceDisable</td>
<td>3</td>
</tr>
<tr>
<td>msoAutomationSecurityLow</td>
<td>1</td>
</tr>
</tbody>
</table>

MsoAutoShapeType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoShape16pointStar</td>
<td>94</td>
</tr>
<tr>
<td>msoShape24pointStar</td>
<td>95</td>
</tr>
<tr>
<td>msoShape32pointStar</td>
<td>96</td>
</tr>
<tr>
<td>msoShape4pointStar</td>
<td>91</td>
</tr>
<tr>
<td>msoShape5pointStar</td>
<td>92</td>
</tr>
<tr>
<td>msoShape8pointStar</td>
<td>93</td>
</tr>
<tr>
<td>msoShapeActionButtonBackorPrevious</td>
<td>129</td>
</tr>
<tr>
<td>msoShapeActionButtonBeginning</td>
<td>131</td>
</tr>
<tr>
<td>msoShapeActionButtonCustom</td>
<td>125</td>
</tr>
<tr>
<td>msoShapeActionButtonDocument</td>
<td>134</td>
</tr>
<tr>
<td>msoShapeActionButtonEnd</td>
<td>132</td>
</tr>
<tr>
<td>msoShapeActionButtonForwardorNext</td>
<td>130</td>
</tr>
<tr>
<td>msoShapeActionButtonHelp</td>
<td>127</td>
</tr>
<tr>
<td>msoShapeActionButtonHome</td>
<td>126</td>
</tr>
<tr>
<td>msoShapeActionButtonInformation</td>
<td>128</td>
</tr>
<tr>
<td>msoShapeActionButtonMovie</td>
<td>136</td>
</tr>
<tr>
<td>msoShapeActionButtonReturn</td>
<td>133</td>
</tr>
<tr>
<td>msoShapeActionButtonSound</td>
<td>135</td>
</tr>
<tr>
<td>msoShapeArc</td>
<td>25</td>
</tr>
<tr>
<td>msoShapeBalloon</td>
<td>137</td>
</tr>
<tr>
<td>Shape Name</td>
<td>Number</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>msoShapeBentArrow</td>
<td>41</td>
</tr>
<tr>
<td>msoShapeBentUpArrow</td>
<td>44</td>
</tr>
<tr>
<td>msoShapeBevel</td>
<td>15</td>
</tr>
<tr>
<td>msoShapeBlockArc</td>
<td>20</td>
</tr>
<tr>
<td>msoShapeCan</td>
<td>13</td>
</tr>
<tr>
<td>msoShapeChevron</td>
<td>52</td>
</tr>
<tr>
<td>msoShapeCircularArrow</td>
<td>60</td>
</tr>
<tr>
<td>msoShapeCloudCallout</td>
<td>108</td>
</tr>
<tr>
<td>msoShapeCross</td>
<td>11</td>
</tr>
<tr>
<td>msoShapeCube</td>
<td>14</td>
</tr>
<tr>
<td>msoShapeCurvedDownArrow</td>
<td>48</td>
</tr>
<tr>
<td>msoShapeCurvedDownRibbon</td>
<td>100</td>
</tr>
<tr>
<td>msoShapeCurvedLeftArrow</td>
<td>46</td>
</tr>
<tr>
<td>msoShapeCurvedRightArrow</td>
<td>45</td>
</tr>
<tr>
<td>msoShapeCurvedUpArrow</td>
<td>47</td>
</tr>
<tr>
<td>msoShapeCurvedUpRibbon</td>
<td>99</td>
</tr>
<tr>
<td>msoShapeDiamond</td>
<td>4</td>
</tr>
<tr>
<td>msoShapeDonut</td>
<td>18</td>
</tr>
<tr>
<td>msoShapeDoubleBrace</td>
<td>27</td>
</tr>
<tr>
<td>msoShapeDoubleBracket</td>
<td>26</td>
</tr>
<tr>
<td>msoShapeDoubleWave</td>
<td>104</td>
</tr>
<tr>
<td>msoShapeDownArrow</td>
<td>36</td>
</tr>
<tr>
<td>msoShapeDownArrowCallout</td>
<td>56</td>
</tr>
<tr>
<td>msoShapeDownRibbon</td>
<td>98</td>
</tr>
<tr>
<td>msoShapeExplosion1</td>
<td>89</td>
</tr>
<tr>
<td>msoShapeExplosion2</td>
<td>90</td>
</tr>
<tr>
<td>msoShapeFlowchartAlternateProcess</td>
<td>62</td>
</tr>
<tr>
<td>msoShapeFlowchartCard</td>
<td>75</td>
</tr>
<tr>
<td>msoShapeFlowchartCollate</td>
<td>79</td>
</tr>
<tr>
<td>msoShapeFlowchartConnector</td>
<td>73</td>
</tr>
<tr>
<td>msoShapeFlowchartData</td>
<td>64</td>
</tr>
<tr>
<td>msoShapeFlowchartDecision</td>
<td>63</td>
</tr>
<tr>
<td>msoShapeFlowchartDelay</td>
<td>84</td>
</tr>
<tr>
<td>Shape Name</td>
<td>Number</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>msoShapeFlowchartDirectAccessStorage</td>
<td>87</td>
</tr>
<tr>
<td>msoShapeFlowchartDisplay</td>
<td>88</td>
</tr>
<tr>
<td>msoShapeFlowchartDocument</td>
<td>67</td>
</tr>
<tr>
<td>msoShapeFlowchartExtract</td>
<td>81</td>
</tr>
<tr>
<td>msoShapeFlowchartInternalStorage</td>
<td>66</td>
</tr>
<tr>
<td>msoShapeFlowchartMagneticDisk</td>
<td>86</td>
</tr>
<tr>
<td>msoShapeFlowchartManualInput</td>
<td>71</td>
</tr>
<tr>
<td>msoShapeFlowchartManualOperation</td>
<td>72</td>
</tr>
<tr>
<td>msoShapeFlowchartMerge</td>
<td>82</td>
</tr>
<tr>
<td>msoShapeFlowchartMultidocument</td>
<td>68</td>
</tr>
<tr>
<td>msoShapeFlowchartOffpageConnector</td>
<td>74</td>
</tr>
<tr>
<td>msoShapeFlowchartOr</td>
<td>78</td>
</tr>
<tr>
<td>msoShapeFlowchartPredefinedProcess</td>
<td>65</td>
</tr>
<tr>
<td>msoShapeFlowchartPreparation</td>
<td>70</td>
</tr>
<tr>
<td>msoShapeFlowchartProcess</td>
<td>61</td>
</tr>
<tr>
<td>msoShapeFlowchartPunchedTape</td>
<td>76</td>
</tr>
<tr>
<td>msoShapeFlowchartSequentialAccessStorage</td>
<td>85</td>
</tr>
<tr>
<td>msoShapeFlowchartSort</td>
<td>80</td>
</tr>
<tr>
<td>msoShapeFlowchartStoredData</td>
<td>83</td>
</tr>
<tr>
<td>msoShapeFlowchartSummingJunction</td>
<td>77</td>
</tr>
<tr>
<td>msoShapeFlowchartTerminator</td>
<td>69</td>
</tr>
<tr>
<td>msoShapeFoldedCorner</td>
<td>16</td>
</tr>
<tr>
<td>msoShapeHeart</td>
<td>21</td>
</tr>
<tr>
<td>msoShapeHexagon</td>
<td>10</td>
</tr>
<tr>
<td>msoShapeHorizontalScroll</td>
<td>102</td>
</tr>
<tr>
<td>msoShapeIsoscelesTriangle</td>
<td>7</td>
</tr>
<tr>
<td>msoShapeLeftArrow</td>
<td>34</td>
</tr>
<tr>
<td>msoShapeLeftArrowCallout</td>
<td>54</td>
</tr>
<tr>
<td>msoShapeLeftBrace</td>
<td>31</td>
</tr>
<tr>
<td>msoShapeLeftBracket</td>
<td>29</td>
</tr>
<tr>
<td>msoShapeLeftRightArrow</td>
<td>37</td>
</tr>
<tr>
<td>msoShapeLeftRightArrowCallout</td>
<td>57</td>
</tr>
<tr>
<td>msoShapeLeftRightUpArrow</td>
<td>40</td>
</tr>
<tr>
<td>msoShapeRegularPentagon</td>
<td>12</td>
</tr>
<tr>
<td>------------------------</td>
<td>----</td>
</tr>
<tr>
<td>msoShapeRightArrow</td>
<td>33</td>
</tr>
<tr>
<td>msoShapeRightArrowCallout</td>
<td>53</td>
</tr>
<tr>
<td>msoShapeRightBrace</td>
<td>32</td>
</tr>
<tr>
<td>msoShapeRightBracket</td>
<td>30</td>
</tr>
<tr>
<td>msoShapeRightTriangle</td>
<td>8</td>
</tr>
<tr>
<td>msoShapeRoundedRectangle</td>
<td>5</td>
</tr>
<tr>
<td>msoShapeRoundedRectangularCallout</td>
<td>106</td>
</tr>
<tr>
<td>msoShapeSmileyFace</td>
<td>17</td>
</tr>
<tr>
<td>msoShapeStripedRightArrow</td>
<td>49</td>
</tr>
<tr>
<td>msoShapeSun</td>
<td>23</td>
</tr>
<tr>
<td>msoShapeTrapezoid</td>
<td>3</td>
</tr>
<tr>
<td>msoShapeUpArrow</td>
<td>35</td>
</tr>
<tr>
<td>msoShapeUpArrowCallout</td>
<td>55</td>
</tr>
<tr>
<td>msoShapeUpDownArrow</td>
<td>38</td>
</tr>
<tr>
<td>msoShapeUpDownArrowCallout</td>
<td>58</td>
</tr>
<tr>
<td>msoShapeUpRibbon</td>
<td>97</td>
</tr>
<tr>
<td>msoShapeUTurnArrow</td>
<td>42</td>
</tr>
<tr>
<td>msoShapeVerticalScroll</td>
<td>101</td>
</tr>
<tr>
<td>msoShapeWave</td>
<td>103</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MsoBalloonButtonType</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoBalloonButtonAbort</td>
<td>-8</td>
</tr>
<tr>
<td>msoBalloonButtonBack</td>
<td>-5</td>
</tr>
<tr>
<td>msoBalloonButtonCancel</td>
<td>-2</td>
</tr>
<tr>
<td>msoBalloonButtonClose</td>
<td>-12</td>
</tr>
<tr>
<td>msoBalloonButtonIgnore</td>
<td>-9</td>
</tr>
<tr>
<td>msoBalloonButtonNext</td>
<td>-6</td>
</tr>
<tr>
<td>msoBalloonButtonNo</td>
<td>-4</td>
</tr>
<tr>
<td>msoBalloonButtonNull</td>
<td>0</td>
</tr>
<tr>
<td>msoBalloonButtonOK</td>
<td>-1</td>
</tr>
</tbody>
</table>
msoBalloonButtonOptions -14
msoBalloonButtonRetry -7
msoBalloonButtonSearch -10
msoBalloonButtonSnooze -11
msoBalloonButtonTips -13
msoBalloonButtonYes -3
msoBalloonButtonYesToAll -15

### MsoBalloonErrorType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoBalloonErrorBadCharacter</td>
<td>8</td>
</tr>
<tr>
<td>msoBalloonErrorBadPictureRef</td>
<td>4</td>
</tr>
<tr>
<td>msoBalloonErrorBadReference</td>
<td>5</td>
</tr>
<tr>
<td>msoBalloonErrorButtonlessModal</td>
<td>6</td>
</tr>
<tr>
<td>msoBalloonErrorButtonModeless</td>
<td>7</td>
</tr>
<tr>
<td>msoBalloonErrorCharNotTopmostForModal</td>
<td>10</td>
</tr>
<tr>
<td>msoBalloonErrorCOMFailure</td>
<td>9</td>
</tr>
<tr>
<td>msoBalloonErrorNone</td>
<td>0</td>
</tr>
<tr>
<td>msoBalloonErrorOther</td>
<td>1</td>
</tr>
<tr>
<td>msoBalloonErrorOutOfMemory</td>
<td>3</td>
</tr>
<tr>
<td>msoBalloonErrorTooBig</td>
<td>2</td>
</tr>
<tr>
<td>msoBalloonErrorTooManyControls</td>
<td>11</td>
</tr>
</tbody>
</table>

### MsoBalloonType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoBalloonTypeBullets</td>
<td>1</td>
</tr>
<tr>
<td>msoBalloonTypeButtons</td>
<td>0</td>
</tr>
<tr>
<td>msoBalloonTypeNumbers</td>
<td>2</td>
</tr>
</tbody>
</table>

### MsoBarPosition

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
</table>
msobarBottom  3
msobarFloating  4
msobarLeft  0
msobarMenuBar  6
msobarPopup  5
msobarRight  2
msobarTop  1

**MsoBarProtection**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msobarNoChangeDock</td>
<td>16</td>
</tr>
<tr>
<td>msobarNoChangeVisible</td>
<td>8</td>
</tr>
<tr>
<td>msobarNoCustomize</td>
<td>1</td>
</tr>
<tr>
<td>msobarNoHorizontalDock</td>
<td>64</td>
</tr>
<tr>
<td>msobarNoMove</td>
<td>4</td>
</tr>
<tr>
<td>msobarNoProtection</td>
<td>0</td>
</tr>
<tr>
<td>msobarNoResize</td>
<td>2</td>
</tr>
<tr>
<td>msobarNoVerticalDock</td>
<td>32</td>
</tr>
</tbody>
</table>

**MsoBarRow**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msobarRowFirst</td>
<td>0</td>
</tr>
<tr>
<td>msobarRowLast</td>
<td>-1</td>
</tr>
</tbody>
</table>

**MsoBarType**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msobarTypeMenuBar</td>
<td>1</td>
</tr>
<tr>
<td>msobarTypeNormal</td>
<td>0</td>
</tr>
<tr>
<td>msobarTypePopup</td>
<td>2</td>
</tr>
</tbody>
</table>

**MsoBlackWhiteMode**
### Constant Value

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoBlackWhiteAutomatic</td>
<td>1</td>
</tr>
<tr>
<td>msoBlackWhiteBlack</td>
<td>8</td>
</tr>
<tr>
<td>msoBlackWhiteBlackTextAndLine</td>
<td>6</td>
</tr>
<tr>
<td>msoBlackWhiteDontShow</td>
<td>10</td>
</tr>
<tr>
<td>msoBlackWhiteGrayOutline</td>
<td>5</td>
</tr>
<tr>
<td>msoBlackWhiteGrayScale</td>
<td>2</td>
</tr>
<tr>
<td>msoBlackWhiteHighContrast</td>
<td>7</td>
</tr>
<tr>
<td>msoBlackWhiteInverseGrayScale</td>
<td>4</td>
</tr>
<tr>
<td>msoBlackWhiteLightGrayScale</td>
<td>3</td>
</tr>
<tr>
<td>msoBlackWhiteMixed</td>
<td>-2</td>
</tr>
<tr>
<td>msoBlackWhiteWhite</td>
<td>9</td>
</tr>
</tbody>
</table>

### MsoButtonSetType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoButtonSetAbortRetryIgnore</td>
<td>10</td>
</tr>
<tr>
<td>msoButtonSetBackClose</td>
<td>6</td>
</tr>
<tr>
<td>msoButtonSetBackNextClose</td>
<td>8</td>
</tr>
<tr>
<td>msoButtonSetBackNextSnooze</td>
<td>12</td>
</tr>
<tr>
<td>msoButtonSetCancel</td>
<td>2</td>
</tr>
<tr>
<td>msoButtonSetNextClose</td>
<td>7</td>
</tr>
<tr>
<td>msoButtonSetNone</td>
<td>0</td>
</tr>
<tr>
<td>msoButtonSetOK</td>
<td>1</td>
</tr>
<tr>
<td>msoButtonSetOkCancel</td>
<td>3</td>
</tr>
<tr>
<td>msoButtonSetRetryCancel</td>
<td>9</td>
</tr>
<tr>
<td>msoButtonSetSearchClose</td>
<td>11</td>
</tr>
<tr>
<td>msoButtonSetTipsOptionsClose</td>
<td>13</td>
</tr>
<tr>
<td>msoButtonSetYesAllNoCancel</td>
<td>14</td>
</tr>
<tr>
<td>msoButtonSetYesNo</td>
<td>4</td>
</tr>
<tr>
<td>msoButtonSetYesNoCancel</td>
<td>5</td>
</tr>
</tbody>
</table>

### MsoButtonState
<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoButtonDown</td>
<td>-1</td>
</tr>
<tr>
<td>msoButtonMixed</td>
<td>2</td>
</tr>
<tr>
<td>msoButtonUp</td>
<td>0</td>
</tr>
</tbody>
</table>

### MsoButtonStyle

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoButtonAutomatic</td>
<td>0</td>
</tr>
<tr>
<td>msoButtonCaption</td>
<td>2</td>
</tr>
<tr>
<td>msoButtonIcon</td>
<td>1</td>
</tr>
<tr>
<td>msoButtonIconAndCaption</td>
<td>3</td>
</tr>
<tr>
<td>msoButtonIconAndCaptionBelow</td>
<td>11</td>
</tr>
<tr>
<td>msoButtonIconAndWrapCaption</td>
<td>7</td>
</tr>
<tr>
<td>msoButtonIconAndWrapCaptionBelow</td>
<td>15</td>
</tr>
<tr>
<td>msoButtonWrapCaption</td>
<td>14</td>
</tr>
</tbody>
</table>

### MsoCalloutAngleType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoCalloutAngle30</td>
<td>2</td>
</tr>
<tr>
<td>msoCalloutAngle45</td>
<td>3</td>
</tr>
<tr>
<td>msoCalloutAngle60</td>
<td>4</td>
</tr>
<tr>
<td>msoCalloutAngle90</td>
<td>5</td>
</tr>
<tr>
<td>msoCalloutAngleAutomatic</td>
<td>1</td>
</tr>
<tr>
<td>msoCalloutAngleMixed</td>
<td>-2</td>
</tr>
</tbody>
</table>

### MsoCalloutDropType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoCalloutDropBottom</td>
<td>4</td>
</tr>
<tr>
<td>msoCalloutDropCenter</td>
<td>3</td>
</tr>
<tr>
<td>msoCalloutDropCustom</td>
<td>1</td>
</tr>
<tr>
<td>msoCalloutDropMixed</td>
<td>-2</td>
</tr>
</tbody>
</table>
### MsoCalloutType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoCalloutFour</td>
<td>4</td>
</tr>
<tr>
<td>msoCalloutMixed</td>
<td>-2</td>
</tr>
<tr>
<td>msoCalloutOne</td>
<td>1</td>
</tr>
<tr>
<td>msoCalloutThree</td>
<td>3</td>
</tr>
<tr>
<td>msoCalloutTwo</td>
<td>2</td>
</tr>
</tbody>
</table>

### MsoCharacterSet

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoCharacterSetArabic</td>
<td>1</td>
</tr>
<tr>
<td>msoCharacterSetCyrillic</td>
<td>2</td>
</tr>
<tr>
<td>msoCharacterSetEnglishWesternEuropeanOtherLatinScript</td>
<td>3</td>
</tr>
<tr>
<td>msoCharacterSetGreek</td>
<td>4</td>
</tr>
<tr>
<td>msoCharacterSetHebrew</td>
<td>5</td>
</tr>
<tr>
<td>msoCharacterSetJapanese</td>
<td>6</td>
</tr>
<tr>
<td>msoCharacterSetKorean</td>
<td>7</td>
</tr>
<tr>
<td>msoCharacterSetMultilingualUnicode</td>
<td>8</td>
</tr>
<tr>
<td>msoCharacterSetSimplifiedChinese</td>
<td>9</td>
</tr>
<tr>
<td>msoCharacterSetThai</td>
<td>10</td>
</tr>
<tr>
<td>msoCharacterSetTraditionalChinese</td>
<td>11</td>
</tr>
<tr>
<td>msoCharacterSetVietnamese</td>
<td>12</td>
</tr>
</tbody>
</table>

### MsoColorType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoColorTypeCMS</td>
<td>4</td>
</tr>
<tr>
<td>msoColorTypeCMYK</td>
<td>3</td>
</tr>
<tr>
<td>msoColorTypeInk</td>
<td>5</td>
</tr>
<tr>
<td>msoColorTypeMixed</td>
<td>-2</td>
</tr>
</tbody>
</table>
msoColorTypeRGB 1
msoColorTypeScheme 2

**MsoComboStyle**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoComboLabel</td>
<td>1</td>
</tr>
<tr>
<td>msoComboNormal</td>
<td>0</td>
</tr>
</tbody>
</table>

**MsoCommandBarButtonHyperlinkType**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoCommandBarButtonHyperlinkInsertPicture</td>
<td>2</td>
</tr>
<tr>
<td>msoCommandBarButtonHyperlinkNone</td>
<td>0</td>
</tr>
<tr>
<td>msoCommandBarButtonHyperlinkOpen</td>
<td>1</td>
</tr>
</tbody>
</table>

**MsoCondition**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoConditionAnyNumberBetween</td>
<td>34</td>
</tr>
<tr>
<td>msoConditionAnytime</td>
<td>25</td>
</tr>
<tr>
<td>msoConditionAnytimeBetween</td>
<td>26</td>
</tr>
<tr>
<td>msoConditionAtLeast</td>
<td>36</td>
</tr>
<tr>
<td>msoConditionAtMost</td>
<td>35</td>
</tr>
<tr>
<td>msoConditionBeginsWith</td>
<td>11</td>
</tr>
<tr>
<td>msoConditionDoesNotEqual</td>
<td>33</td>
</tr>
<tr>
<td>msoConditionEndsWith</td>
<td>12</td>
</tr>
<tr>
<td>msoConditionEquals</td>
<td>32</td>
</tr>
<tr>
<td>msoConditionEqualsCompleted</td>
<td>66</td>
</tr>
<tr>
<td>msoConditionEqualsDeferred</td>
<td>68</td>
</tr>
<tr>
<td>msoConditionEqualsHigh</td>
<td>60</td>
</tr>
<tr>
<td>msoConditionEqualsInProgress</td>
<td>65</td>
</tr>
<tr>
<td>msoConditionEqualsLow</td>
<td>58</td>
</tr>
<tr>
<td>msoConditionEqualsNormal</td>
<td>59</td>
</tr>
<tr>
<td>Condition</td>
<td>Value</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>msoConditionEqualsNot Started</td>
<td>64</td>
</tr>
<tr>
<td>msoConditionEqualsWaitingForSomeoneElse</td>
<td>67</td>
</tr>
<tr>
<td>msoConditionFileTypeAllFiles</td>
<td>1</td>
</tr>
<tr>
<td>msoConditionFileTypeBinders</td>
<td>6</td>
</tr>
<tr>
<td>msoConditionFileTypeCalendarItem</td>
<td>45</td>
</tr>
<tr>
<td>msoConditionFileTypeContactItem</td>
<td>46</td>
</tr>
<tr>
<td>msoConditionFileTypeDatabases</td>
<td>7</td>
</tr>
<tr>
<td>msoConditionFileTypeDataConnectionFiles</td>
<td>51</td>
</tr>
<tr>
<td>msoConditionFileTypeDesignerFiles</td>
<td>56</td>
</tr>
<tr>
<td>msoConditionFileTypeDocumentImagingFiles</td>
<td>54</td>
</tr>
<tr>
<td>msoConditionFileTypeExcelWorkbooks</td>
<td>4</td>
</tr>
<tr>
<td>msoConditionFileTypeJournalItem</td>
<td>48</td>
</tr>
<tr>
<td>msoConditionFileTypeMailItem</td>
<td>44</td>
</tr>
<tr>
<td>msoConditionFileTypeNoteItem</td>
<td>47</td>
</tr>
<tr>
<td>msoConditionFileTypeOfficeFiles</td>
<td>2</td>
</tr>
<tr>
<td>msoConditionFileTypeOutlookItems</td>
<td>43</td>
</tr>
<tr>
<td>msoConditionFileTypePhotoDrawFiles</td>
<td>50</td>
</tr>
<tr>
<td>msoConditionFileTypePowerPointPresentations</td>
<td>5</td>
</tr>
<tr>
<td>msoConditionFileTypeProjectFiles</td>
<td>53</td>
</tr>
<tr>
<td>msoConditionFileTypePublisherFiles</td>
<td>52</td>
</tr>
<tr>
<td>msoConditionFileTypeTaskItem</td>
<td>49</td>
</tr>
<tr>
<td>msoConditionFileTypeTemplates</td>
<td>8</td>
</tr>
<tr>
<td>msoConditionFileTypeVisioFiles</td>
<td>55</td>
</tr>
<tr>
<td>msoConditionFileTypeWebPages</td>
<td>57</td>
</tr>
<tr>
<td>msoConditionFileTypeWordDocuments</td>
<td>3</td>
</tr>
<tr>
<td>msoConditionFreeText</td>
<td>42</td>
</tr>
<tr>
<td>msoConditionIncludes</td>
<td>9</td>
</tr>
<tr>
<td>msoConditionIncludesFormsOf</td>
<td>41</td>
</tr>
<tr>
<td>msoConditionIncludesNearEachOther</td>
<td>13</td>
</tr>
<tr>
<td>msoConditionIncludesPhrase</td>
<td>10</td>
</tr>
<tr>
<td>msoConditionInTheLast</td>
<td>31</td>
</tr>
<tr>
<td>msoConditionInTheNext</td>
<td>30</td>
</tr>
<tr>
<td>msoConditionIsExactly</td>
<td>14</td>
</tr>
</tbody>
</table>
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
msoConditionNotEqualToWaitingForSomeoneElse 72
msoConditionOn            27
msoConditionOnOrAfter     28
msoConditionOnOrBefore    29
msoConditionThisMonth     23
msoConditionThisWeek      20
msoConditionToday         17
msoConditionTomorrow      18
msoConditionYesterday     16

<table>
<thead>
<tr>
<th>MsoConnector</th>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoConnectorAnd</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>msoConnectorOr</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

<p>| MsoConnectorType |</p>
<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoConnectorCurve</td>
<td>3</td>
</tr>
<tr>
<td>msoConnectorElbow</td>
<td>2</td>
</tr>
<tr>
<td>msoConnectorStraight</td>
<td>1</td>
</tr>
<tr>
<td>msoConnectorTypeMixed</td>
<td>-2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MsoControlOLEUsage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
</tr>
<tr>
<td>msoControlOLEUsageBoth</td>
</tr>
<tr>
<td>msoControlOLEUsageClient</td>
</tr>
<tr>
<td>msoControlOLEUsageNeither</td>
</tr>
<tr>
<td>msoControlOLEUsageServer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MsoControlType</th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
</tr>
<tr>
<td>msoControlActiveX</td>
</tr>
<tr>
<td>msoControlAutoCompleteCombo</td>
</tr>
<tr>
<td>msoControlButton</td>
</tr>
<tr>
<td>msoControlButtonDropdown</td>
</tr>
<tr>
<td>msoControlButtonPopup</td>
</tr>
<tr>
<td>msoControlComboBox</td>
</tr>
<tr>
<td>msoControlCustom</td>
</tr>
<tr>
<td>msoControlDropdown</td>
</tr>
<tr>
<td>msoControlEdit</td>
</tr>
<tr>
<td>msoControlExpandingGrid</td>
</tr>
<tr>
<td>msoControlGauge</td>
</tr>
<tr>
<td>msoControlGenericDropdown</td>
</tr>
<tr>
<td>msoControlGraphicCombo</td>
</tr>
<tr>
<td>msoControlGraphicDropdown</td>
</tr>
<tr>
<td>msoControlGraphicPopup</td>
</tr>
<tr>
<td>msoControlGrid</td>
</tr>
<tr>
<td>msoControlLabel</td>
</tr>
<tr>
<td>MsoDiagramNodeType</td>
</tr>
<tr>
<td>-------------------</td>
</tr>
<tr>
<td>msoDiagramAssistant</td>
</tr>
<tr>
<td>msoDiagramNode</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MsoDiagramType</th>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoDiagramCycle</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>msoDiagramMixed</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>msoDiagramOrgChart</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>msoDiagramPyramid</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>msoDiagramRadial</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>msoDiagramTarget</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>msoDiagramVenn</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MsoDistributeCmd</th>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoDistributeHorizontally</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>msoDistributeVertically</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
### MsoDocProperties

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoPropertyTypeBoolean</td>
<td>2</td>
</tr>
<tr>
<td>msoPropertyTypeDate</td>
<td>3</td>
</tr>
<tr>
<td>msoPropertyTypeFloat</td>
<td>5</td>
</tr>
<tr>
<td>msoPropertyTypeNumber</td>
<td>1</td>
</tr>
<tr>
<td>msoPropertyTypeString</td>
<td>4</td>
</tr>
</tbody>
</table>

### MsoEditingType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoEditingAuto</td>
<td>0</td>
</tr>
<tr>
<td>msoEditingCorner</td>
<td>1</td>
</tr>
<tr>
<td>msoEditingSmooth</td>
<td>2</td>
</tr>
<tr>
<td>msoEditingSymmetric</td>
<td>3</td>
</tr>
</tbody>
</table>

### MsoEncoding

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoEncodingArabic</td>
<td>1256</td>
</tr>
<tr>
<td>msoEncodingArabicASMO</td>
<td>708</td>
</tr>
<tr>
<td>msoEncodingArabicAutoDetect</td>
<td>51256</td>
</tr>
<tr>
<td>msoEncodingArabicTransparentASMO</td>
<td>720</td>
</tr>
<tr>
<td>msoEncodingAutoDetect</td>
<td>50001</td>
</tr>
<tr>
<td>msoEncodingBaltic</td>
<td>1257</td>
</tr>
<tr>
<td>msoEncodingCentralEuropean</td>
<td>1250</td>
</tr>
<tr>
<td>msoEncodingCyrillic</td>
<td>1251</td>
</tr>
<tr>
<td>msoEncodingCyrillicAutoDetect</td>
<td>51251</td>
</tr>
<tr>
<td>msoEncodingEBCDICArabic</td>
<td>20420</td>
</tr>
<tr>
<td>msoEncodingEBCDICDenmarkNorway</td>
<td>20277</td>
</tr>
<tr>
<td>msoEncodingEBCDICFinlandSweden</td>
<td>20278</td>
</tr>
<tr>
<td>msoEncodingEBCDICFrance</td>
<td>20297</td>
</tr>
<tr>
<td>msoEncodingEBCDICGermany</td>
<td>20273</td>
</tr>
</tbody>
</table>
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

Constant Value
msoMethodGet 0
msoMethodPost 1

MsoExtrusionColorType

Constant Value
msoExtrusionColorAutomatic 1
msoExtrusionColorCustom 2
msoExtrusionColorTypeMixed -2

**MsoFarEastLineBreakLanguageID**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MsoFarEastLineBreakLanguageJapanese</td>
<td>1041</td>
</tr>
<tr>
<td>MsoFarEastLineBreakLanguageKorean</td>
<td>1042</td>
</tr>
<tr>
<td>MsoFarEastLineBreakLanguageSimplifiedChinese</td>
<td>2052</td>
</tr>
<tr>
<td>MsoFarEastLineBreakLanguageTraditionalChinese</td>
<td>1028</td>
</tr>
</tbody>
</table>

**MsoFeatureInstall**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoFeatureInstallNone</td>
<td>0</td>
</tr>
<tr>
<td>msoFeatureInstallOnDemand</td>
<td>1</td>
</tr>
<tr>
<td>msoFeatureInstallOnDemandWithUI</td>
<td>2</td>
</tr>
</tbody>
</table>

**MsoFileDialogType**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoFileDialogFilePicker</td>
<td>3</td>
</tr>
<tr>
<td>msoFileDialogFolderPicker</td>
<td>4</td>
</tr>
<tr>
<td>msoFileDialogOpen</td>
<td>1</td>
</tr>
<tr>
<td>msoFileDialogSaveAs</td>
<td>2</td>
</tr>
</tbody>
</table>

**MsoFileDialogView**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoFileDialogViewDetails</td>
<td>2</td>
</tr>
<tr>
<td>msoFileDialogViewLargeIcons</td>
<td>6</td>
</tr>
<tr>
<td>msoFileDialogViewList</td>
<td>1</td>
</tr>
<tr>
<td>msoFileDialogViewPreview</td>
<td>4</td>
</tr>
<tr>
<td>msoFileDialogViewProperties</td>
<td>3</td>
</tr>
<tr>
<td>msoFileDialogViewSmallIcons</td>
<td>7</td>
</tr>
<tr>
<td>Constant</td>
<td>Value</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------</td>
</tr>
<tr>
<td>msoListbyName</td>
<td>1</td>
</tr>
<tr>
<td>msoListbyTitle</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoOptionsAdd</td>
<td>2</td>
</tr>
<tr>
<td>msoOptionsNew</td>
<td>1</td>
</tr>
<tr>
<td>msoOptionsWithin</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoFileFindSortbyAuthor</td>
<td>1</td>
</tr>
<tr>
<td>msoFileFindSortbyDateCreated</td>
<td>2</td>
</tr>
<tr>
<td>msoFileFindSortbyDateSaved</td>
<td>4</td>
</tr>
<tr>
<td>msoFileFindSortbyFileName</td>
<td>5</td>
</tr>
<tr>
<td>msoFileFindSortbyLastSavedBy</td>
<td>3</td>
</tr>
<tr>
<td>msoFileFindSortbySize</td>
<td>6</td>
</tr>
<tr>
<td>msoFileFindSortbyTitle</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoViewFileInfo</td>
<td>1</td>
</tr>
<tr>
<td>msoViewPreview</td>
<td>2</td>
</tr>
<tr>
<td>msoViewSummaryInfo</td>
<td>3</td>
</tr>
</tbody>
</table>
## MsoFileNewAction

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoCreateNewFile</td>
<td>1</td>
</tr>
<tr>
<td>msoEditFile</td>
<td>0</td>
</tr>
<tr>
<td>msoOpenFile</td>
<td>2</td>
</tr>
</tbody>
</table>

## MsoFileNewSection

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoBottomSection</td>
<td>4</td>
</tr>
<tr>
<td>msoNew</td>
<td>1</td>
</tr>
<tr>
<td>msoNewfromExistingFile</td>
<td>2</td>
</tr>
<tr>
<td>msoNewfromTemplate</td>
<td>3</td>
</tr>
<tr>
<td>msoOpenDocument</td>
<td>0</td>
</tr>
</tbody>
</table>

## MsoFileType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoFileTypeAllFiles</td>
<td>1</td>
</tr>
<tr>
<td>msoFileTypeBinders</td>
<td>6</td>
</tr>
<tr>
<td>msoFileTypeCalendarItem</td>
<td>11</td>
</tr>
<tr>
<td>msoFileTypeContactItem</td>
<td>12</td>
</tr>
<tr>
<td>msoFileTypeDatabases</td>
<td>7</td>
</tr>
<tr>
<td>msoFileTypeDataConnectionFiles</td>
<td>17</td>
</tr>
<tr>
<td>msoFileTypeDesignerFiles</td>
<td>22</td>
</tr>
<tr>
<td>msoFileTypeDocumentImagingFiles</td>
<td>20</td>
</tr>
<tr>
<td>msoFileTypeExcelWorkbooks</td>
<td>4</td>
</tr>
<tr>
<td>msoFileTypeJournalItem</td>
<td>14</td>
</tr>
<tr>
<td>msoFileTypeMailItem</td>
<td>10</td>
</tr>
<tr>
<td>msoFileTypeNoteItem</td>
<td>13</td>
</tr>
<tr>
<td>msoFileTypeOfficeFiles</td>
<td>2</td>
</tr>
<tr>
<td>msoFileTypeOutlookItems</td>
<td>9</td>
</tr>
<tr>
<td>msoFileTypePhotoDrawFiles</td>
<td>16</td>
</tr>
</tbody>
</table>
msoFileTypePowerPointPresentations 5
msoFileTypeProjectFiles 19
msoFileTypePublisherFiles 18
msoFileTypeTaskItem 15
msoFileTypeTemplates 8
msoFileTypeVisioFiles 21
msoFileTypeWebPages 23
msoFileTypeWordDocuments 3

<table>
<thead>
<tr>
<th>MsoFillType</th>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoFillBackgroundColor</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>msoFillGradient</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>msoFillMixed</td>
<td>-2</td>
<td></td>
</tr>
<tr>
<td>msoFillPatterned</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>msoFillPicture</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>msoFillSolid</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>msoFillTextured</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MsoFilterComparison</th>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoFilterComparisonContains</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>msoFilterComparisonEqual</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>msoFilterComparisonGreaterThan</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>msoFilterComparisonGreaterThanEqual</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>msoFilterComparisonIsBlank</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>msoFilterComparisonIsNotBlank</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>msoFilterComparisonLessThan</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>msoFilterComparisonLessThanEqual</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>msoFilterComparisonNotContains</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>msoFilterComparisonNotEqual</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
### MsoFilterConjunction

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoFilterConjunctionAnd</td>
<td>0</td>
</tr>
<tr>
<td>msoFilterConjunctionOr</td>
<td>1</td>
</tr>
</tbody>
</table>

### MsoFlipCmd

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoFlipHorizontal</td>
<td>0</td>
</tr>
<tr>
<td>msoFlipVertical</td>
<td>1</td>
</tr>
</tbody>
</table>

### MsoGradientColorType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoGradientColorMixed</td>
<td>-2</td>
</tr>
<tr>
<td>msoGradientOneColor</td>
<td>1</td>
</tr>
<tr>
<td>msoGradientPresetColors</td>
<td>3</td>
</tr>
<tr>
<td>msoGradientTwoColors</td>
<td>2</td>
</tr>
</tbody>
</table>

### MsoGradientStyle

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoGradientDiagonalDown</td>
<td>4</td>
</tr>
<tr>
<td>msoGradientDiagonalUp</td>
<td>3</td>
</tr>
<tr>
<td>msoGradientFromCenter</td>
<td>7</td>
</tr>
<tr>
<td>msoGradientFromCorner</td>
<td>5</td>
</tr>
<tr>
<td>msoGradientFromTitle</td>
<td>6</td>
</tr>
<tr>
<td>msoGradientHorizontal</td>
<td>1</td>
</tr>
<tr>
<td>msoGradientMixed</td>
<td>-2</td>
</tr>
<tr>
<td>msoGradientVertical</td>
<td>2</td>
</tr>
</tbody>
</table>

### MsoHorizontalAnchor
### Constant  Value

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoAnchorCenter</td>
<td>2</td>
</tr>
<tr>
<td>msoAnchorNone</td>
<td>1</td>
</tr>
<tr>
<td>msoHorizontalAnchorMixed</td>
<td>-2</td>
</tr>
</tbody>
</table>

### MsoHTMLProjectOpen

### Constant  Value

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoHTMLProjectOpenSourceView</td>
<td>1</td>
</tr>
<tr>
<td>msoHTMLProjectOpenTextView</td>
<td>2</td>
</tr>
</tbody>
</table>

### MsoHTMLProjectState

### Constant  Value

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoHTMLProjectStateDocumentLocked</td>
<td>1</td>
</tr>
<tr>
<td>msoHTMLProjectStateDocumentProjectUnlocked</td>
<td>3</td>
</tr>
<tr>
<td>msoHTMLProjectStateProjectLocked</td>
<td>2</td>
</tr>
</tbody>
</table>

### MsoHyperlinkType

### Constant  Value

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoHyperlinkInlineShape</td>
<td>2</td>
</tr>
<tr>
<td>msoHyperlinkRange</td>
<td>0</td>
</tr>
<tr>
<td>msoHyperlinkShape</td>
<td>1</td>
</tr>
</tbody>
</table>

### MsoIconType

### Constant  Value

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoIconAlert</td>
<td>2</td>
</tr>
<tr>
<td>msoIconAlertCritical</td>
<td>7</td>
</tr>
<tr>
<td>msoIconAlertInfo</td>
<td>4</td>
</tr>
<tr>
<td>msoIconAlertQuery</td>
<td>6</td>
</tr>
<tr>
<td>msoIconAlertWarning</td>
<td>5</td>
</tr>
<tr>
<td>msoIconNone</td>
<td>0</td>
</tr>
<tr>
<td>Constant</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>msolanguageIDAfrikaans</td>
<td>1078</td>
</tr>
<tr>
<td>msolanguageIDAlbanian</td>
<td>1052</td>
</tr>
<tr>
<td>msolanguageIDAmharic</td>
<td>1118</td>
</tr>
<tr>
<td>msolanguageIDArabic</td>
<td>1025</td>
</tr>
<tr>
<td>msolanguageIDArabicAlgeria</td>
<td>5121</td>
</tr>
<tr>
<td>msolanguageIDArabicBahrain</td>
<td>15361</td>
</tr>
<tr>
<td>msolanguageIDArabicEgypt</td>
<td>3073</td>
</tr>
<tr>
<td>msolanguageIDArabicIraq</td>
<td>2049</td>
</tr>
<tr>
<td>msolanguageIDArabicJordan</td>
<td>11265</td>
</tr>
<tr>
<td>msolanguageIDArabicKuwait</td>
<td>13313</td>
</tr>
<tr>
<td>msolanguageIDArabicLebanon</td>
<td>12289</td>
</tr>
<tr>
<td>msolanguageIDArabicLibya</td>
<td>4097</td>
</tr>
<tr>
<td>msolanguageIDArabicMorocco</td>
<td>6145</td>
</tr>
<tr>
<td>msolanguageIDArabicOman</td>
<td>8193</td>
</tr>
<tr>
<td>msolanguageIDArabicQatar</td>
<td>16385</td>
</tr>
<tr>
<td>msolanguageIDArabicSyria</td>
<td>10241</td>
</tr>
<tr>
<td>msolanguageIDArabicTunisia</td>
<td>7169</td>
</tr>
<tr>
<td>msolanguageIDArabicUAE</td>
<td>14337</td>
</tr>
<tr>
<td>msolanguageIDArabicYemen</td>
<td>9217</td>
</tr>
<tr>
<td>msolanguageIDArmenian</td>
<td>1067</td>
</tr>
<tr>
<td>msolanguageIDAssamese</td>
<td>1101</td>
</tr>
<tr>
<td>msolanguageIDAzerbaiCyrillic</td>
<td>2092</td>
</tr>
<tr>
<td>msolanguageIDAzerbaiLatin</td>
<td>1068</td>
</tr>
<tr>
<td>msolanguageIDBasque</td>
<td>1069</td>
</tr>
<tr>
<td>msolanguageIDBelgianDutch</td>
<td>2067</td>
</tr>
<tr>
<td>msolanguageIDBelgianFrench</td>
<td>2060</td>
</tr>
<tr>
<td>msolanguageIDBengali</td>
<td>1093</td>
</tr>
<tr>
<td>msolanguageIDBosnian</td>
<td>4122</td>
</tr>
<tr>
<td>Language ID</td>
<td>Language Name</td>
</tr>
<tr>
<td>------------</td>
<td>--------------------------------</td>
</tr>
<tr>
<td>msoLanguageIDEstonian</td>
<td>Estonian</td>
</tr>
<tr>
<td>msoLanguageIDFaeroese</td>
<td>Faeroese</td>
</tr>
<tr>
<td>msoLanguageIDFarsi</td>
<td>Farsi</td>
</tr>
<tr>
<td>msoLanguageIDFilipino</td>
<td>Filipino</td>
</tr>
<tr>
<td>msoLanguageIDFinnish</td>
<td>Finnish</td>
</tr>
<tr>
<td>msoLanguageIDFrench</td>
<td>French</td>
</tr>
<tr>
<td>msoLanguageIDFrenchCameroon</td>
<td>French Cameroon</td>
</tr>
<tr>
<td>msoLanguageIDFrenchCanadian</td>
<td>French Canadian</td>
</tr>
<tr>
<td>msoLanguageIDFrenchCotedIvoire</td>
<td>French CotedIvoire</td>
</tr>
<tr>
<td>msoLanguageIDFrenchHaiti</td>
<td>French Haiti</td>
</tr>
<tr>
<td>msoLanguageIDFrenchLuxembourg</td>
<td>French Luxembourg</td>
</tr>
<tr>
<td>msoLanguageIDFrenchMali</td>
<td>French Mali</td>
</tr>
<tr>
<td>msoLanguageIDFrenchMonaco</td>
<td>French Monaco</td>
</tr>
<tr>
<td>msoLanguageIDFrenchMorocco</td>
<td>French Morocco</td>
</tr>
<tr>
<td>msoLanguageIDFrenchReunion</td>
<td>French Reunion</td>
</tr>
<tr>
<td>msoLanguageIDFrenchSenegal</td>
<td>French Senegal</td>
</tr>
<tr>
<td>msoLanguageIDFrenchWestIndies</td>
<td>French WestIndies</td>
</tr>
<tr>
<td>msoLanguageIDFrenchZaire</td>
<td>French Zaire</td>
</tr>
<tr>
<td>msoLanguageIDFrisianNetherlands</td>
<td>Frisian Netherlands</td>
</tr>
<tr>
<td>msoLanguageIDFulfulde</td>
<td>Fulfulde</td>
</tr>
<tr>
<td>msoLanguageIDGaelicIreland</td>
<td>Gaelic Ireland</td>
</tr>
<tr>
<td>msoLanguageIDGaelicScotland</td>
<td>Gaelic Scotland</td>
</tr>
<tr>
<td>msoLanguageIDGalician</td>
<td>Galician</td>
</tr>
<tr>
<td>msoLanguageIDGeorgian</td>
<td>Georgian</td>
</tr>
<tr>
<td>msoLanguageIDGerman</td>
<td>German</td>
</tr>
<tr>
<td>msoLanguageIDGermanAustria</td>
<td>German Austria</td>
</tr>
<tr>
<td>msoLanguageIDGermanLiechtenstein</td>
<td>German Liechtenstein</td>
</tr>
<tr>
<td>msoLanguageIDGermanLuxembourg</td>
<td>German Luxembourg</td>
</tr>
<tr>
<td>msoLanguageIDGreek</td>
<td>Greek</td>
</tr>
<tr>
<td>msoLanguageIDGuarani</td>
<td>Guarani</td>
</tr>
<tr>
<td>msoLanguageIDGujarati</td>
<td>Gujarati</td>
</tr>
<tr>
<td>msoLanguageIDHausa</td>
<td>Hausa</td>
</tr>
<tr>
<td>msoLanguageIDHawaiian</td>
<td>Hawaiian</td>
</tr>
<tr>
<td>Language ID</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>msoLanguageIDSpanishArgentina</td>
<td>11274</td>
</tr>
<tr>
<td>msoLanguageIDSpanishBolivia</td>
<td>16394</td>
</tr>
<tr>
<td>msoLanguageIDSpanishChile</td>
<td>13322</td>
</tr>
<tr>
<td>msoLanguageIDSpanishColombia</td>
<td>9226</td>
</tr>
<tr>
<td>msoLanguageIDSpanishCostaRica</td>
<td>5130</td>
</tr>
<tr>
<td>msoLanguageIDSpanishDominicanRepublic</td>
<td>7178</td>
</tr>
<tr>
<td>msoLanguageIDSpanishEcuador</td>
<td>12298</td>
</tr>
<tr>
<td>msoLanguageIDSpanishElSalvador</td>
<td>17418</td>
</tr>
<tr>
<td>msoLanguageIDSpanishGuatemala</td>
<td>4106</td>
</tr>
<tr>
<td>msoLanguageIDSpanishHonduras</td>
<td>18442</td>
</tr>
<tr>
<td>msoLanguageIDSpanishModernSort</td>
<td>3082</td>
</tr>
<tr>
<td>msoLanguageIDSpanishNicaragua</td>
<td>19466</td>
</tr>
<tr>
<td>msoLanguageIDSpanishPanama</td>
<td>6154</td>
</tr>
<tr>
<td>msoLanguageIDSpanishParaguay</td>
<td>15370</td>
</tr>
<tr>
<td>msoLanguageIDSpanishPeru</td>
<td>10250</td>
</tr>
<tr>
<td>msoLanguageIDSpanishPuertoRico</td>
<td>20490</td>
</tr>
<tr>
<td>msoLanguageIDSpanishUruguay</td>
<td>14346</td>
</tr>
<tr>
<td>msoLanguageIDSpanishVenezuela</td>
<td>8202</td>
</tr>
<tr>
<td>msoLanguageIDSutu</td>
<td>1072</td>
</tr>
<tr>
<td>msoLanguageIDSwahili</td>
<td>1089</td>
</tr>
<tr>
<td>msoLanguageIDSwedish</td>
<td>1053</td>
</tr>
<tr>
<td>msoLanguageIDSwedishFinland</td>
<td>2077</td>
</tr>
<tr>
<td>msoLanguageIDSwissFrench</td>
<td>4108</td>
</tr>
<tr>
<td>msoLanguageIDSwissGerman</td>
<td>2055</td>
</tr>
<tr>
<td>msoLanguageIDSwissItalian</td>
<td>2064</td>
</tr>
<tr>
<td>msoLanguageIDSyriac</td>
<td>1114</td>
</tr>
<tr>
<td>msoLanguageIDTajik</td>
<td>1064</td>
</tr>
<tr>
<td>msoLanguageIDTajik</td>
<td>1119</td>
</tr>
<tr>
<td>msoLanguageIDTamazight</td>
<td>2143</td>
</tr>
<tr>
<td>msoLanguageIDTamazightLatin</td>
<td>1097</td>
</tr>
<tr>
<td>msoLanguageIDTatar</td>
<td>1092</td>
</tr>
<tr>
<td>msoLanguageIDTelugu</td>
<td>1098</td>
</tr>
<tr>
<td>msoLanguageIDThai</td>
<td>1054</td>
</tr>
<tr>
<td>msoLanguageID</td>
<td>Value</td>
</tr>
<tr>
<td>---------------</td>
<td>-------</td>
</tr>
<tr>
<td>Tibetan</td>
<td>1105</td>
</tr>
<tr>
<td>Tigrigna Eritrea</td>
<td>2163</td>
</tr>
<tr>
<td>Tigrigna Ethiopic</td>
<td>1139</td>
</tr>
<tr>
<td>Traditional Chinese</td>
<td>1028</td>
</tr>
<tr>
<td>Tsonga</td>
<td>1073</td>
</tr>
<tr>
<td>Tswana</td>
<td>1074</td>
</tr>
<tr>
<td>Turkish</td>
<td>1055</td>
</tr>
<tr>
<td>Turkmen</td>
<td>1090</td>
</tr>
<tr>
<td>Ukrainian</td>
<td>1058</td>
</tr>
<tr>
<td>Urdu</td>
<td>1056</td>
</tr>
<tr>
<td>Uzbek Cyrillic</td>
<td>2115</td>
</tr>
<tr>
<td>Uzbek Latin</td>
<td>1091</td>
</tr>
<tr>
<td>Venda</td>
<td>1075</td>
</tr>
<tr>
<td>Vietnamese</td>
<td>1066</td>
</tr>
<tr>
<td>Welsh</td>
<td>1106</td>
</tr>
<tr>
<td>Xhosa</td>
<td>1076</td>
</tr>
<tr>
<td>Yi</td>
<td>1144</td>
</tr>
<tr>
<td>Yiddish</td>
<td>1085</td>
</tr>
<tr>
<td>Yoruba</td>
<td>1130</td>
</tr>
<tr>
<td>Zulu</td>
<td>1077</td>
</tr>
</tbody>
</table>

**MsoLastModified**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoLastModifiedAnyTime</td>
<td>7</td>
</tr>
<tr>
<td>msoLastModifiedLastMonth</td>
<td>5</td>
</tr>
<tr>
<td>msoLastModifiedLastWeek</td>
<td>3</td>
</tr>
<tr>
<td>msoLastModifiedThisMonth</td>
<td>6</td>
</tr>
<tr>
<td>msoLastModifiedThisWeek</td>
<td>4</td>
</tr>
<tr>
<td>msoLastModifiedToday</td>
<td>2</td>
</tr>
<tr>
<td>msoLastModifiedYesterday</td>
<td>1</td>
</tr>
</tbody>
</table>

**MsoLineDashStyle**
<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoLineDash</td>
<td>4</td>
</tr>
<tr>
<td>msoLineDashDot</td>
<td>5</td>
</tr>
<tr>
<td>msoLineDashDotDot</td>
<td>6</td>
</tr>
<tr>
<td>msoLineDashStyleMixed</td>
<td>-2</td>
</tr>
<tr>
<td>msoLineLongDash</td>
<td>7</td>
</tr>
<tr>
<td>msoLineLongDashDot</td>
<td>8</td>
</tr>
<tr>
<td>msoLineRoundDot</td>
<td>3</td>
</tr>
<tr>
<td>msoLineSolid</td>
<td>1</td>
</tr>
<tr>
<td>msoLineSquareDot</td>
<td>2</td>
</tr>
</tbody>
</table>

### MsoLineStyle

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoLineSingle</td>
<td>1</td>
</tr>
<tr>
<td>msoLineStyleMixed</td>
<td>-2</td>
</tr>
<tr>
<td>msoLineThickBetweenThin</td>
<td>5</td>
</tr>
<tr>
<td>msoLineThickThin</td>
<td>4</td>
</tr>
<tr>
<td>msoLineThinThick</td>
<td>3</td>
</tr>
<tr>
<td>msoLineThinThin</td>
<td>2</td>
</tr>
</tbody>
</table>

### MsoMenuAnimation

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoMenuAnimationNone</td>
<td>0</td>
</tr>
<tr>
<td>msoMenuAnimationRandom</td>
<td>1</td>
</tr>
<tr>
<td>msoMenuAnimationSlide</td>
<td>3</td>
</tr>
<tr>
<td>msoMenuAnimationUnfold</td>
<td>2</td>
</tr>
</tbody>
</table>

### MsoMixedType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoIntegerMixed</td>
<td>32768</td>
</tr>
<tr>
<td>msoSingleMixed</td>
<td>-2147483648</td>
</tr>
</tbody>
</table>
### MsoModeType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoModeAutoDown</td>
<td>1</td>
</tr>
<tr>
<td>msoModeModal</td>
<td>0</td>
</tr>
<tr>
<td>msoModeModeless</td>
<td>2</td>
</tr>
</tbody>
</table>

### MsoMoveRow

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoMoveRowFirst</td>
<td>-4</td>
</tr>
<tr>
<td>msoMoveRowNbr</td>
<td>-1</td>
</tr>
<tr>
<td>msoMoveRowNext</td>
<td>-2</td>
</tr>
<tr>
<td>msoMoveRowPrev</td>
<td>-3</td>
</tr>
</tbody>
</table>

### MsoOLEMenuGroup

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoOLEMenuGroupContainer</td>
<td>2</td>
</tr>
<tr>
<td>msoOLEMenuGroupEdit</td>
<td>1</td>
</tr>
<tr>
<td>msoOLEMenuGroupFile</td>
<td>0</td>
</tr>
<tr>
<td>msoOLEMenuGroupHelp</td>
<td>5</td>
</tr>
<tr>
<td>msoOLEMenuGroupNone</td>
<td>-1</td>
</tr>
<tr>
<td>msoOLEMenuGroupObject</td>
<td>3</td>
</tr>
<tr>
<td>msoOLEMenuGroupWindow</td>
<td>4</td>
</tr>
</tbody>
</table>

### MsoOrgChartLayoutType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoOrgChartLayoutBothHanging</td>
<td>2</td>
</tr>
<tr>
<td>msoOrgChartLayoutLeftHanging</td>
<td>3</td>
</tr>
<tr>
<td>msoOrgChartLayoutMixed</td>
<td>-2</td>
</tr>
<tr>
<td>msoOrgChartLayoutRightHanging</td>
<td>4</td>
</tr>
<tr>
<td>msoOrgChartLayoutStandard</td>
<td>1</td>
</tr>
</tbody>
</table>
### MsoOrgChartOrientation

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoOrgChartOrientationMixed</td>
<td>-2</td>
</tr>
<tr>
<td>msoOrgChartOrientationVertical</td>
<td>1</td>
</tr>
</tbody>
</table>

### MsoOrientation

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoOrientationHorizontal</td>
<td>1</td>
</tr>
<tr>
<td>msoOrientationMixed</td>
<td>-2</td>
</tr>
<tr>
<td>msoOrientationVertical</td>
<td>2</td>
</tr>
</tbody>
</table>

### MsoPatternType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoPattern10Percent</td>
<td>2</td>
</tr>
<tr>
<td>msoPattern20Percent</td>
<td>3</td>
</tr>
<tr>
<td>msoPattern25Percent</td>
<td>4</td>
</tr>
<tr>
<td>msoPattern30Percent</td>
<td>5</td>
</tr>
<tr>
<td>msoPattern40Percent</td>
<td>6</td>
</tr>
<tr>
<td>msoPattern50Percent</td>
<td>7</td>
</tr>
<tr>
<td>msoPattern5Percent</td>
<td>1</td>
</tr>
<tr>
<td>msoPattern60Percent</td>
<td>8</td>
</tr>
<tr>
<td>msoPattern70Percent</td>
<td>9</td>
</tr>
<tr>
<td>msoPattern75Percent</td>
<td>10</td>
</tr>
<tr>
<td>msoPattern80Percent</td>
<td>11</td>
</tr>
<tr>
<td>msoPattern90Percent</td>
<td>12</td>
</tr>
<tr>
<td>msoPatternDarkDownwardDiagonal</td>
<td>15</td>
</tr>
<tr>
<td>msoPatternDarkHorizontal</td>
<td>13</td>
</tr>
<tr>
<td>msoPatternDarkUpwardDiagonal</td>
<td>16</td>
</tr>
<tr>
<td>msoPatternDarkVertical</td>
<td>14</td>
</tr>
<tr>
<td>msoPatternDashedDownwardDiagonal</td>
<td>28</td>
</tr>
<tr>
<td>msoPatternDashedHorizontal</td>
<td>32</td>
</tr>
</tbody>
</table>
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
<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoPermissionChange</td>
<td>15</td>
</tr>
<tr>
<td>msoPermissionEdit</td>
<td>2</td>
</tr>
<tr>
<td>msoPermissionExtract</td>
<td>8</td>
</tr>
<tr>
<td>msoPermissionFullControl</td>
<td>64</td>
</tr>
<tr>
<td>msoPermissionObjModel</td>
<td>32</td>
</tr>
<tr>
<td>msoPermissionPrint</td>
<td>16</td>
</tr>
<tr>
<td>msoPermissionRead</td>
<td>1</td>
</tr>
<tr>
<td>msoPermissionSave</td>
<td>4</td>
</tr>
<tr>
<td>msoPermissionView</td>
<td>1</td>
</tr>
</tbody>
</table>

**MsoPictureColorType**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoPictureAutomatic</td>
<td>1</td>
</tr>
<tr>
<td>msoPictureBlackAndWhite</td>
<td>3</td>
</tr>
<tr>
<td>msoPictureGrayscale</td>
<td>2</td>
</tr>
<tr>
<td>msoPictureMixed</td>
<td>-2</td>
</tr>
<tr>
<td>msoPictureWatermark</td>
<td>4</td>
</tr>
</tbody>
</table>

**MsoPresetExtrusionDirection**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoExtrusionBottom</td>
<td>2</td>
</tr>
<tr>
<td>msoExtrusionBottomLeft</td>
<td>3</td>
</tr>
<tr>
<td>msoExtrusionBottomRight</td>
<td>1</td>
</tr>
<tr>
<td>msoExtrusionLeft</td>
<td>6</td>
</tr>
<tr>
<td>msoExtrusionNone</td>
<td>5</td>
</tr>
<tr>
<td>msoExtrusionRight</td>
<td>4</td>
</tr>
<tr>
<td>msoExtrusionTop</td>
<td>8</td>
</tr>
<tr>
<td>msoExtrusionTopLeft</td>
<td>9</td>
</tr>
<tr>
<td>msoExtrusionTopRight</td>
<td>7</td>
</tr>
<tr>
<td>msoPresetExtrusionDirectionMixed</td>
<td>-2</td>
</tr>
</tbody>
</table>
### MsoPresetGradientType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoGradientBrass</td>
<td>20</td>
</tr>
<tr>
<td>msoGradientCalmWater</td>
<td>8</td>
</tr>
<tr>
<td>msoGradientChrome</td>
<td>21</td>
</tr>
<tr>
<td>msoGradientChromeII</td>
<td>22</td>
</tr>
<tr>
<td>msoGradientDaybreak</td>
<td>4</td>
</tr>
<tr>
<td>msoGradientDesert</td>
<td>6</td>
</tr>
<tr>
<td>msoGradientEarlySunset</td>
<td>1</td>
</tr>
<tr>
<td>msoGradientFire</td>
<td>9</td>
</tr>
<tr>
<td>msoGradientFog</td>
<td>10</td>
</tr>
<tr>
<td>msoGradientGold</td>
<td>18</td>
</tr>
<tr>
<td>msoGradientGoldII</td>
<td>19</td>
</tr>
<tr>
<td>msoGradientHorizon</td>
<td>5</td>
</tr>
<tr>
<td>msoGradientLateSunset</td>
<td>2</td>
</tr>
<tr>
<td>msoGradientMahogany</td>
<td>15</td>
</tr>
<tr>
<td>msoGradientMoss</td>
<td>11</td>
</tr>
<tr>
<td>msoGradientNightfall</td>
<td>3</td>
</tr>
<tr>
<td>msoGradientOcean</td>
<td>7</td>
</tr>
<tr>
<td>msoGradientParchment</td>
<td>14</td>
</tr>
<tr>
<td>msoGradientPeacock</td>
<td>12</td>
</tr>
<tr>
<td>msoGradientRainbow</td>
<td>16</td>
</tr>
<tr>
<td>msoGradientRainbowII</td>
<td>17</td>
</tr>
<tr>
<td>msoGradientSapphire</td>
<td>24</td>
</tr>
<tr>
<td>msoGradientSilver</td>
<td>23</td>
</tr>
<tr>
<td>msoGradientWheat</td>
<td>13</td>
</tr>
<tr>
<td>msoPresetGradientMixed</td>
<td>-2</td>
</tr>
</tbody>
</table>

### MsoPresetLightingDirection

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoLightingBottom</td>
<td>8</td>
</tr>
<tr>
<td>Constant</td>
<td>Value</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>msoLightingBottomLeft</td>
<td>7</td>
</tr>
<tr>
<td>msoLightingBottomRight</td>
<td>9</td>
</tr>
<tr>
<td>msoLightingLeft</td>
<td>4</td>
</tr>
<tr>
<td>msoLightingNone</td>
<td>5</td>
</tr>
<tr>
<td>msoLightingRight</td>
<td>6</td>
</tr>
<tr>
<td>msoLightingTop</td>
<td>2</td>
</tr>
<tr>
<td>msoLightingTopLeft</td>
<td>1</td>
</tr>
<tr>
<td>msoLightingTopRight</td>
<td>3</td>
</tr>
<tr>
<td>msoPresetLightingDirectionMixed</td>
<td>-2</td>
</tr>
</tbody>
</table>

**MsoPresetLightingSoftness**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoLightingBright</td>
<td>3</td>
</tr>
<tr>
<td>msoLightingDim</td>
<td>1</td>
</tr>
<tr>
<td>msoLightingNormal</td>
<td>2</td>
</tr>
<tr>
<td>msoPresetLightingSoftnessMixed</td>
<td>-2</td>
</tr>
</tbody>
</table>

**MsoPresetMaterial**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoMaterialMatte</td>
<td>1</td>
</tr>
<tr>
<td>msoMaterialMetal</td>
<td>3</td>
</tr>
<tr>
<td>msoMaterialPlastic</td>
<td>2</td>
</tr>
<tr>
<td>msoMaterialWireFrame</td>
<td>4</td>
</tr>
<tr>
<td>msoPresetMaterialMixed</td>
<td>-2</td>
</tr>
</tbody>
</table>

**MsoPresetTextEffect**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoTextEffect1</td>
<td>0</td>
</tr>
<tr>
<td>msoTextEffect10</td>
<td>9</td>
</tr>
<tr>
<td>msoTextEffect11</td>
<td>10</td>
</tr>
<tr>
<td>msoTextEffect12</td>
<td>11</td>
</tr>
<tr>
<td>msoTextEffect13</td>
<td>12</td>
</tr>
<tr>
<td>----------------</td>
<td>----</td>
</tr>
<tr>
<td>msoTextEffect14</td>
<td>13</td>
</tr>
<tr>
<td>msoTextEffect15</td>
<td>14</td>
</tr>
<tr>
<td>msoTextEffect16</td>
<td>15</td>
</tr>
<tr>
<td>msoTextEffect17</td>
<td>16</td>
</tr>
<tr>
<td>msoTextEffect18</td>
<td>17</td>
</tr>
<tr>
<td>msoTextEffect19</td>
<td>18</td>
</tr>
<tr>
<td>msoTextEffect2</td>
<td>1</td>
</tr>
<tr>
<td>msoTextEffect20</td>
<td>19</td>
</tr>
<tr>
<td>msoTextEffect21</td>
<td>20</td>
</tr>
<tr>
<td>msoTextEffect22</td>
<td>21</td>
</tr>
<tr>
<td>msoTextEffect23</td>
<td>22</td>
</tr>
<tr>
<td>msoTextEffect24</td>
<td>23</td>
</tr>
<tr>
<td>msoTextEffect25</td>
<td>24</td>
</tr>
<tr>
<td>msoTextEffect26</td>
<td>25</td>
</tr>
<tr>
<td>msoTextEffect27</td>
<td>26</td>
</tr>
<tr>
<td>msoTextEffect28</td>
<td>27</td>
</tr>
<tr>
<td>msoTextEffect29</td>
<td>28</td>
</tr>
<tr>
<td>msoTextEffect3</td>
<td>2</td>
</tr>
<tr>
<td>msoTextEffect30</td>
<td>29</td>
</tr>
<tr>
<td>msoTextEffect4</td>
<td>3</td>
</tr>
<tr>
<td>msoTextEffect5</td>
<td>4</td>
</tr>
<tr>
<td>msoTextEffect6</td>
<td>5</td>
</tr>
<tr>
<td>msoTextEffect7</td>
<td>6</td>
</tr>
<tr>
<td>msoTextEffect8</td>
<td>7</td>
</tr>
<tr>
<td>msoTextEffect9</td>
<td>8</td>
</tr>
<tr>
<td>msoTextEffectMixed</td>
<td>-2</td>
</tr>
</tbody>
</table>

**MsoPresetTextEffectShape**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoTextEffectShapeArchDownCurve</td>
<td>10</td>
</tr>
<tr>
<td>msoTextEffectShapeArchDownPour</td>
<td>14</td>
</tr>
</tbody>
</table>
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
msoTextEffectShapeFadeDown 36
msoTextEffectShapeFadeLeft 34
msoTextEffectShapeFadeRight 33
msoTextEffectShapeFadeUp 35
msoTextEffectShapeInflate 25
msoTextEffectShapeInflateBottom 27
msoTextEffectShapeInflateTop 29
msoTextEffectShapeMixed -2
msoTextEffectShapePlainText 1
msoTextEffectShapeRingInside 7
msoTextEffectShapeRingOutside 8
msoTextEffectShapeSlantDown 38
<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoPresetTextureMixed</td>
<td>-2</td>
</tr>
<tr>
<td>msoTextureBlueTissuePaper</td>
<td>17</td>
</tr>
<tr>
<td>msoTextureBouquet</td>
<td>20</td>
</tr>
<tr>
<td>msoTextureBrownMarble</td>
<td>11</td>
</tr>
<tr>
<td>msoTextureCanvas</td>
<td>2</td>
</tr>
<tr>
<td>msoTextureCork</td>
<td>21</td>
</tr>
<tr>
<td>msoTextureDenim</td>
<td>3</td>
</tr>
<tr>
<td>msoTextureFishFossil</td>
<td>7</td>
</tr>
<tr>
<td>msoTextureGranite</td>
<td>12</td>
</tr>
<tr>
<td>msoTextureGreenMarble</td>
<td>9</td>
</tr>
<tr>
<td>msoTextureMediumWood</td>
<td>24</td>
</tr>
<tr>
<td>msoTextureNewsprint</td>
<td>13</td>
</tr>
<tr>
<td>msoTextureOak</td>
<td>23</td>
</tr>
<tr>
<td>msoTexturePaperBag</td>
<td>6</td>
</tr>
<tr>
<td>msoTexturePapyrus</td>
<td>1</td>
</tr>
<tr>
<td>msoTextureParchment</td>
<td>15</td>
</tr>
<tr>
<td>msoTexturePinkTissuePaper</td>
<td>18</td>
</tr>
<tr>
<td>msoTexturePurpleMesh</td>
<td>19</td>
</tr>
<tr>
<td>msoTextureRecycledPaper</td>
<td>14</td>
</tr>
<tr>
<td>msoTextureSand</td>
<td>8</td>
</tr>
<tr>
<td>msoTextureStationery</td>
<td>16</td>
</tr>
<tr>
<td>msoTextureWalnut</td>
<td>22</td>
</tr>
<tr>
<td>msoTextureWaterDroplets</td>
<td>5</td>
</tr>
<tr>
<td>msoTextureWhiteMarble</td>
<td>10</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----</td>
</tr>
<tr>
<td>msoTextureWovenMat</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>MsoPresetThreeDFormat</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Constant</strong></td>
</tr>
<tr>
<td>msoPresetThreeDFormatMixed</td>
</tr>
<tr>
<td>msoThreeD1</td>
</tr>
<tr>
<td>msoThreeD10</td>
</tr>
<tr>
<td>msoThreeD11</td>
</tr>
<tr>
<td>msoThreeD12</td>
</tr>
<tr>
<td>msoThreeD13</td>
</tr>
<tr>
<td>msoThreeD14</td>
</tr>
<tr>
<td>msoThreeD15</td>
</tr>
<tr>
<td>msoThreeD16</td>
</tr>
<tr>
<td>msoThreeD17</td>
</tr>
<tr>
<td>msoThreeD18</td>
</tr>
<tr>
<td>msoThreeD19</td>
</tr>
<tr>
<td>msoThreeD2</td>
</tr>
<tr>
<td>msoThreeD20</td>
</tr>
<tr>
<td>msoThreeD3</td>
</tr>
<tr>
<td>msoThreeD4</td>
</tr>
<tr>
<td>msoThreeD5</td>
</tr>
<tr>
<td>msoThreeD6</td>
</tr>
<tr>
<td>msoThreeD7</td>
</tr>
<tr>
<td>msoThreeD8</td>
</tr>
<tr>
<td>msoThreeD9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>MsoRelativeNodePosition</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Constant</strong></td>
</tr>
<tr>
<td>msoAfterLastSibling</td>
</tr>
<tr>
<td>msoAfterNode</td>
</tr>
<tr>
<td>msoBeforeFirstSibling</td>
</tr>
</tbody>
</table>
msoBeforeNode  1

### MsoScaleFrom

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoScaleFromBottomRight</td>
<td>2</td>
</tr>
<tr>
<td>msoScaleFromMiddle</td>
<td>1</td>
</tr>
<tr>
<td>msoScaleFromTopLeft</td>
<td>0</td>
</tr>
</tbody>
</table>

### MsoScreenSize

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoScreenSize1024x768</td>
<td>4</td>
</tr>
<tr>
<td>msoScreenSize1152x882</td>
<td>5</td>
</tr>
<tr>
<td>msoScreenSize1152x900</td>
<td>6</td>
</tr>
<tr>
<td>msoScreenSize1280x1024</td>
<td>7</td>
</tr>
<tr>
<td>msoScreenSize1600x1200</td>
<td>8</td>
</tr>
<tr>
<td>msoScreenSize1800x1440</td>
<td>9</td>
</tr>
<tr>
<td>msoScreenSize1920x1200</td>
<td>10</td>
</tr>
<tr>
<td>msoScreenSize544x376</td>
<td>0</td>
</tr>
<tr>
<td>msoScreenSize640x480</td>
<td>1</td>
</tr>
<tr>
<td>msoScreenSize720x512</td>
<td>2</td>
</tr>
<tr>
<td>msoScreenSize800x600</td>
<td>3</td>
</tr>
</tbody>
</table>

### MsoScriptLanguage

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoScriptLanguageASP</td>
<td>3</td>
</tr>
<tr>
<td>msoScriptLanguageJava</td>
<td>1</td>
</tr>
<tr>
<td>msoScriptLanguageOther</td>
<td>4</td>
</tr>
<tr>
<td>msoScriptLanguageVisualBasic</td>
<td>2</td>
</tr>
</tbody>
</table>

### MsoScriptLocation

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoScriptLocationInBody</td>
<td>2</td>
</tr>
<tr>
<td>msoScriptLocationInHead</td>
<td>1</td>
</tr>
</tbody>
</table>

**MsoSearchIn**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoSearchInCustom</td>
<td>3</td>
</tr>
<tr>
<td>msoSearchInMyComputer</td>
<td>0</td>
</tr>
<tr>
<td>msoSearchInMyNetworkPlaces</td>
<td>2</td>
</tr>
<tr>
<td>msoSearchInOutlook</td>
<td>1</td>
</tr>
</tbody>
</table>

**MsoSegmentType**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoSegmentCurve</td>
<td>1</td>
</tr>
<tr>
<td>msoSegmentLine</td>
<td>0</td>
</tr>
</tbody>
</table>

**MsoShadowType**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoShadow1</td>
<td>1</td>
</tr>
<tr>
<td>msoShadow10</td>
<td>10</td>
</tr>
<tr>
<td>msoShadow11</td>
<td>11</td>
</tr>
<tr>
<td>msoShadow12</td>
<td>12</td>
</tr>
<tr>
<td>msoShadow13</td>
<td>13</td>
</tr>
<tr>
<td>msoShadow14</td>
<td>14</td>
</tr>
<tr>
<td>msoShadow15</td>
<td>15</td>
</tr>
<tr>
<td>msoShadow16</td>
<td>16</td>
</tr>
<tr>
<td>msoShadow17</td>
<td>17</td>
</tr>
<tr>
<td>msoShadow18</td>
<td>18</td>
</tr>
<tr>
<td>msoShadow19</td>
<td>19</td>
</tr>
<tr>
<td>msoShadow2</td>
<td>2</td>
</tr>
<tr>
<td>msoShadow20</td>
<td>20</td>
</tr>
<tr>
<td>Constant</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>msoAutoShape</td>
<td>1</td>
</tr>
<tr>
<td>msoCallout</td>
<td>2</td>
</tr>
<tr>
<td>msoCanvas</td>
<td>20</td>
</tr>
<tr>
<td>msoChart</td>
<td>3</td>
</tr>
<tr>
<td>msoComment</td>
<td>4</td>
</tr>
<tr>
<td>msoDiagram</td>
<td>21</td>
</tr>
<tr>
<td>msoEmbeddedOLEObject</td>
<td>7</td>
</tr>
<tr>
<td>msoFormControl</td>
<td>8</td>
</tr>
<tr>
<td>msoFreeform</td>
<td>5</td>
</tr>
<tr>
<td>msoGroup</td>
<td>6</td>
</tr>
<tr>
<td>msoInk</td>
<td>22</td>
</tr>
<tr>
<td>msoInkComment</td>
<td>23</td>
</tr>
<tr>
<td>msoLine</td>
<td>9</td>
</tr>
<tr>
<td>msoLinkedOLEObject</td>
<td>10</td>
</tr>
<tr>
<td>msoLinkedPicture</td>
<td>11</td>
</tr>
<tr>
<td>msoMedia</td>
<td>16</td>
</tr>
<tr>
<td>msoOLEControlObject</td>
<td>12</td>
</tr>
<tr>
<td>msoPicture</td>
<td>13</td>
</tr>
<tr>
<td>msoPlaceholder</td>
<td>14</td>
</tr>
<tr>
<td>msoScriptAnchor</td>
<td>18</td>
</tr>
<tr>
<td>msoShapeTypeMixed</td>
<td>-2</td>
</tr>
</tbody>
</table>
### MsoSharedWorkspaceTaskPriority

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoSharedWorkspaceTaskPriorityHigh</td>
<td>1</td>
</tr>
<tr>
<td>msoSharedWorkspaceTaskPriorityLow</td>
<td>3</td>
</tr>
<tr>
<td>msoSharedWorkspaceTaskPriorityNormal</td>
<td>2</td>
</tr>
</tbody>
</table>

### MsoSharedWorkspaceTaskStatus

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoSharedWorkspaceTaskStatusCompleted</td>
<td>3</td>
</tr>
<tr>
<td>msoSharedWorkspaceTaskStatusDeferred</td>
<td>4</td>
</tr>
<tr>
<td>msoSharedWorkspaceTaskStatusInProgress</td>
<td>2</td>
</tr>
<tr>
<td>msoSharedWorkspaceTaskStatusNotStarted</td>
<td>1</td>
</tr>
<tr>
<td>msoSharedWorkspaceTaskStatusWaiting</td>
<td>5</td>
</tr>
</tbody>
</table>

### MsoSortBy

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoSortByFileName</td>
<td>1</td>
</tr>
<tr>
<td>msoSortByFileType</td>
<td>3</td>
</tr>
<tr>
<td>msoSortByLastModified</td>
<td>4</td>
</tr>
<tr>
<td>msoSortByNone</td>
<td>5</td>
</tr>
<tr>
<td>msoSortBySize</td>
<td>2</td>
</tr>
</tbody>
</table>

### MsoSortOrder

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoSortOrderAscending</td>
<td>1</td>
</tr>
<tr>
<td>msoSortOrderDescending</td>
<td>2</td>
</tr>
<tr>
<td><strong>MsoSyncAvailableType</strong></td>
<td></td>
</tr>
<tr>
<td>--------------------------</td>
<td>---</td>
</tr>
<tr>
<td>Constant</td>
<td>Value</td>
</tr>
<tr>
<td>msoSyncAvailableAnywhere</td>
<td>2</td>
</tr>
<tr>
<td>msoSyncAvailableNone</td>
<td>0</td>
</tr>
<tr>
<td>msoSyncAvailableOffline</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>MsoSyncCompareType</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>Value</td>
</tr>
<tr>
<td>msoSyncCompareAndMerge</td>
<td>0</td>
</tr>
<tr>
<td>msoSyncCompareSideBySide</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>MsoSyncConflictResolutionType</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>Value</td>
</tr>
<tr>
<td>msoSyncConflictClientWins</td>
<td>0</td>
</tr>
<tr>
<td>msoSyncConflictMerge</td>
<td>2</td>
</tr>
<tr>
<td>msoSyncConflictServerWins</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>MsoSyncErrorType</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Constant</td>
<td>Value</td>
</tr>
<tr>
<td>msoSyncErrorCouldNotCompare</td>
<td>13</td>
</tr>
<tr>
<td>msoSyncErrorCouldNotConnect</td>
<td>2</td>
</tr>
<tr>
<td>msoSyncErrorCouldNotOpen</td>
<td>11</td>
</tr>
<tr>
<td>msoSyncErrorCouldNotResolve</td>
<td>14</td>
</tr>
<tr>
<td>msoSyncErrorCouldNotUpdate</td>
<td>12</td>
</tr>
<tr>
<td>msoSyncErrorFileInUse</td>
<td>6</td>
</tr>
<tr>
<td>msoSyncErrorFileNotFound</td>
<td>4</td>
</tr>
<tr>
<td>msoSyncErrorFileTooLarge</td>
<td>5</td>
</tr>
<tr>
<td>msoSyncErrorNone</td>
<td>0</td>
</tr>
<tr>
<td>msoSyncErrorNoNetwork</td>
<td>15</td>
</tr>
<tr>
<td>msoSyncErrorOutOfSpace</td>
<td>3</td>
</tr>
<tr>
<td>Constant</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>msoSyncErrorUnauthorizedUser</td>
<td>1</td>
</tr>
<tr>
<td>msoSyncErrorUnknown</td>
<td>16</td>
</tr>
<tr>
<td>msoSyncErrorUnknownDownload</td>
<td>10</td>
</tr>
<tr>
<td>msoSyncErrorUnknownUpload</td>
<td>9</td>
</tr>
<tr>
<td>msoSyncErrorVirusDownload</td>
<td>8</td>
</tr>
<tr>
<td>msoSyncErrorVirusUpload</td>
<td>7</td>
</tr>
</tbody>
</table>

### MsoSyncEventType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoSyncEventDownloadFailed</td>
<td>2</td>
</tr>
<tr>
<td>msoSyncEventDownloadInitiated</td>
<td>0</td>
</tr>
<tr>
<td>msoSyncEventDownloadNoChange</td>
<td>6</td>
</tr>
<tr>
<td>msoSyncEventDownloadSucceeded</td>
<td>1</td>
</tr>
<tr>
<td>msoSyncEventOffline</td>
<td>7</td>
</tr>
<tr>
<td>msoSyncEventUploadFailed</td>
<td>5</td>
</tr>
<tr>
<td>msoSyncEventUploadInitiated</td>
<td>3</td>
</tr>
<tr>
<td>msoSyncEventUploadSucceeded</td>
<td>4</td>
</tr>
</tbody>
</table>

### MsoSyncStatusType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoSyncStatusConflict</td>
<td>4</td>
</tr>
<tr>
<td>msoSyncStatusError</td>
<td>6</td>
</tr>
<tr>
<td>msoSyncStatusLatest</td>
<td>1</td>
</tr>
<tr>
<td>msoSyncStatusLocalChanges</td>
<td>3</td>
</tr>
<tr>
<td>msoSyncStatusNewerAvailable</td>
<td>2</td>
</tr>
<tr>
<td>msoSyncStatusNoSharedWorkspace</td>
<td>0</td>
</tr>
<tr>
<td>msoSyncStatusSuspended</td>
<td>5</td>
</tr>
</tbody>
</table>

### MsoSyncVersionType

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoSyncVersionLastViewed</td>
<td>0</td>
</tr>
<tr>
<td>Constant</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>msoSyncVersionServer</td>
<td>1</td>
</tr>
</tbody>
</table>

### MsoTargetBrowser

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoTargetBrowserIE4</td>
<td>2</td>
</tr>
<tr>
<td>msoTargetBrowserIE5</td>
<td>3</td>
</tr>
<tr>
<td>msoTargetBrowserIE6</td>
<td>4</td>
</tr>
<tr>
<td>msoTargetBrowserV3</td>
<td>0</td>
</tr>
<tr>
<td>msoTargetBrowserV4</td>
<td>1</td>
</tr>
</tbody>
</table>

### MsoTextEffectAlignment

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoTextEffectAlignmentCentered</td>
<td>2</td>
</tr>
<tr>
<td>msoTextEffectAlignmentLeft</td>
<td>1</td>
</tr>
<tr>
<td>msoTextEffectAlignmentLetterJustify</td>
<td>4</td>
</tr>
<tr>
<td>msoTextEffectAlignmentMixed</td>
<td>-2</td>
</tr>
<tr>
<td>msoTextEffectAlignmentRight</td>
<td>3</td>
</tr>
<tr>
<td>msoTextEffectAlignmentStretchJustify</td>
<td>6</td>
</tr>
<tr>
<td>msoTextEffectAlignmentWordJustify</td>
<td>5</td>
</tr>
</tbody>
</table>

### MsoTextOrientation

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoTextOrientationDownward</td>
<td>3</td>
</tr>
<tr>
<td>msoTextOrientationHorizontal</td>
<td>1</td>
</tr>
<tr>
<td>msoTextOrientationHorizontalRotatedFarEast</td>
<td>6</td>
</tr>
<tr>
<td>msoTextOrientationMixed</td>
<td>-2</td>
</tr>
<tr>
<td>msoTextOrientationUpward</td>
<td>2</td>
</tr>
<tr>
<td>msoTextOrientationVertical</td>
<td>5</td>
</tr>
<tr>
<td>msoTextOrientationVerticalFarEast</td>
<td>4</td>
</tr>
</tbody>
</table>

### MsoTextureType

<table>
<thead>
<tr>
<th>Constant</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>msoTextureType</td>
<td></td>
</tr>
<tr>
<td>Constant</td>
<td>Value</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>--------</td>
</tr>
<tr>
<td>msoTexturePreset</td>
<td>1</td>
</tr>
<tr>
<td>msoTextureTypeMixed</td>
<td>-2</td>
</tr>
<tr>
<td>msoTextureUserDefined</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoCTrue</td>
<td>1</td>
</tr>
<tr>
<td>msoFalse</td>
<td>0</td>
</tr>
<tr>
<td>msoTriStateMixed</td>
<td>-2</td>
</tr>
<tr>
<td>msoTriStateToggle</td>
<td>-3</td>
</tr>
<tr>
<td>msoTrue</td>
<td>-1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoAnchorBottom</td>
<td>4</td>
</tr>
<tr>
<td>msoAnchorBottomBaseLine</td>
<td>5</td>
</tr>
<tr>
<td>msoAnchorMiddle</td>
<td>3</td>
</tr>
<tr>
<td>msoAnchorTop</td>
<td>1</td>
</tr>
<tr>
<td>msoAnchorTopBaseline</td>
<td>2</td>
</tr>
<tr>
<td>msoVerticalAnchorMixed</td>
<td>-2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoWizardActActive</td>
<td>1</td>
</tr>
<tr>
<td>msoWizardActInactive</td>
<td>0</td>
</tr>
<tr>
<td>msoWizardActResume</td>
<td>3</td>
</tr>
<tr>
<td>msoWizardActSuspend</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoWizardMsgType</td>
<td></td>
</tr>
<tr>
<td>Constant</td>
<td>Value</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>msoWizardMsgLocalStateOff</td>
<td>2</td>
</tr>
<tr>
<td>msoWizardMsgLocalStateOn</td>
<td>1</td>
</tr>
<tr>
<td>msoWizardMsgResuming</td>
<td>5</td>
</tr>
<tr>
<td>msoWizardMsgShowHelp</td>
<td>3</td>
</tr>
<tr>
<td>msoWizardMsgSuspending</td>
<td>4</td>
</tr>
</tbody>
</table>

**MsoZOrderCmd**

<table>
<thead>
<tr>
<th>Constant</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>msoBringForward</td>
<td>2</td>
</tr>
<tr>
<td>msoBringInFrontOfText</td>
<td>4</td>
</tr>
<tr>
<td>msoBringToFront</td>
<td>0</td>
</tr>
<tr>
<td>msoSendBackward</td>
<td>3</td>
</tr>
<tr>
<td>msoSendBehindText</td>
<td>5</td>
</tr>
<tr>
<td>msoSendToBack</td>
<td>1</td>
</tr>
</tbody>
</table>
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.